**Data Encryption Standard (DES)**

Best-known symmetric cryptography method: **DES**

- **1973**: Call for a public cryptographic algorithm standard for commercial purposes by the National Bureau of Standards
  - **Goals**: high security degree, easy to understand, cost-effective to implement
- **1982**: DES developed by the National Bureau of Standards, IBM, the National Security Agency, the Institute of Standardisation and Technology, and the American National Standardisation Institute
  - **Usage**: e.g. for computation of secret numbers for ec-cards
- **All 5 years**: DES review for decision about further usage
- **Result**: Until now, no modifications were made

**Problem of DES:**
- Usage of a **key** with a **length of 56 bit**
  - Criticised for a key length too short for usage in practice
  - January 1999: DES key was broken by a Brute Force attack within 22 hours

**General Structure of DES**

- DES uses blocks of length \( n = 64 \) bit
- Length of key \( k_{\text{DES}} \) is 64 bit
  - (but: only usage of 56 bit of this key, one bit in each byte is used for odd parity)
- Encryption takes place in 16 identical rounds with round keys \( k_i \) of 48 bit length

**Encryption process**

1. **step**: permutation performed on the input block
   - Purpose of permutation: has no security means; but: simple implementations become slower with the permutation, hardware solutions are needed
   - designers want to control the availability of DES?
2. **step**: generation of round keys
3. **step**: performing 16 identical rounds
4. **step**: inverse permutation to step 1
   - (when applying DES several times, the effects of both permutations are compensated in between)

**DES Encryption Process**

1. **Initial permutation**
2. **Generation of round keys**
3. **Encryption in 16 identical rounds**:
   - a. **Substitution**
   - b. **Permutation**
4. **Final permutation**
   - Additional step: swap left and right halves

**Step 1 and 4: Input and Output Permutations**

**Input permutation (IP)**

<table>
<thead>
<tr>
<th>58</th>
<th>50</th>
<th>42</th>
<th>34</th>
<th>26</th>
<th>18</th>
<th>10</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>64</td>
<td>56</td>
<td>48</td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
<td>21</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

**Output permutation (IP\(^{-1}\))**

<table>
<thead>
<tr>
<th>41</th>
<th>8</th>
<th>48</th>
<th>16</th>
<th>56</th>
<th>24</th>
<th>64</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>7</td>
<td>47</td>
<td>15</td>
<td>55</td>
<td>23</td>
<td>63</td>
<td>31</td>
</tr>
<tr>
<td>38</td>
<td>6</td>
<td>46</td>
<td>14</td>
<td>54</td>
<td>22</td>
<td>62</td>
<td>30</td>
</tr>
<tr>
<td>37</td>
<td>5</td>
<td>45</td>
<td>13</td>
<td>53</td>
<td>21</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>44</td>
<td>12</td>
<td>52</td>
<td>20</td>
<td>60</td>
<td>28</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>43</td>
<td>11</td>
<td>51</td>
<td>19</td>
<td>59</td>
<td>27</td>
</tr>
<tr>
<td>34</td>
<td>2</td>
<td>42</td>
<td>10</td>
<td>50</td>
<td>18</td>
<td>58</td>
<td>26</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>41</td>
<td>9</td>
<td>49</td>
<td>17</td>
<td>57</td>
<td>25</td>
</tr>
</tbody>
</table>

**Input permutation**

- See each 64-bit block as 8 Bytes, arranged in a matrix
- Diffusion of bits over all bytes
  - Bits of a column are packed into a row
  - First byte is spread into 8th bits of all bytes
  - Second byte is spread into 7th bits of all bytes
  - ...
### Step 2: Generation of Round Keys (1)

**Preparation:**
- Divide $k_{DES}$ into $C_0$ and $D_0$ (each 28 bit long - no parity bits) by performing permutations similar to DES initial permutation (which has no security value).

**Now:**
- Round keys $k_i$ are computed in 16 rounds from $C_i$ and $D_i$:
  - $C_{i-1} D_{i-1}$ left shift
  - $D_{i} C_{i}$ left shift

### Step 2: Generation of Round Keys (2)

**Characteristics of key generation**
- Left shift:
  - Round 1, 2, 9, and 16: left shift of 1 bit
  - Other rounds: left shift of 2 bits
  - Notice: 10 years later it was found, that performing the left shift with varying step sizes makes the algorithm more secure
- Left half of $k_i$ is only determined by $C_i$, right side only by $D_i$
- Permutations:
  - $C_i$: bits 9, 18, 22, and 25 are discarded (remaining: 24 bits)
  - $D_i$: bits 35, 38, 43, and 54 are discarded (remaining: 24 bits)
  - Perform permutations on remaining bits of $C_i$ resp. $D_i$ to obtain $k_i$ (48 bits)
  - Notice: the choice of the permutations on $C_i$ resp. $D_i$ influence the security of DES, because they determine the quality of the round keys

### Step 3: One DES Round

- Divide input block into two 32-bit blocks $L_i$ and $R_i$
- Compute $L_{i+1}$ as $R_i$ and $R_{i+1}$ as $L_i \oplus f(R_i, K_i)$
  - $f$ is cipher function, i.e. combination of substitution and permutation
  - Security provided by DES depends on the quality of the cipher function
  - Decryption: uses the same algorithm, has same expense like encryption

### Step 3: Cipher Function - Substitution

- Expand 32-bit input block $R_i$ to a 48 input block $R_i'$
- Divide 32-bit input block into 8 chunks of 4 bit
- Expansion: enhance chunks of 4 bit with their neighbour bits to 6 bit
- Divide round key $k_i$ into 8 chunks of 6 bit
- Perform XOR operation on $R_i'$ and $k_i$ chunks
- Use resulting chunks as input for S-Boxes
Step 3: Cipher Function - S-Boxes for Substitution

**S-Box**
- Table for exchanging one 6-bit chunk with a 4-bit cipher block
- Each 4 input values map to the same output value (one in each row)
- First and last bit of a chunk determine a substitution row in a S-Box
- Overall 8 S-Boxes, one for each chunk
- Outputs of S-Boxes are combined into a 32 bit block

S-Box for the first chunk:

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0111</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1110</td>
<td>0100</td>
<td>1101</td>
<td>0001</td>
<td>0010</td>
<td>------</td>
<td>0111</td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>1111</td>
<td>0111</td>
<td>0100</td>
<td>1110</td>
<td>------</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
<td>0001</td>
<td>1110</td>
<td>1000</td>
<td>1101</td>
<td>------</td>
<td>0000</td>
</tr>
<tr>
<td>11</td>
<td>1111</td>
<td>1100</td>
<td>1000</td>
<td>0010</td>
<td>0100</td>
<td>------</td>
<td>1101</td>
</tr>
</tbody>
</table>

Step 3: S-Boxes

- **Choice of S-Boxes influences quality of the DES algorithm**
  - When exchanging e.g. S-Boxes 3 and 7, the security is distinctly restricted
  - Chosen S-Boxes: compromise of resistance against known attacks and low costs for hardware realisation
- **Design criteria given by IBM**
  - Output function should be relatively complex
  - If two input chunks \( j_1, ..., j_6 \) and \( j'_1, ..., j'_6 \) to an S-Box differ in exactly one bit, the outputs should differ in a minimum of two bits
  - If \( j_3 \neq j'_3, j_4 \neq j'_4, \) and \( j_i = j'_i \) else, the outputs should differ in a minimum of two bits
  - For all input chunks with the same 6-bit-difference (there always are 32 input chunks with a common difference), a maximum of 8 of them should have the same difference in the output

Step 3: Cipher Function

After applying S-Boxes: **permutation step**
- Ensure that bits of the output of an S-Box on one round affects the input of multiple S-Boxes in the next round:
  - Two of the output bits of one S-Box should influence the middle of the result, the other two bits should influence the edges
  - The 4 output bits should form the input of 6 S-Boxes in the next round

**Security of DES**
- DES is seen as very secure (except for the key length)
- No attacks with lower costs than a Brute Force attack are known as far
- There are some so-called weak keys and semi-weak keys
  - These keys should not be used!

**Questions on DES**
- Design process for DES was not public
- Are details well-chosen for strength of the DES algorithm?
- Are some weaknesses useful for people involved in the design process?
- Are there other weak keys than the known ones?

Multiple Application of DES

- **Short key in DES (56 bit) is unsuitable**
  - Repeated application of DES to increase key length
- **Double application of DES:**
  - Suitable key length of 112 bit
- But: attack is known which needs 257 steps (~ Brute Force attack for DES) if only 3 or 4 \(<\text{message, cipher}>\)-pairs are known (without knowing the two DES keys \(<k_1, k_2>\):
  - Assume: \(<m, c>\) is a known pair
  - Search for a key pair \(<k_1, k_2>\) matching on \(<m, c>\)
  - Create one table with entries for each possible \(k_1: <m, t_0>\)
  - Create one table with entries for each possible \(k_2: <t_0, c>\)
  - Search for rows with \(t_0 = t_0\)
  - Such a row is a candidate for a key pair \(<k_1, k_2>\)
  - Repeat procedure by testing other known blocks \(<m, c>\)
  - Usually, 3 or 4 test are enough to determine one key pair \(<k_1, k_2>\)
### Triple-DES

- The more applications of DES are made, the more cost is caused
  - Generally used: **Triple-DES**
  - No attack is known for Triple-DES
- Generally accepted method in Triple-DES: **EDE** (encrypt - decrypt - encrypt)
  - Equivalent to using two cipher functions E and D with order EDE
  - Only two keys $k_1$ (in E) and $k_2$ (in D) are used
  - 112 bit key length is enough, not necessary to generate and transmit a third key
- Why using decryption $D$ in the second step?
  - Using $k_1 = k_2$, the second application cancels the first one, and standard DES results (interoperation with standard DES system)
  - In usage of EEE, the initial permutation of one E would redo the end permutation of the previous E:

\[
\text{perm}_{\text{init}} \rightarrow \text{DES rounds} \rightarrow \text{perm}_{\text{end}} \rightarrow \text{DES rounds} \rightarrow \text{perm}_{\text{end}}
\]

### International Data Encryption Algorithm (IDEA)

- Alternative to DES: **IDEA**
- Symmetric cryptography method similar to DES
- Developed 1991 at the ETH Zürich by Lai and Massey
- Goal: create a cryptography algorithm efficiently computable in software, more secure than DES

#### Characteristics
- Usage of a **key** with a **length of 128 bit**
- Operating on blocks of 64 bit length
- Like in DES, based on rounds with a cipher function
- Build from 17 rounds, distinguished between even and odd rounds
  - Odd rounds each use 4 round keys
  - Even rounds each use 2 round keys

### General Structure of IDEA

1. **Key generation**
   - 52 keys of 16 bit length are generated
     - $k_{\text{IDEA}}$ is divided into 8 subkeys $k_1$ to $k_8$, each 16 bit long
     - $k_{\text{IDEA}}$ is shifted left by 25 bit and divided into the next 8 keys
     - The last step is repeated, until 52 keys are generated
   - Division of generated keys into round keys
     - Odd round: use mathematical inverse of the encryption round keys
     - Even round: use same keys as in encryption, because the even round is its own reverse
2. **Division of input block into 4 16-bit blocks**
3. **Encryption in 17 rounds:**
   - a. even rounds
   - b. odd rounds

### Key Generation in IDEA

- 52 keys of 16 bit length are generated
  - $k_{\text{IDEA}}$ is divided into 8 subkeys $k_1$ to $k_8$, each 16 bit long
  - $k_{\text{IDEA}}$ is shifted left by 25 bit and divided into the next 8 keys
  - The last step is repeated, until 52 keys are generated
- Division of generated keys into round keys
  - Round one is assigned with $k_1$ to $k_4$
  - Round two is assigned with $k_5$ to $k_6$
  - In general: even round $r$ gets keys $k_{3r-1}$ to $k_{3r+1}$
  - odd round $r$ gets keys $k_{3r+2}$ to $k_{3r+4}$
  - Notice: in the specification, keys 50 and 51 are exchanged
- Keys in decryption process
  - Uses same algorithm like encryption
  - Odd round: use mathematical inverse of the encryption round keys
  - Even round: use same keys as in encryption, because the even round is its own reverse
One Round in IDEA

- Divide input block into 16-bit blocks $X_a$, $X_b$, $X_c$, $X_d$
- In odd round, use keys $k_a$, $k_b$, $k_c$, $k_d$
- In even round, use keys $k_e$, $k_f$
- Use three operations for transforming 16 bit on another 16 bit
  → XOR ($\oplus$)
  → Addition modulo $2^{16}$ (+)
  → Multiplication modulo $2^{16} + 1$ ($\otimes$)
  Notice: this operation is reversible, with one modification:
  - 0 has no inverse in multiplication
  - $2^{16}$ is in the range of $mod 2^{16} + 1$, but not expressible in 16 bits
  → define 0 (i.e. 00...00) to be $2^{16}$

Odd Round

Even Round

IDEA vs. DES

Advantages of IDEA
- Key length 128 bit is more resistant against Brute Force attacks
- Less attacks on IDEA known
- Not using 'mysterious' S-Boxes and permutations
  - DES has weak keys, i.e. both halves only consist of successions like 0....0, 1....1, 0101....01 or 1010....10
  - Such keys are not known for IDEA

Decision criteria
- DES is more suitable for hardware realisation
- IDEA is more suitable for software realisation
Advanced Encryption Standard (AES)

National Institute of Standards and Technology (NIST), 1997: Call for an encryption algorithm as successor of DES

Requirements for AES

→ Definition of a public available symmetric block encryption algorithm
→ Variable key length of 128, 192, and 256 bit

Final proposals

• MARS - IBM
• RC6 - RSA Laboratories
• RIJNDAEL - Joan Daemen, Vincent Rijmen
• Serpent - Ross Anderson, Eli Biham, Lars Knudsen
• Twofish - Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

RIJNDAEL was selected as AES

Design Criteria of AES

→ Resistance against all known attacks
→ Speed and code compactness on a wide range of platforms
→ Design simplicity

AES design

• AES consists of
  1. An initial round key addition
  2. A variable number of rounds
  3. One final round

• Round transformation - not as in most ciphers - is composed of 3 distinct invertible uniform transformations (called layers)
  (uniform = every bit is treated in a similar way)
  → Linear mixing layer: achieve a high bit diffusion over multiple rounds
  → Non-linear layer: perform a parallel application of S-Boxes that have optimal worst-case non-linearity properties
  → Key addition layer: perform XOR of the round key and the intermediate state

AES Characteristics

• Initial round key addition is performed for security reasons:
  Without key addition, the following actions could be peeled off without knowledge of the key
• AES is a block cipher with variable block length
• Key length can independently specified as 128, 192 or 256 bit
• A cipher key can be seen as an array with 4 rows. The number of columns is denoted by $N_k$ and is equal to the key length divided by 32
• An intermediate block is called State. It can be seen as array with 4 rows and $N_b$ columns, and is equal to the block length divided by 32

$N_b = 6$
$N_k = 4$

<table>
<thead>
<tr>
<th>$a_{0,0}$</th>
<th>$a_{0,1}$</th>
<th>$a_{0,2}$</th>
<th>$a_{0,3}$</th>
<th>$a_{0,4}$</th>
<th>$a_{0,5}$</th>
<th>$a_{0,6}$</th>
<th>$a_{0,7}$</th>
<th>$a_{0,8}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{0,0}$</td>
<td>$b_{0,1}$</td>
<td>$b_{0,2}$</td>
<td>$b_{0,3}$</td>
<td>$b_{0,4}$</td>
<td>$b_{0,5}$</td>
<td>$b_{0,6}$</td>
<td>$b_{0,7}$</td>
<td>$b_{0,8}$</td>
</tr>
</tbody>
</table>

input and output: one-dimensional arrays of 8-bit bytes. Block length: 16, 24, or 32 bytes
(same for cipher key)

state cipher key

Generation of Round Keys in AES

• Each round key is derived from the cipher key by key schedule:
  1. Key expansion
     → An expanded key is an array of 4-byte words $w_i$
     → The first $N_b$ words contain the cipher key
     → All following words are composed of the previous word XOR'ed with the word $N_b$ positions earlier
     → For words in positions of multiple of $N_b$, a round constant is XOR'ed
  2. Key selection
     → The round key $k_i$ is given by the words from $N_b \cdot i$ to $N_b \cdot (i+1)$

• Total number of round keys: with $m$ rounds and blocks of length $n$: generate $n \cdot (m+1)$ keys

![Diagram of round key generation](image)
Rounds in AES

→ The number of rounds $N_r$ depends on $N_b$ and $N_k$

<table>
<thead>
<tr>
<th>$N_r$</th>
<th>$N_b$</th>
<th>$N_k$</th>
<th>$N_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
<td>14</td>
</tr>
</tbody>
</table>

→ Each round transformation is composed of four transformations:
  - ByteSub
  - ShiftRow
  - MixColumn
  - AddRoundKey

→ Exception in the final round:
  - No MixColumn is performed

Bytesub, MixColumn, and AddRoundKey Transformation

ByteSub transformation
→ Non-linear byte substitution
→ Operates on each of the State bytes independently
→ Invertible S-Boxes are used as substitution table

MixColumn transformation
→ Columns of the State are considered as polynomials and multiplied with a fixed polynomial $c(x) = '03' x^3 + '01' x^2 + '01' x + '02' \mod x^4 + 1$
→ Can be written as matrix multiplication

AddRoundKey transformation
→ Round key is applied to the State by a bitwise xor
→ Round key length is equal to the block length $N_b$

ShiftRow Transformation

→ Rows of the State are cyclically shifted by different offsets:
  - Row 0: not shifted
  - Row 1: shifted by $C_1$ bytes
  - Row 2: shifted by $C_2$ bytes
  - Row 3: shifted by $C_3$ bytes

→ Offsets $C_1$, $C_2$, $C_3$ depend on $N_b$

Example: $N_b = 6$ → $C_1 = 1$, $C_2 = 2$, $C_3 = 3$

Characteristics of AES

- **Strength against former attacks (but not against new attacks...)**
  → Care has been taken to eliminate symmetry in the behaviour of the cipher obtained by round constants different for each round
- **Advantages of the algorithm:**
  → Implementation runs fast for a block cipher (trade-off between table size and performance)
  → Can be implemented on a smart card
  → Round transformation can be performed in parallel
  → Extendable design: each key length in 32 bit steps is possible, number of rounds can be extended
- **Can be used as self-synchronising stream cipher**
- **Limitations:**
  → Decryption needs more code and cycles than encryption, as well as other tables than the cipher
Chapter 2.1: Secret Key Cryptography

Encryption of Long Messages

- Encrypt long messages (streams) with block ciphers:
  - Usual message has much more than 64 bits, i.e. consists of several blocks
- Defined for usage with DES
- Applicable on all block cipher algorithms using fixed-length blocks

Requirements

1.) The receiver should be able to detect the integrity of the block sequence i.e. should be able to find out whether a block has been changed or inserted/deleted, or whether a permutation has been done
2.) An attacker should be not able to detect the "natur" of the message

Division of messages for coding with DES or IDEA (64 bit blocks)

1. Electronic Code Book (ECB)
2. Cipher Block Chaining (CBC)
3. k-bit Cipher Feedback Mode (CFB)
4. k-bit Output Feedback Mode (OFB)

Electronic Code Book (ECB)

- Simplest method: divide a message into blocks and encrypt each one
- Disadvantage: Too simple, too dangerous; does not satisfy the requirements
  - Identical blocks \( m_i \) are encrypted to the same cipher block \( c_i \) and can be identified by an attacker
  - The message structure can be identified
  - If the attacker knows, what context the plaintext has, parts of message can be manipulated
- Advantage: fast access to single blocks

Decryption in CBC

Decryption looks similar to encryption:

- For each message to be encoded, a new IV should be used
- Usage of the same IV for all messages would cause some problems:
  - First differences in similar messages can be found by an attacker
  - Old messages can be sent by an attacker at a later time
  - Chosen plaintext can be applied as an attack
- CBC is often used in combination with Triple-DES
Two Alternatives for Combining CBC and Triple-DES

CBC on the outside

- More common alternative
- More dangerous (attacks!) than CBC on the inside

CBC on the inside

- Looks complicated but performs better, because two "half"-operations can be done in parallel

Characteristics of CBC

- What happens if an attacker changes a block, e.g. $C_3$?
  - $m_1$ will be changed (when decrypted) in a predictable way
  - $m_1$ will be changed in a "random" way
  - Solution: attach a 32-bit CRC to the plaintext before encryption

- What happens if an attacker does some permutation of the $c_i$?
  - If the attacker knows $m_1$, ..., $m_n$ and $c_1$, ..., $c_n$ he can produce any new plaintext which is a permutation of $m_1$, ..., $m_n$ (because $d_k(c_i) = m_i \oplus c_{i-1}$)
  - Solution: also compute a CRC before encryption
    - The attacker could construct $c_{2n}$, ..., $c_{mn}$ plus the corresponding plaintext until the CRC fits
    - Solution: only compute a longer CRC
    - In practice, using a 64-bit CRC would suffice, only a Brute Force attack would be possible

Output Feedback Mode (OFB)

- Acts like a pseudo-random number generator
- Perform a XOR-operation on a message and a pseudo-random stream
  - One-time pad
- Generation of a pseudo-random stream
  - Cipher algorithm together with a shift register is used as random number generator
  - For a block length of $n < 64$, only use the first $n$ bits of the encrypted IV
  - For each next pseudo-random number, discard the first $n$ bits of IV and append the $n$ bits used before
  - Only IV must be transmitted

OFB - Simple Version

- Use an IV of the same length as the block length (= 64 bit)
- No part of the vector is discarded, each random number can be generated very easy without using a shift register: $e_k(IV), e_k(e_k(IV)), e_k(e_k(e_k(IV))), ...$
Characteristics of OFB

Advantages of OFB

- Random numbers $b_0, b_1, b_2, \ldots$ can be generated in advance
- No costly coding operation
- If some blocks of the message are garbled, only single blocks of the cipher text are also garbled
- Blocks of any length can be encoded (For example: blocks of length 8, i.e. one character)

Disadvantage of OFB

- If a block is lost or inserted, the following message can't be decrypted (but the receiver could try to place a mask on the received message blocks in order to find insertions or deletions)
- An attacker who knows plaintext and cipher text, can obtain a string with which he could encrypt own messages

OFB is an example of using a block cipher as synchronous stream cipher

Cipher Feedback Mode (CFB)

- Similar to OFB
  - Encrypt an IV with the key $k$
  - Use $n$ bits of the encrypted string for a XOR-operation with $n$ bits of plain text
- Difference to OFB
  - $n$ bits shifted in are taken from the cipher text of the previous block, not from the encrypted IV

Characteristics of CFB

Advantages of CFB

- Errors in cipher text causes errors only in the next $n$ plaintext blocks (CFB is self-resynchronising)
- Best suited for serial transmission of data, because lost or added words are only causing a short error
- CFB is more secure than OFB, because the cipher text is used for generating the random numbers
- No re-arrangement of blocks is possible
- Blocks of any length can be encoded (like in OFB)

Disadvantage of CFB

- Random numbers cannot be computed in advance; each block of plaintext needs a DES operation - a lot of calculation

Usage of CFB

- In general used for words of 8 bit
- CFB detects alterations better than OFB but not as well as CBC

CFB is an example for using a block cipher as self-synchronising stream cipher
**Message Integrity Code (MIC)**

*Purpose:* guarantee integrity of a message  
- Use e.g. CBC to encrypt a message  
- Use the last block of the cipher text as a *message integrity code*. This last block is called *CBC residue*.  
- Transmit the plaintext together with the MIC (as a kind of signature), don't transmit $c_1, ..., c_{r-1}$.

\[ \begin{align*}  
IV & \oplus m_1 \to c_1 \\
& \oplus m_2 \to c_2 \\
& \oplus m_3 \to c_3 \\
& \oplus m_4 \to c_4 \\
& \oplus m_5 \to c_5 \\
& \oplus m_6 \to k \\
\end{align*} \]

**Guaranteeing Privacy and Integrity**

- **Privacy:** encrypt message using CBC  
- **Integrity:** use CBC residue as MIC  
- **Problem:** how to guarantee integrity for a CBC-encoded message?  
- **Solution:** use different keys for encryption and generation of the MIC:  
  1.) Use two independent keys  
  2a.) Only *change one bit* in the encryption key for generating the MIC  
  2b.) Generate the MIC key $k_2$ from the encryption key $k_1$ by  
      \[ k_2 = k_1 \oplus \text{F0F0F0F0F0F0F0F0}_{16} = 1111000011110000....11110000 \]  
      → This method *preserves the key parity*. It never changes a non-weak key into a weak key (as possible in 2a)  
      → This method is practised by KERBEROS V  
  3.) Compute a hash value $h(m_1, ..., m_n)$ of 128 or 192 bit  
      Add it to the message, i.e. enhance it by 2 or 3 more blocks  
      Encrypt the enhanced message with CBC