UNIT III

BLOCK CIPHERS
BLOCK CIPHER PRINCIPLES

• Stream Cipher – encrypts one bit or one byte at a time
  – Ex: Vigenere cipher, Vernam Cipher

• Block Cipher – block of plain text is treated as a whole and produce the cipher text
  – Block size 64 or 128
Motivation for Feistel Cipher Structure

• A block operates on n-bits of PT to produce n-bits of cipher text
• $2^n$ possible PT blocks for encryption to be reversible to produce a unique cipher text
• Such transformation is non-singular or reversible
• In reversible mapping the no. of different transformations is $2^n$!
IDEAL BLOCK CIPHER

Diagram showing a process involving a 4-bit input, a 4 to 16 decoder, and a 16 to 4 encoder, followed by a 4-bit output.
IDEAL BLOCK CIPHER

• A 4-bit input produces 16 possible i/p states which is mapped by the substitution cipher into unique one of the possible o/p states

• Practical problems
  – Easy to build for small block size say n=4, but vulnerable to attacks
  – If n large, mapping of PT to CT is allowed, statistical characteristics of the mapping is masked, but infeasible
FEISTEL CIPHER

• Feistel approximated ideal block cipher with the concept of product cipher which make encryption stronger

• i.e develop a block cipher with a key length of $k$-bits and a block length of $n$-bits, allowing $2^k$ possible transformations rather than $2^n!$ transformations
Proposed the use of cipher that alternates substitutions and permutations (proposal of Shannon) to develop a product cipher that alternates *confusion* and *diffusion*.

*Confusion*: makes relationship between ciphertext and key as complex as possible.

*Diffusion*: dissipates statistical structure of plaintext over bulk of ciphertext.
FEISTEL CIPHER STRUCTURE
• PT of block length $2w$ bits and key $K$ is given as i/p to the encryption algorithm
• PT block is divided into two halves $L_o$ and $R_o$
• These two halves pass through $n$ rounds and then combine to form cipher text
• Each round $i$ has inputs as $L_{i-1}$ and $R_{i-1}$ derived from previous round, as well as sub key $K_i$ derived from the overall $K$. 
FEISTEL CIPHER STRUCTURE

\[ L_i = R_{i-1} \]
\[ R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \]

- Substitution on left half of data( F(\( R_{i-1}, K_i \)) )
- Permutation consists of interchange of two halves of data
- This is a particular form of substitution-permutation network (SPN) proposed by Shannon.
FEISTEL CIPHER DESIGN ELEMENTS

- block size
- key size
- number of rounds
- subkey generation algorithm
- round function
- fast software en/decryption
- ease of analysis
FEISTEL DECRYPTION ALGORITHM
DATA ENCRYPTION STANDARD (DES)

• most widely used block cipher in world
• adopted in 1977 by NBS (now NIST)
  – as FIPS PUB 46
• encrypts 64-bit data using 56-bit key
• has widespread use
• has been considerable controversy over its security
• Diffusion is achieved through numerous permutations and confusions is achieved through the XOR operation and the S-Boxes.
DES (History)

- IBM developed Lucifer cipher
  - by team led by Feistel in late 60’s
  - used 64-bit data blocks with 128-bit key
- then redeveloped as a commercial cipher with input from NSA and others
- in 1973 NBS issued request for proposals for a national cipher standard
- IBM submitted their revised Lucifer which was eventually accepted as the DES
DES ENCRYPTION

1. Initial Permutation
2. Round 1
   - K1
   - Permuted Choice 2
   - Left circular shift
3. Round 2
   - K2
   - Permuted Choice 2
   - Left circular shift
4. Round 16
   - K16
   - Permuted Choice 2
   - Left circular shift
5. 32-bit Swap
6. Inverse Initial Permutation
7. 64-bit ciphertext

64-bit plaintext
64-bit key
SINGLE ROUND OF DES
DES S-BOXES

• have eight S-boxes which map 6 to 4 bits
• each S-box is actually 4 little 4 bit boxes
  – outer bits 1 & 6 (row bits) select one row of 4
  – inner bits 2-5 (col bits) are substituted
  – result is 8 lots of 4 bits, or 32 bits
• For example, in S1, for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.
DES S-BOXES

R (32 bits) → E → 48 bits → + → K (48 bits) → S1 → S2 → S3 → S4 → S5 → S6 → S7 → S8 → P → 32 bits
<table>
<thead>
<tr>
<th>S_1</th>
<th>14</th>
<th>4</th>
<th>13</th>
<th>1</th>
<th>2</th>
<th>15</th>
<th>11</th>
<th>8</th>
<th>3</th>
<th>10</th>
<th>6</th>
<th>12</th>
<th>5</th>
<th>9</th>
<th>0</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_2</th>
<th>15</th>
<th>1</th>
<th>8</th>
<th>14</th>
<th>6</th>
<th>11</th>
<th>3</th>
<th>4</th>
<th>9</th>
<th>7</th>
<th>2</th>
<th>13</th>
<th>12</th>
<th>0</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>13</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>2</td>
<td>8</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>7</td>
<td>11</td>
<td>10</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_3</th>
<th>10</th>
<th>0</th>
<th>9</th>
<th>14</th>
<th>6</th>
<th>3</th>
<th>15</th>
<th>5</th>
<th>1</th>
<th>13</th>
<th>12</th>
<th>7</th>
<th>11</th>
<th>4</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>7</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>15</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>14</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>15</td>
<td>14</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_4</th>
<th>7</th>
<th>13</th>
<th>14</th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>9</th>
<th>10</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>5</th>
<th>11</th>
<th>12</th>
<th>4</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>12</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_5</th>
<th>2</th>
<th>12</th>
<th>4</th>
<th>1</th>
<th>7</th>
<th>10</th>
<th>11</th>
<th>6</th>
<th>8</th>
<th>5</th>
<th>3</th>
<th>15</th>
<th>13</th>
<th>0</th>
<th>14</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>11</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>8</td>
<td>15</td>
<td>9</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>12</td>
<td>7</td>
<td>1</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_6</th>
<th>12</th>
<th>1</th>
<th>10</th>
<th>15</th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>8</th>
<th>0</th>
<th>13</th>
<th>3</th>
<th>4</th>
<th>14</th>
<th>7</th>
<th>5</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>13</td>
<td>14</td>
<td>0</td>
<td>11</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>11</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_7</th>
<th>4</th>
<th>11</th>
<th>2</th>
<th>14</th>
<th>15</th>
<th>0</th>
<th>8</th>
<th>13</th>
<th>3</th>
<th>12</th>
<th>9</th>
<th>7</th>
<th>5</th>
<th>10</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_8</th>
<th>13</th>
<th>2</th>
<th>8</th>
<th>4</th>
<th>6</th>
<th>15</th>
<th>11</th>
<th>1</th>
<th>10</th>
<th>9</th>
<th>3</th>
<th>14</th>
<th>5</th>
<th>0</th>
<th>12</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>13</td>
<td>8</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>15</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>14</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>13</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
DES DECRYPTION

• Same as encryption except the application of the key is reversed.
AVALANCHE EFFECT

• key desirable property of encryption algorithm.
• where a change of one input or key bit results in changing approx half output bits
• making attempts to “home-in” by guessing keys impossible
• DES exhibits strong avalanche
STRENGTH OF DES – KEY SIZE

• 56-bit keys have \(2^{56} = 7.2 \times 10^{16}\) values
• brute force search looks hard
• recent advances have shown is possible
  – in 1997 on Internet in a few months
  – in 1998 on dedicated h/w (EFF) in a few days
  – in 1999 above combined in 22hrs!
• still must be able to recognize plaintext
• must now consider alternatives to DES
Strength of DES – Analytic Attack

• now have several analytic attacks on DES
• these utilise some deep structure of the cipher
  – by gathering information about encryptions
  – can eventually recover some/all of the sub-key bits
  – if necessary then exhaustively search for the rest
• generally these are statistical attacks
• include
  – differential cryptanalysis
  – linear cryptanalysis
  – related key attacks
Strength of DES – Timing attack

• attacks actual implementation of cipher
• use knowledge of consequences of implementation to derive information about some/all subkey bits
• specifically use fact that calculations can take varying times depending on the value of the inputs to it
MULTIPLE ENCRYPTION & DES

• potential vulnerability of DES to a brute-force attack, made a considerable interest in finding an alternative.
  – One approach is to design a completely new algorithm like AES
  – Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys.

• We examine the widely accepted triple DES (3DES) approach
DOUBLE DES

• The simplest form of multiple encryption has two encryption stages and two keys - Double-DES.
  – could use 2 DES encrypts on each block
  – \( C = E_{K2}(E_{K1}(P)) \)
• issue of reduction to single stage (might have a single key that is equivalent to using 2 keys as above)
• “meet-in-the-middle” attack
  – works whenever use a cipher twice
  – known plaintext attack
  – since \( X = E_{K1}(P) = D_{K2}(C) \)
  – takes \( O(2^{56}) \) search steps
TRIPLE-DES WITH TWO-KEYS

• Triple-DES with two keys is a popular alternative to single-DES
• Encryption & decryption stages are equivalent, but the chosen structure allows for compatibility with single-DES implementations
• hence must use 3 encryptions
  – would seem to need 3 distinct keys
• but can use 2 keys with E-D-E sequence
  – $C = E_{K_1} \left( D_{K_2} \left( E_{K_1} \left( P \right) \right) \right)$
  – encrypt & decrypt equivalent in security
  – if $K_1=K_2$ then can work as single DES
TRIPLE-DES WITH THREE-KEYS

• Although are no practical attacks on two-key Triple-DES have some indications

• We can use Triple-DES with Three-Keys to avoid even these
  \[ C = E_{K3}(D_{K2}(E_{K1}(P))) \]

• It has been adopted by some Internet applications, eg PGP, S/MIME
MODES OF OPERATION

• block ciphers encrypt fixed size blocks
  – eg. DES encrypts 64-bit blocks with 56-bit key
• need some way to en/decrypt arbitrary amounts of data in practise
• **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
• subsequently 5 defined for AES & DES
MODES OF OPERATION

- Electronic Codebook Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- Output FeedBack (OFB)
- Counter (CTR)
ELECTRONIC CODEBOOK BOOK (ECB)

Time = 1
\[ P_1 \]
K → Encrypt
\[ C_1 \]

Time = 2
\[ P_2 \]
K → Encrypt
\[ C_2 \]

Time = N
\[ P_N \]
K → Encrypt
\[ C_N \]

(a) Encryption

\[ C_1 \]
K → Decrypt
\[ P_1 \]

\[ C_2 \]
K → Decrypt
\[ P_2 \]

\[ C_N \]
K → Decrypt
\[ P_N \]

(b) Decryption
ELECTRONIC CODEBOOK BOOK (ECB)

- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks
  \[ C_i = DES_{K_1}(P_i) \]
- uses: secure transmission of single values
- *eg. a session key encrypted using a master key*
ADVANTAGES AND LIMITATIONS OF ECB

• The ECB method is ideal for short amount of data.
• If we want to transmit a DES key securely, ECB is the appropriate to mode to use.
• For lengthy messages, the ECB may not be secure.
• The most significant characteristic of ECB is that the same b-bit block of plain text, if it appears more than once in the message, always produces the same cipher text.
CIPHER BLOCK CHAINING (CBC)
CIPHER BLOCK CHAINING (CBC)

- message is broken into blocks
- linked together in encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

\[
C_i = \text{DES}_{K_1}(P_i \oplus C_{i-1}) \\
C_{-1} = \text{IV}
\]

- uses: bulk data encryption, authentication
- eg email, FTP, web etc
ADVANTAGES AND LIMITATIONS OF CBC

• a ciphertext block depends on all blocks before it
• any change to a block affects all following ciphertext blocks
• need Initialization Vector (IV)
  – which must be known to sender & receiver
  – if sent in clear, attacker can change bits of first block, and change IV to compensate
  – hence IV must either be a fixed value
  – or must be sent encrypted in ECB mode before rest of message
CIPHER FEEDBACK (CFB)
CIPHER FEEDBACK (CFB)

- message is treated as a stream of bits to avoid padding
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1, 8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)
  \[ C_i = P_i \ XOR \ DES_{K_1} (C_{i-1}) \]
  \[ C_{-1} = IV \]
- uses: stream data encryption, authentication
- eg. terminal session, sensor value etc
ADVANTAGES AND LIMITATIONS OF CFB

• appropriate when data arrives in bits/bytes
• most common stream mode
• limitation is need to stall while do block encryption after every n-bits
• note that the block cipher is used in encryption mode at both ends
• errors propogate for several blocks after the error
Output Feedback (OFB)
Output Feedback (OFB)

- message is treated as a stream of bits
- output of cipher is added to message
- output is then feedback (hence name)
- feedback is independent of message
- can be computed in advance
  \[ C_i = P_i \oplus O_i \]
  \[ O_i = DES_{K_1}(O_{i-1}) \]
  \[ O_{-1} = IV \]
- uses: stream encryption on noisy channels
- eg satellite TV transmissions
ADVANTAGES AND LIMITATIONS OF OFB

- bit errors do not propagate
- more vulnerable to message stream modification
- a variation of a Vernam cipher
  – hence must never reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- subsequent research has shown that only full block feedback (ie CFB-64 or CFB-128) should ever be used
COUNTER (CTR)

• a “new” mode, though proposed early on
• similar to OFB but encrypts counter value rather than any feedback value
• must have a different key & counter value for every plaintext block (never reused)

\[
C_i = P_i \ XOR \ O_i \\
O_i = DES_{K_1}(i)
\]

• uses: high-speed network encryptions
• applications in ATM (asynchronous transfer mode) network security and IPSec (IP security).
COUNTER (CTR)

(a) Encryption

Counter
K → Encrypt
P₁ → C₁

Counter + 1
K → Encrypt
P₂ → C₂

Counter + N - 1
K → Encrypt
Pₙ → Cₙ

(b) Decryption

Counter 1
K → Encrypt
C₁ → P₁

Counter 2
K → Encrypt
C₂ → P₂

Counter N
K → Encrypt
Cₙ → Pₙ
ADVANTAGES AND LIMITATIONS OF CTR

- **efficiency**
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- **random access to encrypted data blocks**
- **provable security** (good as other modes)
- **but must ensure never reuse key/counter values, otherwise could break**
ADVANCE ENCRYPTION STANDARD

• Origins
  - replacement for DES was needed
  - can use Triple-DES – but slow, has small blocks
  - US NIST issued call for ciphers in 1997
  - 15 candidates accepted in Jun 98
  - 5 were shortlisted in Aug-99
  - Rijndael was selected as the AES in Oct-2000 (both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen)
  - issued as FIPS PUB 197 standard in Nov-2001
ADVANCE ENCRYPTION STANDARD

• published by NIST in 2001.
• significant increase in the block size – from 64-bits up to 128-bits; and keys from 128 to 256-bits.
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
• provide full specification & design details
• both C & Java implementations
• NIST have released all submissions & unclassified analyses
AES Evaluation Criteria

- **initial criteria:**
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics

- **final criteria**
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)
AES Shortlist

• after testing and evaluation, shortlist in Aug-99:
  – MARS (IBM) - complex, fast, high security margin
  – RC6 (USA) - v. simple, v. fast, low security margin
  – Rijndael (Belgium) - clean, fast, good security margin
  – Serpent (Euro) - slow, clean, v. high security margin
  – Twofish (USA) - complex, v. fast, high security margin
• then subject to further analysis & comment
• saw contrast between algorithms with
  – few complex rounds verses many simple rounds
  – which refined existing ciphers verses new proposals
The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity
Rijndael

• The input is a single 128-bit block, (a square matrix of bytes)
• This block is copied into the State array, (modified at each stage
• After the final stage, State is copied to an output.
• The key is expanded into 44/52/60 lots of 32-bit words with 4 used in each round.
Rijndael

• 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)
  – view as alternating XOR key & scramble data bytes (final 10\textsuperscript{th}/12\textsuperscript{th}/14\textsuperscript{th} step of byte subs + mix cols + add round key)

• All of the steps are easily reversed, and can be efficiently implemented using XOR’s & table lookups.
Rijndael

(a) Encryption

(b) Decryption
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 indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte \{95\} is replaced by byte in row 9 column 5
  - which has value \{2A\}
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks
Byte Substitution
Shift Rows

- a circular byte shift in each
  - 1\textsuperscript{st} row is unchanged
  - 2\textsuperscript{nd} row does 1 byte circular shift to left
  - 3\textsuperscript{rd} row does 2 byte circular shift to left
  - 4\textsuperscript{th} row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns
Shift Rows
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 $m(x) = x^8 + x^4 + x^3 + x + 1$
Mix Columns
Mix Columns

- can express each col as 4 equations
  - to derive each new byte in col
- decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder
- have an alternate characterisation
  - each column a 4-term polynomial
  - with coefficients in GF(2^8)
  - and polynomials multiplied modulo (x^4+1)
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 identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security
Add Round Key

\[ \begin{array}{cccc}
  s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
  s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
  s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
  s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\
\end{array} \oplus \begin{array}{cccc}
  w_i & w_{i+1} & w_{i+2} & w_{i+3} \\
\end{array} = \begin{array}{cccc}
  s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\
  s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\
  s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\
  s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \\
\end{array} \]
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
  - 1\textsuperscript{st} word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4\textsuperscript{th} back
AES Key Expansion
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU’s
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description
AES Decryption
AES Decryption

• AES decryption is not identical to encryption since steps done in reverse
• 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 shift
  - add round key works on byte XOR’s
  - mix columns requires matrix multiply in GF($2^8$) which works on byte values, can be simplified to use table lookups & byte XOR’s
Implementation Aspects

• can efficiently implement on 32-bit CPU
  – redefine steps to use 32-bit words
  – can precompute 4 tables of 256-words
  – then each column in each round can be computed using 4 table lookups + 4 XORs
  – at a cost of 4Kb to store tables

• designers believe this very efficient implementation was a key factor in its selection as the AES cipher
Steps in Rijndael at a High Level

• Do the following one-time initialization processes:
  – Expand the 16-byte key to get the actual key block to be used.
  – Do one time initialization of the 16-byte plain text block (called as state).
  – XOR the state with the key block.

• For each round, do the following:
  – Apply S-box to each of the plain text bytes.
  – Rotate row k of the plain text block (state) by k bytes.
  – XOR state with the key block.
AES Key Expansion

Key expansion(byte key[16], word w[44])
{
    Word temp
    for(i=0;i<4;i++)
    {
        W[i]=(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
        For(i=4;i<44;i++)
        {temp=w[i-1];
            If(I mod 4=0)
                Temp=SubWord(RotWord(temp))XOR Rcon[i/4];
        W[i]= w[i-4] XOR temp
(i) Function *Rotate* performs a circular left shift on the contents of the word by one byte. Thus, if an input word contains four bytes numbered \([B_1, B_2, B_3, B_4]\); then the output word would contain \([B_2, B_3, B_4, B_1]\).

(ii) Function *Substitute* performs a byte substitution on each byte of the input word. For this purpose, it uses an S-box, shown in Fig. 3.79.

![AES S-box](image)

(iii) In the function *Constant*, the output of the above steps is XORed with a constant. This constant is a word, consisting of 4 bytes. The value of the constant depends on the round number. The last three bytes of a constant word always contain 0. Thus, XORing any input word with such a constant is as good as XORing only with the first byte of the input word. These constant values per round are listed in Fig. 3.80.

![Fig. 3.79 AES S-box](image)

![Fig. 3.80 Values of constants per round, to be used in the Constant function](image)

Let us understand how the whole thing works, with an example.

1. Suppose that our original unexpanded 4-word (i.e. 16-byte) key is as shown in Fig. 3.81.

![Fig. 3.81 Key expansion module - Step 1: Original 16-word key](image)
2. In the first four rounds, the original 4-word input key would be expanded into the first 4-word output key, so our algorithm step as follows:

\[
\begin{align*}
&\text{W1} = (W_0, W_1, W_2, W_3), \\
&\text{W2} = (W_4, W_5, W_6, W_7), \\
&\text{W3} = (W_8, W_9, W_{10}, W_{11}), \\
&\text{W4} = (W_{12}, W_{13}, W_{14}, W_{15}).
\end{align*}
\]

This is constructed from the input 4-word (4s. W1-W4) keys as follows:

(1) Firstly, the first four bytes, i.e., one word of the input key, namely 0x00 0x00 0x00 0x00 is copied into the first word of the initial key, W 11.

(2) Next, the next four bytes of the input key, i.e., 0x00 0x00 0x00 0x00 would be copied into the second word of the initial key, i.e., W 11.

(3) Following this, W 11 and W 11 would be preprocessed with the remaining parameters of the initial key byte, as shown in the figure.

<table>
<thead>
<tr>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>

(4) Fig. 3.10: Key expansion example - Step 2: 4096 of first key expansion key.

3. Now let us understand how the second word of the output key block, i.e., W 11 is produced, for this purpose, the following algorithm would be executed:

\[
\text{W11} \oplus \overline{\text{W11}} = \text{W11} \oplus \text{W11} = 0x00 0x00 0x00 0x00
\]

Based on this, we have the following:

\[
\begin{align*}
&\text{W1} = (W_0, W_1, W_2, W_3), \\
&\text{W2} = (W_4, W_5, W_6, W_7), \\
&\text{W3} = (W_8, W_9, W_{10}, W_{11}), \\
&\text{W4} = (W_{12}, W_{13}, W_{14}, W_{15}).
\end{align*}
\]

Since 0x00 and 0x00, therefore, we will have the following step:

\[
\text{e.g., Substitute (Basic key)} \rightarrow \text{NOE Cation (W1)}
\]

We know that the first step will produce RHSR (40 00 00 00) which replaces 0x00 0x00 0x00 0x00.

Now, we need to use the Substitution (Basic key) for this purpose, we need to make changes to the first half of the algorithm. For example, our first byte is 0x00. So, we replace it with 0x00, and so on.

Thus, the end of the Substitution (Basic key) stage, it is transformed into the output of 0x00 0x00 0x00 0x00.

We now want to NOR the last two stages with the upper half stage, i.e., 0x00 0x00 0x00 0x00, which is still equal to the initial 4-key block. As we know, we
The text seems to be discussing cryptographic processes, possibly related to hash functions or encryption algorithms. The page contains diagrams and equations, which are typical in the field of computer science and cryptography. The content appears to be technical and detailed, focusing on specific algorithms or protocols.
As this stage, the calculation is complex and we won't study it further.

### 3.8.4. Processes in Each Round

The following steps are executed 10 times, one per round.

1. **Apply S-box** to Each of the Three Tent Bytes

   This step is very straightforward. The contents of the state array and placed up into the S-box. By Substitution is done to replace the contents of the state array. The only S-box used is the linear S-box which has neither S-box.

2. **Byte-Row Key of the Plain Text Mode, k, is added by R Bytes.**

   Here, each of the four rows of the state array are sequentially the R bytes. Here, it is a random function (i.e., it is random in all four 8-bit strings by bytes. Each 8-bit random S-box and once 8-bit random 5-byte NOR 5-byte. This helps in flattening of data. Thus, the operation 16 total of the state array has been modified (16, 13, 10, 2, 3, 4, 8, 14, 15, 11, 1, 5, 9, 16, 15). At this stage, the state operation would change the values as follows:

<table>
<thead>
<tr>
<th>Original array</th>
<th>Modified array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 13</td>
<td>1 2 3 13</td>
</tr>
<tr>
<td>2 4 6 14</td>
<td>2 4 6 14</td>
</tr>
<tr>
<td>5 7 11 10</td>
<td>5 7 11 10</td>
</tr>
<tr>
<td>8 9 12 16</td>
<td>8 9 12 16</td>
</tr>
</tbody>
</table>

3. **Perform a Mix Column Operation.**

   Now, each column is mixed independently of the other. Hence, multiplication is used. The output of this step is the state: multiplication of the second and a constant matrix.

   This is perhaps the most complex step to understand and also to implement. No calculation, you will make an effort. This is found in many articles by Adi Shamir.

   These are the aspects of this step. The first explains which parts of the states are multiplied against which parts of the states. The second expands how the multiplication is implemented over which is called as the Galois Field.

4. **Matrix Multiplication.**

   We know that the state is arranged into a 4 x 4 matrix.

   The multiplication is performed over each column of a state (i.e., a 4 x 4 matrix). Each value in the elements is essentially multiplied against every value of the matrix (i.e., 36 total multiplications are performed). The results of these multiplications are XORed together to produce only 4 resulting bytes for the new state. We together have 4 bytes input. 36 multiplications, 36 XOR operations and 4 bytes output. The multiplication is performed over each column. row to a total of 16, value of a state column.

   The example considers that our matrix is shown in Fig. 3.46.
Next, let us consider the 16-byte array in Fig. 3.14.

\[
\begin{pmatrix}
16 & 8 & 36 & 13 \\
17 & 101 & 45 & 22 \\
5 & 9 & 113 & 7 \\
6 & 10 & 114 & 8
\end{pmatrix}
\]

**Fig. 3.14: 16-byte data array**

The first result byte is calculated by taking the four values of the data array with the first value of the first row of the matrix. The result of such a multiplication is then XOR'd to produce the first byte:

\[ A = (01100101) \times (1011) = (10010000) \text{ (XOR)} (00101000) \text{ (XOR)} (10011000) = (10101000) \]

Next, the second result byte is calculated by multiplying the same four values of the same column against the second value of the second row of the matrix. The result of such a multiplication is then XOR'd to produce one byte:

\[ B = (01100101) \times (11001101) = (10010000) \text{ (XOR)} (00101000) \text{ (XOR)} (10011000) = (10101000) \]

The third result byte is calculated by multiplying the same four values of the same column against the fourth value of the first row of the matrix. The result of such a multiplication is then XOR'd to produce one byte:

\[ C = (01100101) \times (11010100) = (10010000) \text{ (XOR)} (00101000) \text{ (XOR)} (10011000) = (10101000) \]

The fourth result byte is calculated by multiplying the same four values of the same column against the second value of the second row of the matrix. The result of such a multiplication is then XOR'd to produce one byte:

\[ D = (01100101) \times (11001101) = (10010000) \text{ (XOR)} (00101000) \text{ (XOR)} (10011000) = (10101000) \]

This procedure is repeated with the next column of the matrix, and there are no more columns.

To summarize:

The first column will contain data bytes 1-4 and will be multiplied against the matrix in the following manner:

\[
\begin{align*}
16 & = (01100) \text{ XOR (00001)} \text{ XOR (00001)} \text{ XOR (00001)} \\
17 & = (1011) \text{ XOR (00001)} \text{ XOR (00001)} \text{ XOR (00001)} \\
5 & = (01100) \text{ XOR (00001)} \text{ XOR (00001)} \text{ XOR (00001)} \\
6 & = (1011) \text{ XOR (00001)} \text{ XOR (00001)} \text{ XOR (00001)}
\end{align*}
\]

\[
\begin{align*}
\text{and the specified first byte of the first} \\
\text{column will be multiplied against the first column of the matrix.}
\end{align*}
\]

The second column will be multiplied against the second column of the matrix with the following manner:

\[
\begin{align*}
36 & = (01100010) \text{ XOR (00001001)} \text{ XOR (00001001)} \text{ XOR (00001001)} \\
45 & = (11100001) \text{ XOR (00001001)} \text{ XOR (00001001)} \text{ XOR (00001001)} \\
113 & = (01110001) \text{ XOR (00001001)} \text{ XOR (00001001)} \text{ XOR (00001001)} \\
114 & = (01110001) \text{ XOR (00001001)} \text{ XOR (00001001)} \text{ XOR (00001001)}
\end{align*}
\]

\[
\begin{align*}
\text{and the specified second byte of the second} \\
\text{column will be multiplied against the second} \\
\text{column of the matrix.}
\end{align*}
\]

And so on until all columns of the matrix are processed.
BLOW FISH

• Blowfish is a symmetric block cipher that can be used as a replacement forDES.
• It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.
• Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms.
• Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm.
• Blowfish is unpatented and license-free, and is available free for all uses.
BLOW FISH

• The original Blowfish paper was presented at the *First Fast Software Encryption workshop* in Cambridge, UK
  – (proceedings published by Springer-Verlag, *Lecture Notes in Computer Science #809, 1994*)
BLOW FISH

- Blowfish is a block cipher that encrypts data in 8-byte blocks.
- The algorithm consists of two parts:
  - a key-expansion part
  - a data-encryption part.
BLOW FISH

• **The expansion of the key:**
  – break the original key into a set of subkeys.
  – Specifically, a key of no more than 448 bits is separated into 4168 bytes.
  – There is a P-array and four 32-bit S-boxes. The P-array contains 18 32-bit subkeys, while each S-box contains 256 entries.

• **The encryption of the data:** 64-bit input is denoted with an x, while the P-array is denoted with a Pi (where i is the iteration).
KEY EXPANSION

• Blowfish has a 64-bit block size and a key length of anywhere from 32 bits to 448 bits (32-448 bits in steps of 8 bits; default 128 bits).

• It is a 16-round Feistel cipher and uses large key-dependent S-boxes.
KEY EXPANSION

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string.
2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.
BLOW FISH IN ACTION

- Each line represents 32 bits.
- The algorithm keeps two subkey arrays:
  - the 18-entry P-array
  - four 256-entry S-boxes.
- The S-boxes accept 8-bit input and produce 32-bit output.
- One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries.
BLOW FISH IN ACTION

• Blowfish has 16 rounds. The input is a 64-bit data element, $x$. Divide $x$ into two 32-bit halves: $x_L$, $x_R$. Then, for $i = 1$ to 16:
  — $x_L = x_L \oplus P_i$
  — $x_R = F(x_L) \oplus x_R$
  Swap $x_L$ and $x_R$

• After the sixteenth round, swap $x_L$ and $x_R$ again to undo the last swap. Then, $x_R = x_R \oplus P_{17}$ and $x_L = x_L \oplus P_{18}$. 

BLOW FISH F-FUNCTION

- The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo $2^{32}$ and XORed to produce the final 32-bit output.

- Divide $xL$ into four eight-bit quarters: $a$, $b$, $c$, and $d$. Then, $F(xL) = ((S1,a + S2,b \mod 2^{32}) \text{ XOR } S3,c) + S4,d \mod 2^{32}$
DECRIPTION

- Decryption is exactly the same as encryption, except that P1, P2, ..., P18 are used in the reverse order.
RC5

- Designed by Ronald Rivest for RSA Data Security
  - Secret-key block cipher
  - RC5 is word-oriented
    - Two-word input and two-word output
  - Representation
    - word size, \( w \)
    - number of rounds, \( r \)
    - number of bytes in key, \( b \)
    - RC5 algorithm notation: RC5-\(w/r/b\)
RC5

- RC5 algorithm example: RC5-32/16/7
  - similar to DES
  - Two 32-bit word inputs and outputs
  - 16 rounds
  - 7-byte (56-bit) secret key

- The three components of RC5
  - Key expansion algorithm
  - Encryption algorithm
  - Decryption algorithm
RC5

• Key expansion, magic constants
  – $P_w = \text{Odd} \left( (e - 2)2^w \right)$
  – $Q_w = \text{Odd} \left( (\phi - 1)2^w \right)$

• Key expansion algorithm, step one
  – convert secret key bytes to words
    for $i = b - 1$ downto 0 do
      $L[i/u] = (L[i/u] << < 8) + K[i]$;
RC5

• Key expansion algorithm, step two
  – create an expanded key table, $S[0...t-1]$
    • has $t$ entries, $t = 2(r + 1)$ $w$-bit words
  – Initialize array $S$
    $S[0] = P_w$
    for $i = 1$ to $t - 1$ do
      $S[i] = S[i - 1] + Q_w$
RC5

• Key expansion algorithm, step three
  – Mix the secret key into table, $S$
    
    $i = j = 0; \quad A = B = 0;$
    do $3 \times \text{max}(t, c)$ times:

    $A = S[i] = (S[i] + A + B) \ll 3;$
    $B = L[j] = (L[j] + A + B) \ll (A + B);$  
    $i = (i + 1) \mod(t);$  
    $j = (j + 1) \mod(c);$
RC5

• Encryption

\[
\begin{align*}
A &= A + S[0]; \\
B &= B + S[1]; \\
\text{for } i = 1 \text{ to } r \text{ do} \\
A &= ((A \text{xor } B) <<< B) + S[2*i]; \\
B &= ((B \text{xor } A) <<< A) + S[2*i + 1];
\end{align*}
\]
RC5

• Decryption
  
  for $i = r$ downto 1 do
    
    $B = ((B - S[2*i + 1]) >>> A) \text{ xor } A;$
    
    $A = ((A - S[2*i]) >>> B) \text{ xor } B;$
    
    $B = B - S[1];$
    
    $A = A - S[0];$