CSC 474 Network Security

Topic 3.1 Secret Key Cryptography – Algorithms
Outline

- Introductory Remarks
- Feistel Cipher
- DES
- AES
Introduction
Generic Block Encryption

- Converts one input plaintext block of fixed size $k$ bits to an output ciphertext block also of $k$ bits
- Benefits of large $k$? of short $k$?
Key Sizes

- Keys should be selected from a large potential set, to prevent brute force attacks
- Secret key sizes
  - 40 bits were considered adequate in 70’s
  - 56 bits used by DES were adequate in the 80’s
  - 128 bits are adequate for now
- If computers increase in power by 40% per year, need roughly 5 more key bits per decade to stay “sufficiently” hard to break
## Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \oplus Y$</td>
<td>Bit-wise exclusive-or of $X$ and $Y$</td>
</tr>
<tr>
<td>$X \mid Y$</td>
<td>Concatenation of $X$ and $Y$</td>
</tr>
<tr>
<td>$K{m}$</td>
<td>Message $m$ encrypted with secret key $K$</td>
</tr>
</tbody>
</table>
Two Principles for Cipher Design

- **Confusion:**
  - Make the relationship between the \(<\text{plaintext, key}>\) input and the \(<\text{ciphertext}>\) output as complex (non-linear) as possible

- **Diffusion:**
  - Spread the influence of each input bit across many output bits
Exploiting the Principles

- Idea: use **multiple, alternating** permutations and substitutions, e.g.,
  - S → P → S → P → S → ...
  - P → S → P → S → P → ...
- Do they have to alternate? e.g.….  
  - S → S → S → P → P → P → S → S → …??
- Confusion is mainly accomplished by **substitutions**
- Diffusion is mainly accomplished by **permutations**
- Example ciphers: **DES, AES**
Secret Key… (Cont’d)

• Basic technique used in secret key ciphers: multiple applications of alternating substitutions and permutations

Well-known examples: DES, AES
Basic Form of Modern Block Ciphers

Plaintext block

Preprocessing

Rounds of Encryption
\[ i=1,2,\ldots,n \]

Postprocessing

Ciphertext block

Key

Sub-Key Generation

Sub-Key #1
Sub-Key #2
Sub-Key #3

... Sub-Key #n
Feistel Ciphers
Overview

• Feistel Cipher has been a very influential “template” for designing a block cipher
• Major benefit: can do encryption and decryption with the same hardware
• Examples: DES, RC5
One “Round” of Feistel Encryption

1. Break input block $i$ into left and right halves $L_i$ and $R_i$
2. Copy $R_i$ to create output half block $L_{i+1}$
3. Half block $R_i$ and key $K_i$ are “scrambled” by function $f$
4. XOR result with input half-block $L_i$ to create output half-block $R_{i+1}$
One “Round” of Feistel Decryption

• Just reverse the arrows!

\begin{align*}
L_{i+1} & \oplus f(L_i) \rightarrow R_i \\
K_i & \rightarrow f(L_i) \\
R_{i+1} & \leftarrow L_i \\
L_i & \leftarrow R_{i+1}
\end{align*}
Complete Feistel Cipher: Encryption

Plaintext (2w bits)

\[
\begin{align*}
L_0 & \quad R_0 \\
\downarrow & \quad \downarrow \\
L_1 & \quad R_1 \\
\downarrow & \quad \downarrow \\
\vdots & \quad \vdots \\
L_n & \quad R_n \\
\downarrow & \quad \downarrow \\
L_{n+1} & \quad R_{n+1}
\end{align*}
\]

Ciphertext (2w bits)

Round 1

Round \( i \)

Round \( n \)

\( f \)

\( K_1 \)

\( K_2 \)

\( K_n \)

\( L_0 \)

\( R_0 \)

\( L_1 \)

\( R_1 \)

\( \vdots \)

\( L_n \)

\( R_n \)

\( L_{n+1} \)

\( R_{n+1} \)

note this final swap!
Feistel Cipher: Decryption

Ciphertext (2w bits)

Round 1

Round $i$

Round $n$

(note this final swap!)

Plaintext (2w bits)
Parameters of a Feistel Cipher

• Block size
• Key size
• Number of rounds
• Subkey generation algorithm
• “Scrambling” function $f$
Comments

• Decryption is same as encryption, only reversing the order in which round keys are applied
  – Reversability of Feistel cipher derives from reversibility of XOR

• Function $f$ can be anything
  –Hopefully something easy to compute
  –There is no need to invert $f$
DES (Data Encryption Standard)
DES (Data Encryption Standard)

• Standardized in 1976 by NBS
  – proposed by IBM,
  – Feistel cipher

• Criteria (official)
  – provide high level of security
  – security must reside in key, not algorithm
  – not patented
  – must be exportable
  – efficient to implement in hardware
DES… (Cont’d)

• Criteria (unofficial)
  – must be slow to execute in software
  – must be breakable by NSA :-)

CSC 474
Dr. Peng Ning
DES Basics

- **Blocks**: 64 bit plaintext input, 64 bit ciphertext output
- **Rounds**: 16
- **Key**: 64 bits
  - every 8\textsuperscript{th} bit is a parity bit, so really \textbf{56 bits} long
DES Top Level View

64-bit Input

Initial Permutation

Round 1

48-bit $K_1$

Round 2

48-bit $K_2$

Round 16

48-bit $K_{16}$

Swap Halves

Final Permutation

64-bit Output

56-bit Key

Generate round keys
Initial and Final Permutations

- **Initial** permutation given below
  - input bit #58 $\rightarrow$ output bit #1, input bit #50 $\rightarrow$ output bit #2, ...

<table>
<thead>
<tr>
<th></th>
<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>1</td>
<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>2</td>
<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>3</td>
<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>4</td>
<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>5</td>
<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>6</td>
<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>7</td>
<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>

Diagram:
- 64-bit Input $\rightarrow$ Initial Permutation $\rightarrow$ Round 1 $\rightarrow$ Round 2 $\rightarrow$ ... $\rightarrow$ Round 16 $\rightarrow$ Swap Halves $\rightarrow$ Final Permutation $\rightarrow$ 64-bit Output
- Generate round keys $\rightarrow$ 48-bit K $\rightarrow$ 48-bit K $\rightarrow$ 48-bit K...

56-bit Key
Initial... (Cont’d)

• Final permutation is just inverse of initial permutation, i.e.,
  – input bit #1 → output bit #58
  – input bit #2 → output bit #50
  – ...

Diagram:

- 64-bit Input
- Initial Permutation
- Round 1
- Round 2
- ...
- Round 16
- Swap Halves
- Final Permutation
- 56-bit Key
- Generate round keys
- 48-bit K
- 48-bit K
- 48-bit K
- 64-bit Output

Diagram:

Flowchart:

- 64-bit Input
- Initial Permutation
- Round 1
- Round 2
- ...
- Round 16
- Swap Halves
- Final Permutation
- 56-bit Key
- Generate round keys
- 48-bit K
- 48-bit K
- 48-bit K
- 64-bit Output
Initial… (Cont’d)

• Note #1: Initial Permutation is fully specified (independent of key)
  – therefore, does not improve security!
  – why needed?

• Note #2: Final Permutation is needed to make this a Feistel cipher
  – i.e., can use same hardware for both encryption and decryption
Key Generation: First Permutation

- **First step:** throw out 8 parity bits, then permute resulting 56 bits

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

*Parity bits left out: 8,16,24,...*
KeyGen: Processing Per Round

\[ C_{i-1} \text{ 28 bits} \rightarrow \text{Circular Left Shift} \rightarrow \text{Permutation with Discard} \rightarrow \text{Circular Left Shift} \rightarrow D_{i-1} \text{ 28 bits} \]

- Rounds \( i = 1, 2, 9, 16 \): left circular shift 1 bit
- Other rounds: left circular shift 2 bits

48 bit \( K_i \)
KeyGen: Permutation with Discard

- 28 bits $\rightarrow$ 24 bits, each half of key

**Left** half of $K_i = $ permutation of $C_i$

<table>
<thead>
<tr>
<th>14</th>
<th>17</th>
<th>11</th>
<th>24</th>
<th>1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>28</td>
<td>15</td>
<td>6</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>19</td>
<td>12</td>
<td>4</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>27</td>
<td>20</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>

**Bits left out:** 9, 18, 22, 25

**Right** half of $K_i = $ permutation of $D_i$

<table>
<thead>
<tr>
<th>41</th>
<th>52</th>
<th>31</th>
<th>37</th>
<th>47</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>40</td>
<td>51</td>
<td>45</td>
<td>33</td>
<td>48</td>
</tr>
<tr>
<td>44</td>
<td>49</td>
<td>39</td>
<td>56</td>
<td>34</td>
<td>53</td>
</tr>
<tr>
<td>46</td>
<td>42</td>
<td>50</td>
<td>36</td>
<td>29</td>
<td>32</td>
</tr>
</tbody>
</table>

**Bits left out:** 35, 38, 43, 54
One DES (Feistel) Round

Input block $i$

$L_i$  $R_i$

$f$

$K_i$

$L_{i+1}$  $R_{i+1}$

Output block $i+1$

---

**Diagram Details:**

- **Input block $i$:**
  - $L_i$
  - $R_i$

- **Function $f$:**
  - Encodes the intermediate block.

- **Output block $i+1$:**
  - $L_{i+1}$
  - $R_{i+1}$

---

**Flowchart:**

- **Initial Permutation**
- **Final Permutation**
- **Swap Halves**
- **Generate round keys**
- **64-bit Input**
- **56-bit Key**

---

**Key Points:**

- One DES (Feistel) Round consists of multiple rounds.
- Each round involves input block transformation, function application, and output block generation.
- DES uses Feistel network structure for encryption and decryption.

---

**Notes:**

- The Feistel structure is symmetric, allowing the same process to be used for both encryption and decryption.
- Each round key is derived from the master key through a series of transforms.
- The initial block is permuted before processing.
- The final permutation is applied after the last round.

---

**References:**

- For detailed DES algorithm, refer to the original specification or academic sources.
- Understanding the Feistel network is crucial for grasping modern cryptographic techniques.

---

**Further Reading:**

- Bruce Schneier's *Applied Cryptography* provides a comprehensive overview.
- Wikipedia articles on DES and Feistel networks are also valuable resources.
DES Round: \( f \) (Mangler) Function

### Input block \( i \)
- \( L_i \)
- \( R_i \)

### Output block \( i+1 \)
- \( L_{i+1} \)
- \( R_{i+1} \)

Function \( f = ”\text{Mangler”} \)

- 32-bit half block
- 48 bits

Expansion

S-Box (substitution)

Permutation

32-bit half block

\( K_i \)
**f: Expansion Function**

- 32 bits $\rightarrow$ 48 bits

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
**f**: **S-Box** (Substitute, Shrink)

- 48 bits ➔ 32 bits
  - 6 bits are used to select a 4-bit substitution
  - i.e., for every output, there are **four inputs** that map to it

```
  2 bits row
  /   \        \   /
 I1   I2   I3   I4   I5   I6

  4 bits column

  S_i

  \            /  \
  O1   O2   O3   O4

  an integer between 0 and 15

for i = 1, ..., 8
```
\( f: S_1 \) (Substitution)

Each row and column contain different numbers

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>…</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>E</td>
<td>4</td>
<td>D</td>
<td>1</td>
<td>2</td>
<td>F</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>F</td>
<td>7</td>
<td>4</td>
<td>E</td>
<td>2</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>E</td>
<td>8</td>
<td>D</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>C</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: input = 100110, output = 1000

for \( S_2..S_8 \) (and rest of \( S_1 \)), see the textbook
**f**: Permutation

- 32bits → 32bits

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>7</td>
<td>20</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>12</td>
<td>28</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>23</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>31</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>24</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>27</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>30</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>11</td>
<td>4</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
DES Implementation

• That’s it!

• Operations
  – Permutation
  – Swapping halves
  – Substitution (S-box, table lookup)
  – Bit discard
  – Bit replication
  – Circular shift
  – XOR

• Hard to implement? HW: No, SW: Yes
DES Analysis
Good Design?

• “We don’t know if
  – the particular details were well-chosen for strength,
  – whether someone flipped coins to construct the S-boxes,
    • Probably not, since choosing alternative S-boxes is known to weaken the strength
  – or whether the details were chosen to have a weakness that could be exploited by the designers.”
Issues for Block Ciphers

• Number of rounds should be large enough to make advanced attacks as expensive as exhaustive search for the key
Principles for S-Box Design

• S-box is the only non-linear part of DES
• Each row in the S-Box table should be a permutation of the possible output values
• Output of one S-box should affect other S-boxes in the following round
Desirable Property: Avalanche Effect

• Roughly: a small change in either the plaintext or the key should produce a big change in the ciphertext
• Better: any output bit should be inverted (flipped) with probability .5 if any input bit is changed

• $f$ function
  – must be difficult to un-scramble
  – should achieve avalanche effect
  – output bits should be uncorrelated
DES Avalanche Effect: Example

- 2 plaintexts with 1 bit difference:
  0x0000000000000000 and
  0x8000000000000000
  encrypted using the same key:
  0x016B24621C181C32
- Resulting ciphertexts differ in 34 bits (out of 64)
- Similar results when keys differ by 1 bit
Example (cont’d)

• An experiment: number of rounds vs. number of bits difference

<table>
<thead>
<tr>
<th>Round #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits changed</td>
<td>1</td>
<td>6</td>
<td>21</td>
<td>35</td>
<td>39</td>
<td>34</td>
<td>32</td>
<td>31</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>44</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>26</td>
<td>29</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>
DES: Keys to Avoid Using

• “Weak keys”: 4 keys with property
  \[ K\{K\{m\}\} = m \]

• These are keys which, after the first key permutation, are:
  – 28 0’s followed by 28 0’s
  – 28 0’s followed by 28 1’s
  – 28 1’s followed by 28 0’s
  – 28 1’s followed by 28 1’s

• What’s bad about that?
  – Rotation in subkey generation has no effect
  – \(2^{32}\) fixed points for encryption (i.e., \(K\{m\} = m\))
More Keys to Avoid!

• “Semi-weak keys”: pairs of keys with the property
  \[ K_1 \{ K_2 \{ m \} \} = m \]

• These are keys which, after the first key permutation, are:
  1. 28 0’s followed by alternating 0’s and 1’s
  2. 28 0’s followed by alternating 1’s and 0’s
     ...
  12. alternating 1’s and 0’s followed by alternating 1’s and 0’s

• What’s bad about that?
  – \(2^{32}\) fixed points for encryption (i.e., \(K\{m\} = \overline{m}\))
DES Key Size

- 56 bits is currently too small to resist brute force attacks using readily-available hardware
- Ten years ago it took $250,000 to build a machine that could crack DES in a few hours
- Now?
Cryptanalysis of DES

- **Differential cryptanalysis** exploits differences between encryptions of two different plaintext blocks
  - provides insight into possible key values
  - DES well designed to defeat differential analysis

- **Linear cryptanalysis** requires known plaintext / ciphertext pairs, analyzes relationships to discover key value
  - for DES, requires analyzing $O(2^{47})$ pairs

- No attacks on DES so far are significantly better than brute force attacks, for comparable cost
AES
Overview

• Selected from an open competition, organized by NSA
  – winner: Rijndael algorithm, standardized as AES
• Some similarities to DES (rounds, round keys, alternate permutation+substitution)
  – but not a Feistel cipher
• Block size = 128 bits
• Key sizes = 128, 192, or 256
• Main criteria: secure, well justified, fast
AES-128 Overview

- **Q1:** What happens in each round?
- **Q2:** How are round keys generated?
AES-128 State

- Each plaintext block of 16 bytes is arranged as 4 columns of 4 bytes each

\[
\begin{array}{cccccccccccccccc}
\text{a}_0 & \text{a}_1 & \text{a}_2 & \text{a}_3 & \text{a}_4 & \text{a}_5 & \text{a}_6 & \text{a}_7 & \text{a}_8 & \text{a}_9 & \text{a}_{10} & \text{a}_{11} & \text{a}_{12} & \text{a}_{13} & \text{a}_{14} & \text{a}_{15} \\
\end{array}
\]

(Padding necessary for messages not a multiple of 16 bytes)
One AES-128 Round

1. Apply **S-box** function to each byte of the state (i.e., 16 substitutions)
2. **Rotate**…
   - (row 0 of state is unchanged)
   - row 1 of the state left 1 column
   - row 2 of the state left 2 columns
   - row 3 of the state left 3 columns
3. Apply **MixColumn** function to each column of state
   - last round omits this step
Round Step 1. AES S-Box

- Each byte of state is replaced by a value from following table
  - eg. byte with value \texttt{0x95} is replaced by byte in row 9 column 5, which has value \texttt{0x2A}
S-Box (Cont’d)

The S-Box is what makes AES a non-linear cipher

For every value of \( b \) there is a unique value for \( b' \)
- It is faster to use a substitution table (and easier).

\[
\begin{array}{cccccccc}
& b'_0 & b'_1 & b'_2 & b'_3 & b'_4 & b'_5 & b'_6 & b'_7 \\
\end{array} = \begin{array}{cccccccc}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \begin{array}{c}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
\end{array} + \begin{array}{c}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0 \\
\end{array}
\]

\[ x = b^{-1} \text{ in } GF(2^8), \text{ i.e., } x \text{ is the inverse of byte } b \]
S-Box Example

• The S-Box is what makes AES a non-linear cipher

State

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>10</td>
<td>D0</td>
<td>81</td>
</tr>
<tr>
<td>60</td>
<td>20</td>
<td>4A</td>
<td>93</td>
</tr>
<tr>
<td>70</td>
<td>30</td>
<td>E1</td>
<td>A1</td>
</tr>
<tr>
<td>00</td>
<td>C0</td>
<td>F7</td>
<td>AF</td>
</tr>
</tbody>
</table>

Sbox(50) Sbox(10) Sbox(D0) Sbox(81)
Sbox(60) Sbox(20) Sbox(4A) Sbox(93)
Sbox(70) Sbox(30) Sbox(E1) Sbox(A1)
Sbox(00) Sbox(C0) Sbox(F7) Sbox(AF)

After SubBytes

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>CA</td>
<td>70</td>
<td>0C</td>
</tr>
<tr>
<td>D0</td>
<td>B7</td>
<td>D6</td>
<td>DC</td>
</tr>
<tr>
<td>51</td>
<td>04</td>
<td>F8</td>
<td>32</td>
</tr>
<tr>
<td>63</td>
<td>BA</td>
<td>68</td>
<td>79</td>
</tr>
</tbody>
</table>
## Round Step 2. Rotate (Example)

<table>
<thead>
<tr>
<th>Before Shift Rows</th>
<th>After Shift Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Before Shift Rows Grid" /></td>
<td><img src="image2.png" alt="After Shift Rows Grid" /></td>
</tr>
</tbody>
</table>

*Note: Images show the before and after states of the Shift Rows step.*
Round Step 3. MixColumn Function

• Applied to each column of the state
• For each column, each byte $a_i \ldots a_{i+3}$ of the column is used to look up four 4-byte intermediate columns $t_i \ldots t_{i+3}$ from a table (next slide)
• The intermediate columns $t_i \ldots t_{i+3}$ are then combined (next slide + 1):
  – rotate vertically so top octet of $t_i$ is in same row as input octet ($a_i$)
  – XOR the four rotated columns together
MixColumn… (Cont’d)

- Part of the MixColumn table:

right (low-order) nibble (4 bits)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0a</td>
<td>0b</td>
<td>0c</td>
<td>0d</td>
<td>0e</td>
<td>0f</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1a</td>
<td>1b</td>
<td>1c</td>
<td>1d</td>
<td>1e</td>
<td>1f</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>3a</td>
<td>3b</td>
<td>3c</td>
<td>3d</td>
<td>3e</td>
<td>3f</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>5a</td>
<td>5b</td>
<td>5c</td>
<td>5d</td>
<td>5e</td>
<td>5f</td>
</tr>
</tbody>
</table>

left (high-order) nibble (4 bits)

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>b9</th>
<th>bf</th>
<th>bd</th>
<th>b3</th>
<th>b1</th>
<th>b7</th>
<th>b5</th>
<th>ab</th>
<th>a9</th>
<th>af</th>
<th>ad</th>
<th>a3</th>
<th>a1</th>
<th>a7</th>
<th>a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>d9</td>
<td>df</td>
<td>dd</td>
<td>d3</td>
<td>d1</td>
<td>d7</td>
<td>d5</td>
<td>cb</td>
<td>c9</td>
<td>cf</td>
<td>cd</td>
<td>c3</td>
<td>c1</td>
<td>c7</td>
<td>c5</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>e0</td>
<td>e1</td>
<td>e2</td>
<td>e3</td>
<td>e4</td>
<td>e5</td>
<td>e7</td>
<td>e8</td>
<td>e9</td>
<td>ea</td>
<td>eb</td>
<td>ec</td>
<td>ed</td>
<td>ee</td>
<td>ef</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>f9</td>
<td>ff</td>
<td>fd</td>
<td>f3</td>
<td>f1</td>
<td>f7</td>
<td>f5</td>
<td>eb</td>
<td>e9</td>
<td>ef</td>
<td>ed</td>
<td>e3</td>
<td>e1</td>
<td>e7</td>
<td>e5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f0</td>
<td>f1</td>
<td>f2</td>
<td>f3</td>
<td>f4</td>
<td>f5</td>
<td>f6</td>
<td>f7</td>
<td>f8</td>
<td>f9</td>
<td>fa</td>
<td>fb</td>
<td>fc</td>
<td>fd</td>
<td>fe</td>
<td>ff</td>
</tr>
</tbody>
</table>
MixColumn… (Cont’d)

- Example
Generating Round Keys in AES-128

The key (16 bytes) is arranged in 4 columns of 4 rows, as for the input (plaintext) block.

Deriving the round keys makes use of a table of constants:

Removes symmetry and linearity from key expansion

<table>
<thead>
<tr>
<th>Round i</th>
<th>Constant $c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x6C</td>
</tr>
<tr>
<td>2</td>
<td>0xD8</td>
</tr>
<tr>
<td>3</td>
<td>0xAB</td>
</tr>
<tr>
<td>4</td>
<td>0x4D</td>
</tr>
<tr>
<td>5</td>
<td>0x9A</td>
</tr>
<tr>
<td>6</td>
<td>0x2F</td>
</tr>
<tr>
<td>7</td>
<td>0x5E</td>
</tr>
<tr>
<td>8</td>
<td>0xBC</td>
</tr>
<tr>
<td>9</td>
<td>0x63</td>
</tr>
<tr>
<td>10</td>
<td>0xC6</td>
</tr>
</tbody>
</table>
Round Keys… (Cont’d)

For $i^{th}$ round of keys, $i = 1..10$

for column index $j = 0$
  temp = column 3 of $(i-1)^{th}$ (previous) round
  rotate temp upward one byte
  S-Box transform each byte of temp
  XOR first byte of temp with $c_i$

for column index $j = 1..3$
  temp = column $j-1$ of $i^{th}$ (this) round

result = temp XOR $j^{th}$ column of key round $i-1$
Key Expansion Rationale

• Designed to resist known attacks
• Design criteria include
  – knowing part of the key doesn’t make it easy to find entire key
  – key expansion must be invertible, but enough non-linearity to hinder analysis
  – should be fast to compute, simple to describe and analyze
  – key bits should be diffused into the round keys
AES Operates on the binary field $GF(2^8)$

- this can be represented as a polynomial $b(x)$ with binary coefficients $b \in \{0,1\}$:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

Multiplication in $GF(2^8)$ consists of multiplying two polynomials modulo an irreducible polynomial of degree 8

- AES uses the following irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1$$
AES-128 Decryption (Conceptual)

- Run cipher in reverse, with inverse of each operation replacing the encryption operations.
- Inverse operations:
  - XOR is its own inverse
  - inverse of S-box is just the inverse table (next slide)
  - inverse of rotation in one direction is rotation in other direction
  - inverse of MixColumn is just the inverse table (next slide + 1)
## Inverse S-Box

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>52 9 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb</td>
</tr>
<tr>
<td>1</td>
<td>7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb</td>
</tr>
<tr>
<td>2</td>
<td>54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e</td>
</tr>
<tr>
<td>3</td>
<td>8 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25</td>
</tr>
<tr>
<td>4</td>
<td>72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92</td>
</tr>
<tr>
<td>5</td>
<td>6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84</td>
</tr>
<tr>
<td>6</td>
<td>90 d8 ab 0 8c bc d3 0a f7 e4 58 5 b8 b3 45 6</td>
</tr>
<tr>
<td>7</td>
<td>d0 2c 1e 8f ca 3f 0f 2 c1 af bd 3 1 13 8a 6b</td>
</tr>
<tr>
<td>8</td>
<td>3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73</td>
</tr>
<tr>
<td>9</td>
<td>96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e</td>
</tr>
<tr>
<td>a</td>
<td>47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b</td>
</tr>
<tr>
<td>b</td>
<td>fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4</td>
</tr>
<tr>
<td>c</td>
<td>1f dd a8 33 88 7 c7 31 b1 12 10 59 27 80 ec 5f</td>
</tr>
<tr>
<td>d</td>
<td>60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef</td>
</tr>
<tr>
<td>e</td>
<td>a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61</td>
</tr>
<tr>
<td>f</td>
<td>17 2b 4 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d</td>
</tr>
</tbody>
</table>
## InvMixColumn

**right (low-order) nibble (4 bits)**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0e</td>
<td>1c</td>
<td>12</td>
<td>38</td>
<td>36</td>
<td>24</td>
<td>2a</td>
<td>70</td>
<td>7e</td>
<td>6c</td>
<td>52</td>
<td>48</td>
<td>46</td>
<td>54</td>
<td>5a</td>
</tr>
<tr>
<td>00</td>
<td>09</td>
<td>12</td>
<td>1b</td>
<td>24</td>
<td>2d</td>
<td>36</td>
<td>3f</td>
<td>48</td>
<td>41</td>
<td>5a</td>
<td>53</td>
<td>6c</td>
<td>65</td>
<td>7e</td>
<td>77</td>
</tr>
<tr>
<td>00</td>
<td>0d</td>
<td>1a</td>
<td>17</td>
<td>34</td>
<td>39</td>
<td>2e</td>
<td>23</td>
<td>68</td>
<td>65</td>
<td>72</td>
<td>7f</td>
<td>5c</td>
<td>51</td>
<td>46</td>
<td>4b</td>
</tr>
<tr>
<td>00</td>
<td>0b</td>
<td>16</td>
<td>1d</td>
<td>2c</td>
<td>27</td>
<td>3a</td>
<td>31</td>
<td>58</td>
<td>53</td>
<td>4e</td>
<td>45</td>
<td>74</td>
<td>7f</td>
<td>62</td>
<td>69</td>
</tr>
<tr>
<td>00</td>
<td>e0</td>
<td>ee</td>
<td>fc</td>
<td>f2</td>
<td>d8</td>
<td>d6</td>
<td>c4</td>
<td>ca</td>
<td>90</td>
<td>9e</td>
<td>8c</td>
<td>82</td>
<td>a8</td>
<td>a6</td>
<td>b4</td>
</tr>
<tr>
<td>00</td>
<td>90</td>
<td>99</td>
<td>82</td>
<td>8b</td>
<td>b4</td>
<td>bd</td>
<td>a6</td>
<td>af</td>
<td>d8</td>
<td>d1</td>
<td>ca</td>
<td>c3</td>
<td>fc</td>
<td>f5</td>
<td>ee</td>
</tr>
<tr>
<td>00</td>
<td>dd</td>
<td>ca</td>
<td>c7</td>
<td>e4</td>
<td>e9</td>
<td>fe</td>
<td>f3</td>
<td>b8</td>
<td>b5</td>
<td>a2</td>
<td>af</td>
<td>8c</td>
<td>81</td>
<td>96</td>
<td>9b</td>
</tr>
<tr>
<td>00</td>
<td>b0</td>
<td>bb</td>
<td>a6</td>
<td>ad</td>
<td>9c</td>
<td>97</td>
<td>8a</td>
<td>81</td>
<td>e8</td>
<td>e3</td>
<td>fe</td>
<td>f5</td>
<td>c4</td>
<td>cf</td>
<td>d2</td>
</tr>
<tr>
<td>00</td>
<td>db</td>
<td>d5</td>
<td>c7</td>
<td>c9</td>
<td>e3</td>
<td>ed</td>
<td>ff</td>
<td>f1</td>
<td>ab</td>
<td>a5</td>
<td>b7</td>
<td>b9</td>
<td>93</td>
<td>9d</td>
<td>8f</td>
</tr>
<tr>
<td>00</td>
<td>3b</td>
<td>32</td>
<td>29</td>
<td>20</td>
<td>1f</td>
<td>16</td>
<td>0d</td>
<td>04</td>
<td>73</td>
<td>7a</td>
<td>61</td>
<td>68</td>
<td>57</td>
<td>5e</td>
<td>45</td>
</tr>
<tr>
<td>00</td>
<td>bb</td>
<td>b6</td>
<td>a1</td>
<td>ac</td>
<td>8f</td>
<td>82</td>
<td>95</td>
<td>98</td>
<td>d3</td>
<td>de</td>
<td>c9</td>
<td>c4</td>
<td>e7</td>
<td>ea</td>
<td>fd</td>
</tr>
<tr>
<td>00</td>
<td>7b</td>
<td>70</td>
<td>6d</td>
<td>66</td>
<td>57</td>
<td>5c</td>
<td>41</td>
<td>4a</td>
<td>23</td>
<td>28</td>
<td>35</td>
<td>3e</td>
<td>0f</td>
<td>04</td>
<td>19</td>
</tr>
<tr>
<td>00</td>
<td>3b</td>
<td>35</td>
<td>27</td>
<td>20</td>
<td>1f</td>
<td>16</td>
<td>0d</td>
<td>04</td>
<td>11</td>
<td>e6</td>
<td>1a</td>
<td>4b</td>
<td>45</td>
<td>57</td>
<td>50</td>
</tr>
</tbody>
</table>

**left (high-order) nibble (4 bits)**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0a</td>
<td>17</td>
<td>1e</td>
<td>34</td>
<td>3a</td>
<td>2a</td>
<td>26</td>
<td>7c</td>
<td>72</td>
<td>60</td>
<td>6e</td>
<td>44</td>
<td>4a</td>
<td>58</td>
<td>56</td>
</tr>
<tr>
<td>0a</td>
<td>03</td>
<td>18</td>
<td>11</td>
<td>2e</td>
<td>27</td>
<td>3c</td>
<td>35</td>
<td>42</td>
<td>4b</td>
<td>50</td>
<td>59</td>
<td>66</td>
<td>6f</td>
<td>74</td>
<td>7d</td>
</tr>
<tr>
<td>00</td>
<td>d7</td>
<td>39</td>
<td>2b</td>
<td>25</td>
<td>0f</td>
<td>01</td>
<td>13</td>
<td>1d</td>
<td>47</td>
<td>49</td>
<td>5b</td>
<td>55</td>
<td>7f</td>
<td>71</td>
<td>63</td>
</tr>
<tr>
<td>00</td>
<td>41</td>
<td>a8</td>
<td>b3</td>
<td>ba</td>
<td>85</td>
<td>8c</td>
<td>97</td>
<td>9e</td>
<td>e9</td>
<td>e0</td>
<td>fb</td>
<td>f2</td>
<td>cd</td>
<td>c4</td>
<td>df</td>
</tr>
<tr>
<td>00</td>
<td>0c</td>
<td>01</td>
<td>16</td>
<td>1b</td>
<td>38</td>
<td>35</td>
<td>22</td>
<td>2f</td>
<td>64</td>
<td>69</td>
<td>7e</td>
<td>73</td>
<td>50</td>
<td>5d</td>
<td>4a</td>
</tr>
<tr>
<td>00</td>
<td>7a</td>
<td>71</td>
<td>6c</td>
<td>67</td>
<td>56</td>
<td>5d</td>
<td>40</td>
<td>4b</td>
<td>22</td>
<td>29</td>
<td>34</td>
<td>3f</td>
<td>0e</td>
<td>05</td>
<td>18</td>
</tr>
<tr>
<td>00</td>
<td>d7</td>
<td>d9</td>
<td>cb</td>
<td>c5</td>
<td>ef</td>
<td>e1</td>
<td>f3</td>
<td>fd</td>
<td>a7</td>
<td>a9</td>
<td>bb</td>
<td>b5</td>
<td>9f</td>
<td>91</td>
<td>83</td>
</tr>
<tr>
<td>00</td>
<td>31</td>
<td>38</td>
<td>23</td>
<td>2a</td>
<td>15</td>
<td>1c</td>
<td>07</td>
<td>0e</td>
<td>79</td>
<td>70</td>
<td>6b</td>
<td>62</td>
<td>5d</td>
<td>54</td>
<td>4f</td>
</tr>
<tr>
<td>00</td>
<td>dc</td>
<td>d1</td>
<td>c6</td>
<td>cb</td>
<td>e8</td>
<td>e5</td>
<td>82</td>
<td>ff</td>
<td>b4</td>
<td>b9</td>
<td>ae</td>
<td>a3</td>
<td>80</td>
<td>8d</td>
<td>9a</td>
</tr>
<tr>
<td>00</td>
<td>ca</td>
<td>c1</td>
<td>dc</td>
<td>d7</td>
<td>e6</td>
<td>ed</td>
<td>f0</td>
<td>fb</td>
<td>92</td>
<td>99</td>
<td>84</td>
<td>8f</td>
<td>be</td>
<td>b5</td>
<td>a8</td>
</tr>
</tbody>
</table>
AES Decryption (Actual)

- Run cipher in **forward** direction, except…
  - use inverse operations
  - apply round keys in reverse order
  - apply InvMixColumn to round keys K1..K9

- Decryption takes more memory and cycles encryption
  - can only partially reuse hardware for encryption
AES Assessment

• Speed: about 16 clock cycles/byte on modern 32-bit CPUs
  – 200 MByte/s on a PC, no special hardware!

• No known successful attacks on full AES
  – best attacks work on 7-9 rounds (out of 10-14 rounds)

• Clean design

• For brute force attacks, AES-128 will take
  \(4 \times 10^{21} X \ (= 2^{72})\) more effort than DES
Attacks on AES

**Differential Cryptanalysis**: based on how differences in inputs correlate with differences in outputs
- greatly reduced due to high number of rounds

**Linear Cryptanalysis**: based on correlations between input and output
- S-Box & MixColumns are designed to frustrate Linear Analysis

**Side Channel Attacks**: based on peculiarities of the implementation of the cipher
Side Channel Attacks

**Timing Attacks:** measure the time it takes to do operations
- some operations, with some operands, are much faster than other operations, with other operand values
- provides clues about what internal operations are being performed, and what internal data values are being produced

**Power Attacks:** measures power to do operations
- changing one bit requires considerably less power than changing many bits in a byte
Summary

- Secret key crypto is (a) good quality, (b) faster to compute than public key crypto, and (c) the most widely used crypto
- DES strong enough for non-critical applications, but triple-DES is better
- AES even better (stronger and much faster), has versions with 128-, 192-, and 256-bit keys
- Secret key crypto requires “out-of-band”, bilateral key negotiation/agreement