Lecture 4

Encryption Continued...

Data Encryption Standard (DES)

- 64 bit input block
- 64 bit output block
- 16 rounds
- 64 (effective 56) bit key
- Key schedule computed at startup
- Aimed at bulk data
- >16 rounds doesn’t help
- >56 bit key doesn’t help
- Other S-boxes usually hurt...
Figure 2.3 General Depiction of DES Encryption Algorithm

Encryption Process:
- 64 Bit Plaintext
- Initial Permutation
- 32 Bit L0, 32 Bit R0
- F(R0, K1)
- 32 Bit L1, 32 Bit R1
- ... (16 rounds)
- F(R15, K16)
- 32 Bit L15, 32 Bit R15
- Final Permutation
- 64 Bit Ciphertext

Key Schedule:
- 64 Bit Key
- Permutation Choice 1
- 56 Bit Key
- 28 Bit C0, 28 Bit D0
- Ki(48 bits) Left Shift
- Permutation Choice 2
- Ki6(48 bits)
- C16, D16
- Right Shift

Building Blocks:
- F(Ri, Ki)
- F(Ri5, Ki6)
Function F

\[ \text{Expansion Permutation 48 bits} \]

\[ \text{S-Box Substitution choice 32 bits} \]

\[ \text{P-box Permutation} \]

\[ \text{Li: 32 bits} \]

\[ \text{Ri: 32 bits} \]

56 bits Key Permutation Choice 48 bits

DES Substitution Boxes Operation

Expanded \( R_{i-1} \oplus \text{Key} \)

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\( B_1 \) & \( B_2 \) & \( B_3 \) & \( B_4 \) & \( B_5 \) & \( B_6 \) & \( B_7 \) & \( B_8 \) &  \\
\hline
Bit & 1 & 6 & 7 & 12 & \cdots &  &  & 43 & 48  \\
\hline
\hline
Bits & \( S_1 \) & \( S_2 \) & \( S_3 \) & \( S_4 \) & \( S_5 \) & \( S_6 \) & \( S_7 \) & \( S_8 \) &  \\
\hline
\end{tabular}
Key Schedule -- KS

Key schedule of shifts

<table>
<thead>
<tr>
<th>Iteration(i)</th>
<th>No. of shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

Key permutation PC-1

| 37 | 49 | 41 | 33 | 25 | 17 | 9  |
| 1  | 58 | 50 | 42 | 34 | 36 | 18 |
| 10 | 2  | 59 | 51 | 43 | 35 | 27 |
| 19 | 11 | 3  | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 |
| 7  | 45 | 54 | 40 | 38 | 30 | 22 |
| 14 | 6  | 61 | 53 | 45 | 37 | 29 |
| 21 | 13 | 5  | 28 | 20 | 12 | 4  |

Key permutation PC-2

| 14 | 17 | 11 | 24 | 1 | 5 |
| 3  | 28 | 15 | 6  | 20 | 10 |
| 25 | 19 | 12 | 4  | 26 | 8 |
| 16 | 7  | 27 | 20 | 13 | 2 |
| 41 | 32 | 31 | 37 | 47 | 55 |
| 30 | 40 | 51 | 45 | 33 | 48 |
| 44 | 49 | 39 | 56 | 34 | 54 |
| 46 | 42 | 50 | 36 | 29 | 32 |

Operation Tables of DES
(Key Schedule, PC-1, PC-2)
Operation Tables of DES (IP, IP⁻¹, E and P)

**Initial Permutation (IP)**

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

**Bit-Selection Table E**

<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>3</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>

**Inverse Initial Permutation (IP⁻¹)**

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

**Permutation P**

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

---

**DES: Modes of Operation**

- **Electronic code-book (ECB)**
  \[ C_i = E(K, P_i) \]  \( \rightarrow \) Local error, permutation attack, parallel encr.

- **Chained block cipher (CBC)**
  \[ C_i = E(K, C_{i-1} \ XOR \ P_i) \]  \( \rightarrow \) Need IV, error causes 2-block loss, no parallel enc.

- **Output feedback (OFB)**
  \[ V_i = E(K, V_{i-1}) \ C_i = P_i \ XOR \ V_i \]  \( \rightarrow \) Stream cipher, local error, pre-computation

- **Cipher feedback (CFB)**
  \[ C_i = P_i \ XOR \ E(K, C_{i-1}) \]  \( \rightarrow \) Plaintext dependence, avalanche effect, parallel decryption.

- **Message Auth Code (MAC)**
  \[ P_1, \ldots, P_n, C_n \ (CBC) \]
CBC Mode

- Cipher Block Chaining Mode (CBC)
  - Input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block.
  - Repeating pattern of 64-bits are not exposed
  - Block rearrangement made difficult!

\[
C_i = E_K [C_{i-1} \oplus P_i] \\
D_K [C_i] = D_K [E_K (C_{i-1} \oplus P_i)] \\
D_K [C_i] = (C_{i-1} \oplus P_i) \\
C_{i-1} \oplus D_K [C_i] = C_{i-1} \oplus C_{i-1} \oplus P_i = P_i
\]

Figure 2.7 Cipher Block Chaining (CBC) Mode
Breaking DES (Cryptanalysis)

Differential Cryptanalysis

- Looks for correlations in $f()$-function input and output

Linear cryptanalysis

- Looks for correlations between key and cipher input and output

Related-key cryptanalysis

- Looks for correlations between key changes and cipher input/output

Differential cryptanalysis discovered in 1990; virtually all block ciphers from before that time are vulnerable...

...except DES. IBM (and the NSA) knew about it 15 years earlier

Breaking DES (Cryptanalysis)

DES Key size = 56 bits

- Brute force = $2^{55}$ attempts
- Differential cryptanalysis = $2^{47}$ attempts
- Linear cryptanalysis = $2^{43}$ attempts

- Longer than 56 bit keys don’t make it any stronger
- More than 16 rounds don’t make it any stronger

DES Key Problems:

- Weak keys (all 0s, all 1s, a few others)
- Key size = 56 bits = $8 \times 7$-bit ASCII
- Alphanumeric-only password converted to uppercase
  $8 \times \sim5$-bit chars = 40 bits
**DES Variants**

- **3-DES (triple DES)**
  - \( C = E(K_1, D(K_2, E(K_1, P)) \) \) → 112 effective key bits
  - \( C = E(K_3, D(K_2, E(K_1, P)) \) \) → 168 effective key bits

- **DESx**
  - \( C = K_3 \text{ XOR } E(K_2, (K_1 \text{ XOR } P)) \) → seems like 184 key bits
  - Effective key bits → approx. 118

- **2-DES:**
  - \( C = E(K_2, E(K_1, P)) \)

- **Another simple variation:**
  - \( C = K_1 \text{ XOR } E(K_1', P) \) → weak!

---

**Lecture 5**

Encryption Continued...
DES Variants

- **3-DES (triple DES)**
  - $C = E(K_1, D(K_2, E(K_1, P))) \rightarrow 112$ effective key bits
  - $C = E(K_3, D(K_2, E(K_1, P))) \rightarrow 168$ effective key bits

- **DESx**
  - $C = K_3 \ XOR \ E(K_2, (K_1 \ XOR \ P)) \rightarrow$ seems like $184$ key bits
  - Effective key bits $\rightarrow$ approx. $118$

- **2-DES**
  - $C = E(K_2, E(K_1, P))$

- **Another simple variation:**
  - $C = K_1 \ XOR \ E(K_1', P) \rightarrow$ weak!

---

Why not 2-DES?

- **2DES:** $C = DES(K_1, DES(K_2, P))$
- Seems to be hard to break by “brute force”, approx. $2^{111}$ trials
- Assume Eve is trying to break 2DES and has a single $(P, C)$ pair

**Meet-in-the-middle (or Rendesvouz) ATTACK:**

I. For each possible $K'_i$ (where $0 < i < 2^{56}$)
   1. Compute $C'_i = DES(K'_i, P)$
   2. Store: $[K'_i, C'_i]$ in table $T$ (sorted by $C'_i$)

II. For each possible $K''_i$ (where $0 < i < 2^{56}$)
   1. Compute $C''_i = DES^{-1}(K''_i, C)$
   2. Lookup $C''_i$ in $T \leftarrow$ not expensive!
   3. If lookup succeeds, output: $K_1=K'_i, K_2=K''_i$

**TOTAL COST:** $O(2^{56})$ operations + $O(2^{56})$ storage
DES Variants

- 3-DES (triple DES)
  - $C = E(K_1, D(K_2, E(K_1, P))) \rightarrow 112$ effective key bits
  - $C = E(K_3, D(K_2, E(K_1, P))) \rightarrow 168$ effective key bits

- DESx
  - $C = K_3 \ XOR \ E(K_2, (K_1 \ XOR \ P)) \rightarrow \text{seems like 184 key bits}$
  - Effective key bits $\rightarrow$ approx. 118

- 2-DES:
  - $C = E(K_2, E(K_1, P)) \rightarrow \text{rendezvous (meet-in-the-middle attack)}$

- Another simple variation:
  - $C = K_1 \ XOR \ E(K_1', P) \rightarrow \text{weak!}

Why does 3-DES (or generally n-DES) work?

Because, as a function, DES is not a group...

A “group” is an algebraic structure. One of its properties is that,
  - taking any 2 elements of the group $(a, b)$ and applying an operator $F()$ yields another element $c$ in the group.

Suppose: $C = \text{DES}(K_1, \text{DES}(K_2, P))$

There is no $K$, such that:

for each possible plaintext $P$, $\text{DES}(K, P) = C$
DES summary

- Permutation/substitution block cipher
- 64-bit data blocks
- 56-bit keys (8 parity bits)
- 16 rounds (shifts, XORs)
- Key schedule
- S-box selection secret...
- DES “aging”
- 2-DES: rendezvous attack
- 3-DES: 112-bit security
- DESx : 118-bit security

Other Symmetric Ciphers

Skipjack
- Classified algorithm originally designed for Clipper,
- declassified in 1998
- 32 rounds, breakable with 31 rounds
- 80 bit key, inadequate for long-term security

GOST
- GOST 28147, Russian answer to DES
- 32 rounds, 256 bit key
- Incompletely specified
### Other Symmetric Ciphers

- **IDEA (X. ILai, J. Massey, ETH)**
  - Developed as PES (proposed encryption standard),
  - adapted to resist differential cryptanalysis
  - Gained popularity via PGP, 128 bit key
  - Patented (Ascom CH)

- **Blowfish (B. Schneier, Counterpane)**
  - Optimized for high-speed execution on 32-bit processors
  - 448 bit key, relatively slow key setup
  - Fast for bulk data on most PCs/laptops
  - Easy to implement, runs in ca. 5K of memory

**Other Symmetric Ciphers**

**RC4 (Ron’s Cipher #4) Stream cipher:**
- Optimized for fast software implementation
- Character streaming (not bit)
- 8-bit output
- Former trade secret of RSADSI,
- Reverse-engineered and posted to the net in 1994:
- 2048-bit key
- Used in many products until about 1999-2000
Other Symmetric Ciphers (RC4)

\[ x = y = 0; \]

while( length-- )
{
    /* state[0-255] contains key bytes */
    sx = state[ ++x & 0xFF ];
    y += sx & 0xFF;
    sy = state[ y ];
    state[ y ] = sx;
    state[ x ] = sy;
    *data++ ^= state[ ( sx+sy ) & 0xFF ];
}

Takes about a minute to implement from memory

Other Symmetric Ciphers

- RC5
  - Suitable for hardware and software
  - Fast, simple
  - Adaptable to processors of different word lengths
  - Variable number of rounds
  - Variable-length key (0-256 bytes)
  - Very low memory requirements
  - High security (no effective attacks, yet...)
  - Data-dependent rotations
Other Symmetric Ciphers

• RC5 single round pseudocode:

\[
\begin{align*}
L & \leftarrow L \text{ XOR } R \\
L & \leftarrow L \ll R \\
L & \leftarrow L + \text{subkey}[2i] \\
R & \leftarrow R \text{ XOR } L \\
R & \leftarrow R \ll L \\
R & \leftarrow R + \text{subkey}[2i + 1]
\end{align*}
\]

AES:
The Rijndael Block Cipher
Introduction and History

- National Institute of Science and Technology (NIST) regulates standardization in the US
- DES is an aging standard that no longer meets today's needs for strong encryption
- Triple-DES: Endorsed by NIST as a "de facto" standard
- AES: Advanced Encryption Standard
  - Finalized in 2001
  - Goal is to define the Federal Information Processing Standard (FIPS) by selecting a new encryption algorithm suitable for encrypting (non-classified non-military) government documents
  - Candidate algorithms must be:
    - Symmetric-key ciphers supporting 128, 192, and 256 bit keys
    - Royalty-Free
    - Unclassified (i.e. public domain)
    - Available for worldwide export

Introduction and History

- AES Round-3 Finalist Algorithms:
  - MARS
    - Candidate offering from IBM Research
  - RC6
    - By Ron Rivest of MIT & RSA Labs, creator of the widely used RC4/RC5 algorithm and "R" in RSA
  - Twofish
    - From Counterpane Internet Security, Inc. (MN)
  - Serpent
    - by Ross Anderson (UK), Eli Biham (ISR) and Lars Knudsen (NO)
  - Rijndael
    - by Joan Daemen and Vincent Rijmen (B)
The Winner: Rijndael

- Joan Daemen (of Proton World International) and Vincent Rijmen (of Katholieke Universiteit Leuven).
- pronounced “Rhine-doll”
- Allows only 128, 192, and 256-bit key sizes (unlike other candidates)
- Variable input block length: 128, 192, or 256 bits. All nine combinations of key-block length possible.
  - A block is the smallest data size the algorithm will encrypt
- Vast speed improvement over DES in both hw and sw implementations
  - 8,416 bytes/sec on a 20MHz 8051
  - 8.8 Mbytes/sec on a 200MHz Pentium Pro

Key Expansion

- Key is expanded to a set of n round keys
- Input block P put thru n rounds, each with a distinct round sub-key.
- Strength of algorithm relies on difficulty of obtaining intermediate results (or state) of round i from round i+1 without the round key.
Each round performs the following operations:
- Non-linear Layer: No linear relationship between the input and output of a round
- Linear Mixing Layer: Guarantees high diffusion over multiple rounds
  - Very small correlation between bytes of the round input and the bytes of the output
- Key Addition Layer: Bytes of the input are simply XOR’ed with the expanded round key

Rijndael provides “full diffusion” after only two rounds

Immune to:
- Linear and differential cryptanalysis
- Related-key attacks
- Square attack
- Interpolation attacks
- Weak keys

Rijndael has been “shown” secure:
- No key recovery attacks faster than exhaustive search exist
- No known symmetry properties in the round mapping
- No weak keys identified
- No related-key attacks: No two keys have a high number of expanded round keys in common
Rijndael: ByteSub (192)

Each byte at the input of a round undergoes a non-linear byte substitution according to the following transform:

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  y_5 \\
  y_6 \\
  y_7
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
  1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
  1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
  1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
  0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
  0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
  0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7
\end{bmatrix}
\]

Substitution ("S")-box

Rijndael: ShiftRow

Depending on the block length, each "row" of the block is cyclically shifted according to the above table:
Rijndael: MixColumn

Each column is multiplied by a fixed polynomial
\( C(x) = \text{`03'*X}^3 + \text{`01'*X}^2 + \text{`01'*X} + \text{`02'} \)

This corresponds to matrix multiplication \( b(x) = c(x) \otimes a(x) \):

\[
\begin{bmatrix}
  b_0 \\ b_1 \\ b_2 \\ b_3
\end{bmatrix} =
\begin{bmatrix}
  02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 02 & 03 & 03 \\ 03 & 01 & 01 & 02
\end{bmatrix}
\begin{bmatrix}
  a_0 \\ a_1 \\ a_2 \\ a_3
\end{bmatrix}
\]

Not xor

Rijndael: Key Expansion and Addition

Each word is simply XOR'ed with the expanded round key

Key Expansion algorithm:

```c
KeyExpansion(int* Key[4*Nk], int* EKey[4* Nb*(Nr+1)])
{
    for(i = 0; i < Nk; i++)
        EKey[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++)
    {
        temp = EKey[i - 1];
        if (i % Nk == 0)
            temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
        EKey[i] = EKey[i - Nk] ^ temp;
    }
}
```
Rijndael: Implementations

- Well-suited for software implementations on 8-bit processors (important for "Smart Cards")
  - Atomic operations focus on bytes and nibbles, not 32- or 64-bit integers
  - Layers such as ByteSub can be efficiently implemented using small tables in ROM (e.g. < 256 bytes).
  - No special instructions are required to speed up operation, e.g. barrel rotates
- For 32-bit implementations:
  - An entire round can be implemented via a fast table lookup routine on machines with 32-bit or higher word lengths
  - Considerable parallelism exists in the algorithm
    - Each layer of Rijndael operates in a parallel manner on the bytes of the round state, all four component transforms act on individual parts of the block
    - Although the Key expansion is complicated and cannot benefit much from parallelism, it only needs to be performed once until the two parties switch keys.

Rijndael: Implementations

- Hardware Implementations
  - Rijndael performs very well in software, but there are cases when better performance is required (e.g. server and VPN applications).
  - Multiple S-Box engines, round-key XORs, and byte shifts can all be implemented efficiently in hardware when absolute speed is required
  - Small amount of hardware can vastly speed up 8-bit implementations
- Inverse Cipher
  - Except for the non-linear ByteSub step, each part of Rijndael has a straightforward inverse and the operations simply need to be undone in the reverse order.
  - However, Rijndael was specially written so that the same code that encrypts a block can also decrypt the same block simply by changing certain tables and polynomials for each layer. The rest of the operation remains identical.
Conclusions and The Future

• Rijndael is an extremely fast, state-of-the-art, highly secure algorithm
• Amenable to efficient implementation in both hw and sw; requires no special instructions to obtain good performance on any computing platform
• Triple-DES, still highly secure and supported by NIST, is expected to be common for the foreseeable future.

Reminder:
World’s best cipher!
One-time pad

For each character:

\[
\begin{array}{cccccccccc}
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1
\end{array}
\]

\[
\oplus
\]

\[
\begin{array}{cccccccccc}
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
\]

One-time pad (cont.)

- Symmetric
- Pad is selected at random
- Pad is as long as plaintext
- Perfectly secure, but...
- One time only:
  so sending the pad is just as hard as sending the msg