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>$X</td>
<td>Y$</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 <plaintext, key> input and the <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 \rightarrow P \rightarrow S \rightarrow P \rightarrow S \rightarrow P \rightarrow S \rightarrow ...
  - $P \rightarrow S \rightarrow P \rightarrow S \rightarrow P \rightarrow S \rightarrow P \rightarrow ...

- Do they have to alternate? e.g.,
  - $S \rightarrow S \rightarrow S \rightarrow P \rightarrow P \rightarrow P \rightarrow S \rightarrow S \rightarrow ...

- 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**
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!

Complete Feistel Cipher: Encryption

- Note the final swap!
Feistel Cipher: Decryption

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
  - Reversibility 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)

- 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 :-)

DEPARTMENT OF COMPUTER SCIENCE

CSC 474
Dr. Peng Ning
DES Basics

• Blocks: 64 bit plaintext input, 64 bit ciphertext output
• Rounds: 16
• Key: 64 bits
  – every 8th bit is a parity bit, so really 56 bits long

DES Top Level View

Initial and Final Permutations

• Initial permutation given below
  – input bit #58 → output bit #1, input bit #50 → output bit #2, ...

<table>
<thead>
<tr>
<th>Input</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></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></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></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></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></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></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></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>
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
  - …

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>57 49 41 33 25 17 9</th>
<th>57 49 41 33 25 17 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 58 50 42 34 26 18</td>
<td>57 49 41 33 25 17 9</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

- Circular Left Shift
- Permutation with Discard

Rounds $i = 1, 2, 9, 16$: left circular shift 1 bit
Other rounds: left circular shift 2 bits

KeyGen: Permutation with Discard

- 28 bits $\rightarrow$ 24 bits, each half of key
- Left half of $K_i$ = permutation of $C_i$
- Right half of $K_i$ = permutation of $D_i$

Bits left out: $9, 18, 22, 25$
Bits left out: $35, 38, 43, 54$

One DES (Feistel) Round

- Input block $i$
- Output block $i+1$
- Generate round keys
- 64-bit Input
- 56-bit Key
- 64-bit Output
DES Round: $f$ (Mangler) Function

Input block $i$

$\begin{align*}
\text{L}_i & \quad \text{R}_i \\
\text{L}_{i+1} & \quad \text{R}_{i+1}
\end{align*}$

Output block $i+1$

$\text{L}_{i+1} \oplus \text{R}_{i+1}$

Function $f = \text{“Mangler”}$

32-bit half block

Expansion

48 bits

S-Box (substitution)

Permutation

32-bit half block

$K_i$

32-bit half block

Input block $i$

32-bit half block

$K_i$

32-bit half block

Expansion

Function $f$:

• 32 bits $\Rightarrow$ 48 bits

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

$S_i$ for $i = 1, \ldots, 8$

$\begin{align*}
\text{I}_1 & \quad \text{I}_2 \quad \text{I}_3 \quad \text{I}_4 \\
\text{O}_1 & \quad \text{O}_2 \quad \text{O}_3 \quad \text{O}_4
\end{align*}$

$\text{I}_1 \quad \text{I}_2 \quad \text{I}_3 \quad \text{I}_4$

$\text{O}_1 \quad \text{O}_2 \quad \text{O}_3 \quad \text{O}_4$

$f$: S-Box (Substitute, Shrink)

• 48 bits $\Rightarrow$ 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
Each row and column contain different numbers

<table>
<thead>
<tr>
<th>0</th>
<th>E</th>
<th>4</th>
<th>D</th>
<th>1</th>
<th>2</th>
<th>F</th>
<th>B</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</tr>
</tbody>
</table>

Example: input = 100110, output = 1000

*for S_2, S_3 (and rest of S_j), see the textbook*

---

### f: Permutation

- 32bits \(\rightarrow\) 32bits

```
16  7  20  21
29  12  28  17
  1  15  23  26
  5  18  31  10
  2  8  24  14
32  27  3  9
19  13  30  6
22  11  4  25
```

---

### 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>32</td>
<td>31</td>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>44</td>
<td>32</td>
<td>30</td>
<td>26</td>
<td>29</td>
<td>34</td>
<td></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., \(K[m] = m\))
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

(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 0x95 is replaced by byte in row 9 column 5, which has value 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).

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

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

**State**

| 50 | 10 | 00 | 81 |
| 60 | 30 | 4A | 9A |
| 70 | 0C | FF | AB |

After SubBytes

| 53 | CA | 70 | 0C |
| D0 | B7 | D6 | DC |
| 51 | 04 | F8 | 32 |
| 63 | BA | 68 | 79 |

Round Step 2. Rotate (Example)

**Before Shift Rows**

| 53 | CA | 70 | 0C |
| D0 | B7 | D6 | DC |
| 51 | 04 | F8 | 32 |
| 63 | BA | 68 | 79 |

**After Shift Rows**

| 53 | CA | 70 | 0C |
| 87 | B6 | Dc | D9 |
| F8 | 32 | 51 | 04 |
| 79 | 63 | BA | 68 |

Round Step 3. MixColumn Function

- Applied to each column of the state
- For each column, each byte $a_i$…$a_{i+3}$ of the column is used to look up four 4-byte intermediate columns $t_i$…$t_{i+3}$ from a table (next slide)
- The intermediate columns $t_i$…$t_{i+3}$ 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)

left (high-order) nibble (4 bits)

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
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

Mathematics

**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>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
</tr>
<tr>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

InvMixColumn

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
</tr>
<tr>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></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 $4 \times 10^{21} \cdot X = 2^{72}$ 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