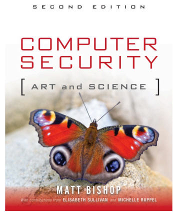
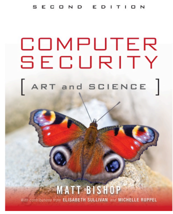


The Encryption Standards

Appendix F

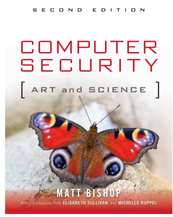
Outline

- Data Encryption Standard
  - Algorithm
- Advanced Encryption Standard
  - Background mathematics
  - Algorithm



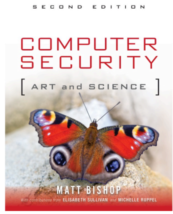
Data Encryption Standard (DES)

- Input: 64 bit blocks
- Key: 64 bits
    - 8 bits are immediately discarded, so it is effectively 56 bits
- Output: 64 bit blocks

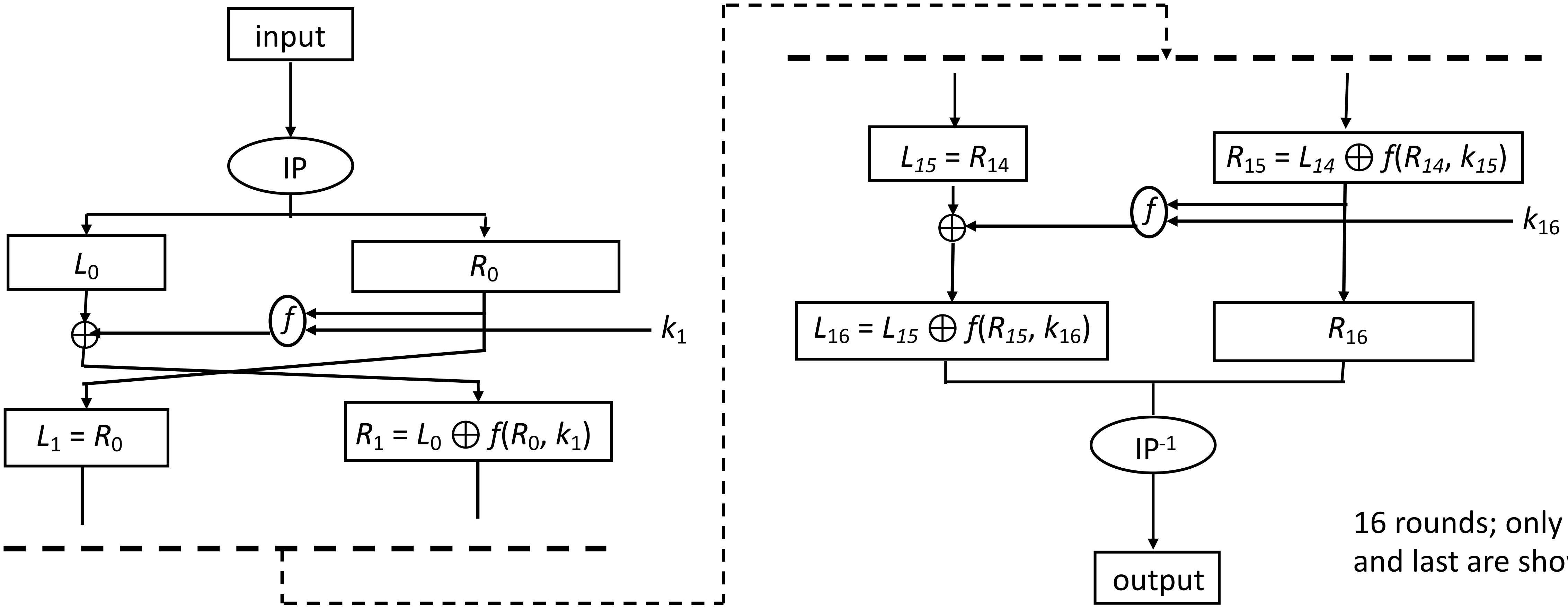


Main Algorithm

- Key permuted, split into 2 28-bit parts
  - Each part rotated left by 1 or 2 bits
  - Then the halves combined, permuted, and 48 bits output (*round key*)
- Input permuted, split into 2 32-bit parts
  - Right half, round key fed into function *f*
  - Result of this xor'ed with left half
  - This left half becomes right half, right half becomes left half, as input to next round (but in the last round, this does not occur)
- After 16 rounds, halves combined, then permuted and that is output
  - Permutation here is inverse of initial input permutation

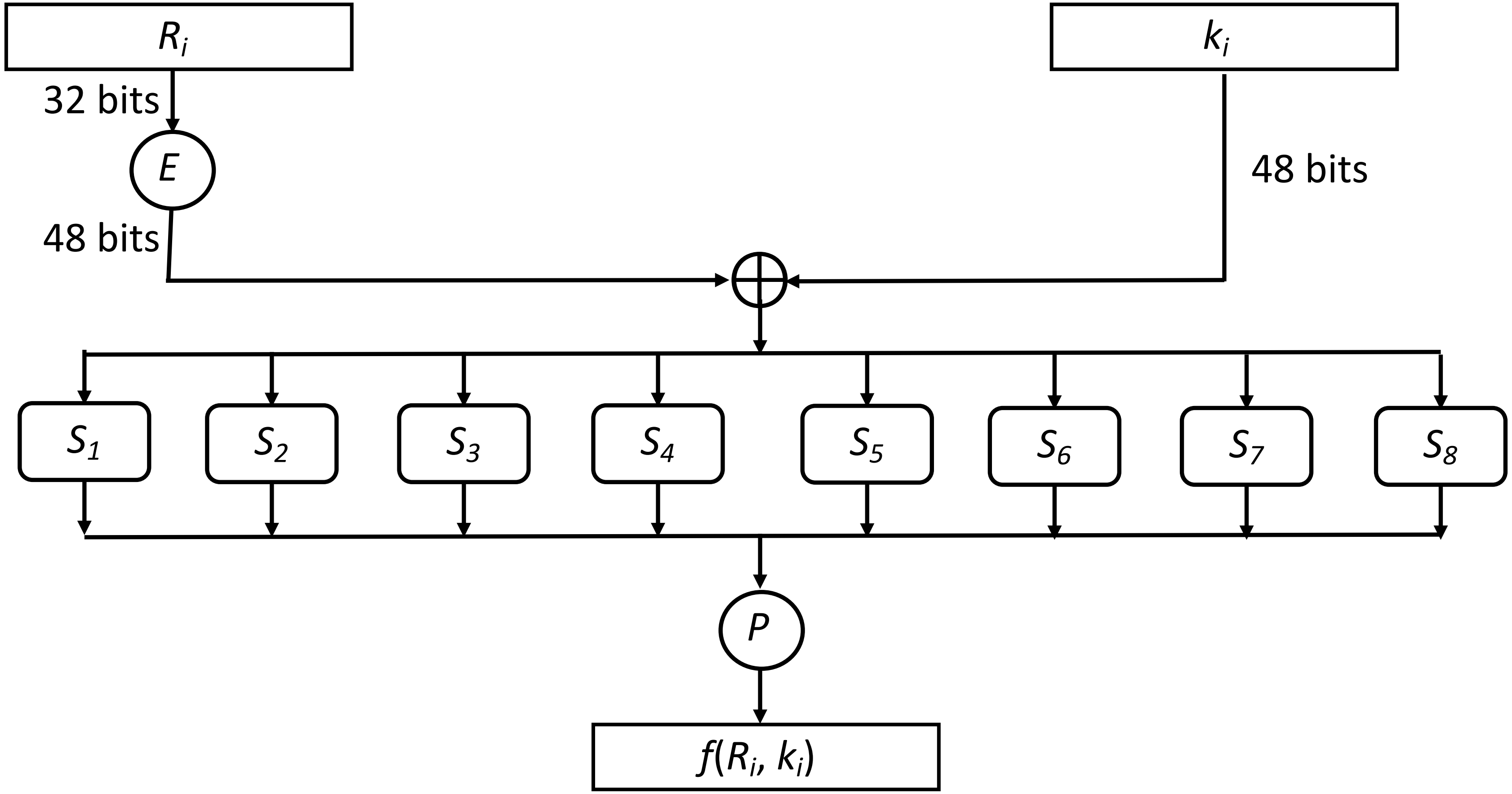


DES Algorithm: Rounds

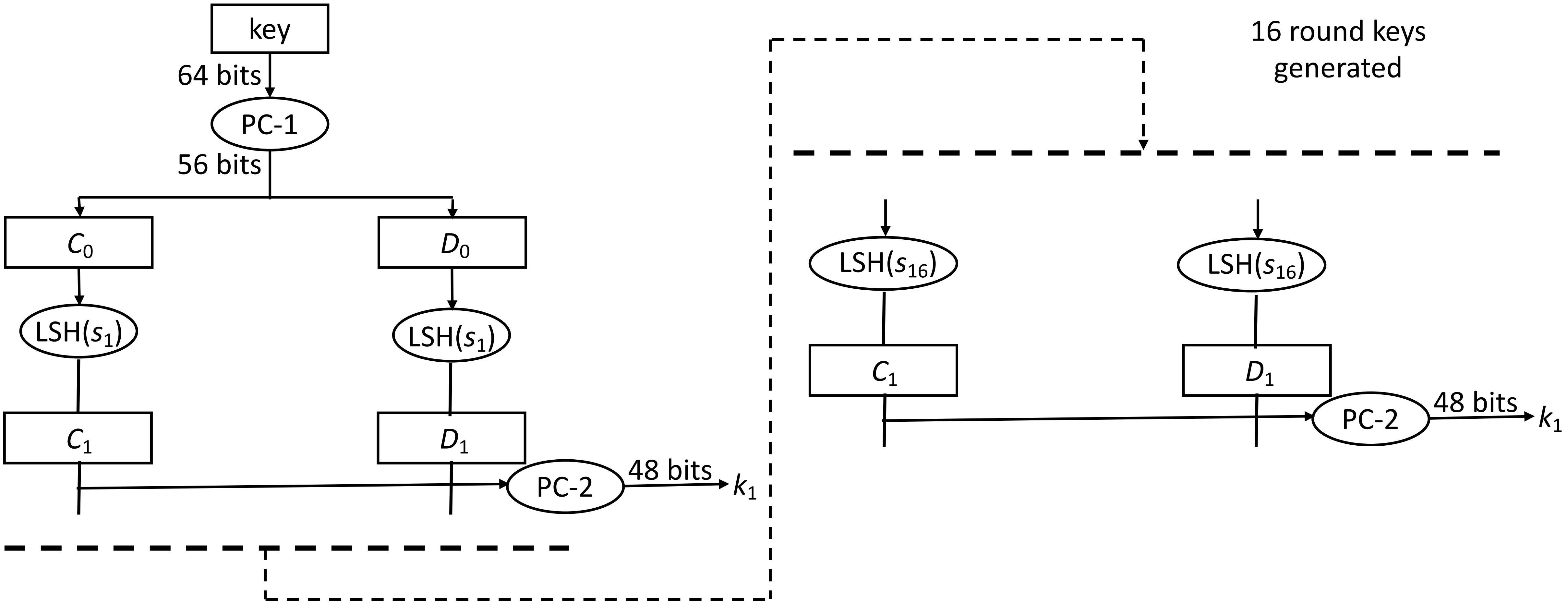


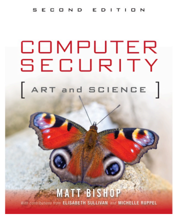
16 rounds; only first and last are shown

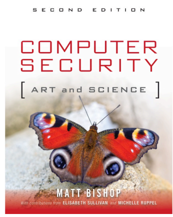
DES Algorithm: *f*



DES Algorithm: Round Key Generation





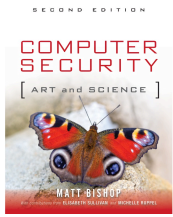


How to Read the Tables

- The *i*th element of the table, t_i, means that t_i is the bit of input that is output
- Example: first row of IP table is:

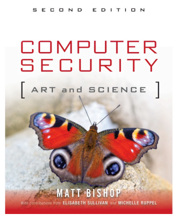
  58 50 42 34 26 18 10 2

  so the first bit out output is bit 58 of the input; the second bit of output is bit 50 of the input; and so forth
- LSH table: when generating the *i*th round key, the corresponding table entry *si* is the number of bits to rotate left (note: *rotate*, not shift)
- Example: s_i = 1 means rotate to the left 1 bit; s_i = 2 means rotate to the left 2 bits



Advanced Encryption Standard

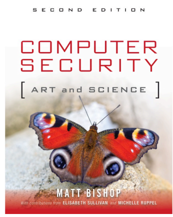
- All take input of 128 bits and produce outputs of 128 bits
  - AES-128: key length of 128 bits, 10 rounds
  - AES-192: key length of 192 bits, 12 rounds
  - AES-256: key length of 256 bits, 14 rounds
- In what follows:
  - *Nk* number of 32 bit words in the key
  - *Nb* number of 32 bit words in the block size
  - *Nr* number of rounds
  - w_i the *i*th set of 32 bits (4 bytes) of key schedule
  - Represent bytes as 2 hexadecimal digits or 8 binary digits



Background: Polynomials in *GF*(2^8)

- Manipulation of bytes treat them as polynomials in *GF*(2^8), each bit being a coefficient
  - Byte b5 (hex) is 10110101 (binary) and $x^7 + x^5 + x^4 + x^2 + 1$ (polynomial)
  - Arithmetic involving coefficients is done modulo 2
- Addition: same as exclusive or of two bytes:

```
   5b                        01011011
 ⊕ a4    as, in binary,    ⊕ 10101000
 ----                      ----------
   f3                        11110011
```



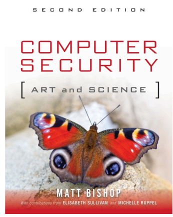
Background: Polynomials in $GF(2^8)$

- To multiply *a* and *b* ($a \bullet b$), convert them to polynomials, multiply them mod $x^8 + x^4 + x^3 + x + 1$
  - Note multiplication of coefficients is done mod 2
- Example: multiply bytes `57` (hex; `01010111` binary), `83` (hex; `10000011` binary)

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

$$= (x^8 + x^4 + x^3 + x + 1)(x^5 + x^3) + (x^7 + x^6 + 1)$$

So the result is `11000001` (binary) or `c1` (hex), so `57` • `83` = `c1`



AES: Input, State, Output

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

input bytes

→

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

state array

→

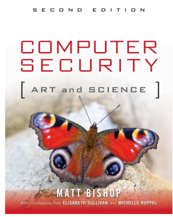
out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

output bytes

AES: Basic Encryption Transformations

Built up from 4 of these:

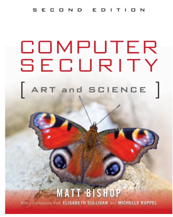
- `SubBytes`
- `ShiftRows`
- `MixColumns`
- `AddRoundKey`



AES: SubBytes

- A substitution table: takes 1 byte of input, produces 1 byte of output
  - First 4 bits give the row, next 4 the column
- Table constructed as follows:
  - Map byte 00 to itself, other bytes to their multiplicative inverse in $GF(2^8)$; call the result b, with bits $b_0b_1b_2b_3b_4b_5b_6b_7$
  - Let c_i be the ith bit of 01100011
  - Construct b’, with bits $b_0'b_1'b_2'b_3'b_4'b_5'b_6'b_7'$, where for i = 0, …, 7:

$$b_i' = b_i + b_{(i+4) \bmod 8} + b_{(i+5) \bmod 8} + b_{(i+6) \bmod 8} + b_{(i+7) \bmod 8} + c_i$$



AES: ShiftRows

- Rotate (shift cyclically) to the left by the number of the row

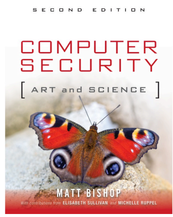
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

state array before

→

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

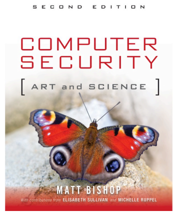
state array after



AES: MixColumns

Let c = 0, 1, 2, 3 and $s_{0,c}'$, $s_{1,c}'$, $s_{2,c}'$ and $s_{3,c}'$ the outputs of this

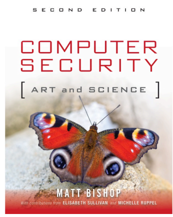
- $s_{0,c}' = (02 \bullet s_{0,c}) \oplus (03 \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$
- $s_{1,c}' = s_{0,c} \oplus (02 \bullet s_{1,c}) \oplus (03 \bullet s_{2,c}) \oplus s_{3,c}$
- $s_{2,c}' = s_{0,c} \oplus s_{1,c} \oplus (02 \bullet s_{2,c}) \oplus (03 \bullet s_{3,c})$
- $s_{3,c}' = (03 \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (02 \bullet s_{3,c})$



AES: AddRoundKey

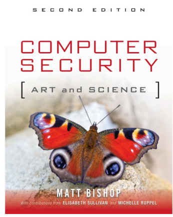
- Let r be the current round
- Remember w_i is ith set of 32 bits of key schedule
- Let c = 0, 1, 2, 3 and $s_{0,c}'$, $s_{1,c}'$, $s_{2,c}'$ and $s_{3,c}'$ the outputs of this

$$[s_{0,c}', s_{1,c}', s_{2,c}', s_{3,c}'] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{4r+c}]$$



AES: Encryption Algorithm

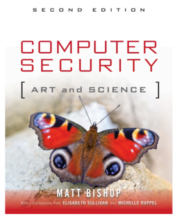
```
encrypt(byte in[4*Nb], byte out[4*NB], word w[Nb*(Nr+1)])
begin
     byte state[4,Nb]; state := in;
     AddRoundKey(state, w[0, Nb-1]);
     for round := 1 to Nr-1 do begin
          SubBytes(state);
          ShiftRows(state);
          MixColumns(state);
          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1]);
     end
     SubBytes(state);
     ShiftRows(state);
     AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]);
     out := state;
end
```



AES: Basic Encryption Transformations

Built up from 4 of these:

- `SubBytes`
- `ShiftRows`
- `MixColumns`
- `AddRoundKey`



AES: SubBytes

- A substitution table: takes 1 byte of input, produces 1 byte of output
  - First 4 bits give the row, next 4 the column
- Table constructed as follows:
  - Map byte 00 to itself, other bytes to their multiplicative inverse in $GF(2^8)$; call the result b, with bits $b_0b_1b_2b_3b_4b_5b_6b_7$
  - Let c_i be the ith bit of 01100011
  - Construct b', with bits $b_0'b_1'b_2'b_3'b_4'b_5'b_6'b_7'$, where for i = 0, …, 7:

$$b_i' = b_i + b_{(i+4) \bmod 8} + b_{(i+5) \bmod 8} + b_{(i+6) \bmod 8} + b_{(i+7) \bmod 8} + c_i$$

AES: ShiftRows

- Rotate (shift cyclically) to the left by the number of the row

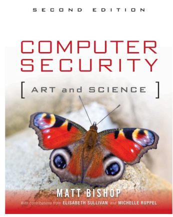
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

state array before

→

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

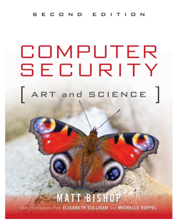
state array after



AES: MixColumns

Let c = 0, 1, 2, 3 and $s_{0,c}'$, $s_{1,c}'$, $s_{2,c}'$ and $s_{3,c}'$ the outputs of this

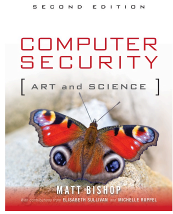
- $s_{0,c}' = (02 \bullet s_{0,c}) \oplus (03 \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$
- $s_{1,c}' = s_{0,c} \oplus (02 \bullet s_{1,c}) \oplus (03 \bullet s_{2,c}) \oplus s_{3,c}$
- $s_{2,c}' = s_{0,c} \oplus s_{1,c} \oplus (02 \bullet s_{2,c}) \oplus (03 \bullet s_{3,c})$
- $s_{3,c}' = (03 \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (02 \bullet s_{3,c})$



AES: AddRoundKey

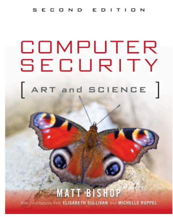
- Let r be the current round
- Remember w_i is ith set of 32 bits of key schedule
- Let c = 0, 1, 2, 3 and $s_{0,c}'$, $s_{1,c}'$, $s_{2,c}'$ and $s_{3,c}'$ the outputs of this

$$[s_{0,c}', s_{1,c}', s_{2,c}', s_{3,c}'] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{4r+c}]$$



AES: Encryption Algorithm

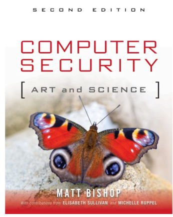
```
encrypt(byte in[4*Nb], byte out[4*NB], word w[Nb*(Nr+1)])
begin
      byte state[4,Nb];
      state := in;
      AddRoundKey(state, w[0, Nb-1]);
      for round := 1 to Nr-1 do begin
            SubBytes(state);
            ShiftRows(state);
            MixColumns(state);
            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1]);
      end
      SubBytes(state);
      ShiftRows(state);
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]);
      out := state;
end
```



AES: Basic Decryption Transformations

Built up from 4 of these:

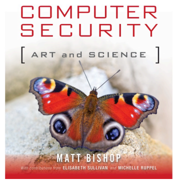
- `InvSubBytes` is the inverse transformation of `SubBytes`
- `InvShiftRows` is the inverse of `ShiftRows` (cyclic shift to the right by the number of the row)
- `InvMixColumns`
- `AddRoundKey`



AES: InvMixColumns

Let c = 0, 1, 2, 3 and $s_{0,c}'$, $s_{1,c}'$, $s_{2,c}'$ and $s_{3,c}'$ the outputs of this

- $s_{0,c}' = (\texttt{0e} \bullet s_{0,c}) \oplus (\texttt{0b} \bullet s_{1,c}) \oplus (\texttt{0d} \bullet s_{2,c}) \oplus (\texttt{09} \bullet s_{3,c})$
- $s_{1,c}' = (\texttt{09} \bullet s_{0,c}) \oplus (\texttt{0e} \bullet s_{1,c}) \oplus (\texttt{0b} \bullet s_{2,c}) \oplus (\texttt{0d} \bullet s_{3,c})$
- $s_{2,c}' = (\texttt{0d} \bullet s_{0,c}) \oplus (\texttt{09} \bullet s_{1,c}) \oplus (\texttt{0e} \bullet s_{2,c}) \oplus (\texttt{0b} \bullet s_{3,c})$
- $s_{3,c}' = (\texttt{0b} \bullet s_{0,c}) \oplus (\texttt{0d} \bullet s_{1,c}) \oplus (\texttt{09} \bullet s_{2,c}) \oplus (\texttt{0e} \bullet s_{3,c})$



AES: Decryption Algorithm

```
decrypt(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
      byte state[4,Nb];
      state := in;
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]);
      for round := 1 to Nr-1 do begin
            InvShiftRows(state);
            InvSubBytes(state);
            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1]);
            InvMixColumns(state);
      end
      InvShiftRows(state);
      InvSubBytes(state);
      AddRoundKey(state, w[0, Nb-1]);
      out := state;
end
```

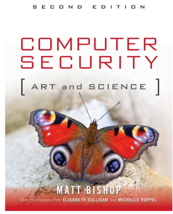
AES: Basic Round Key Generation. Transformations

Two transformations:

- SubWord takes 4 bytes as input, applies SubByte to each byte individually, and outputs the result
- RotWord takes a 4-byte word as input, rotates it right by 1 byte, and outputs the result

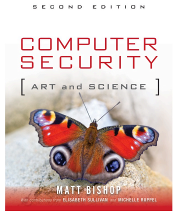
And a round constant word array:

- For *i*-th round, `Rcon[`*i*`]` = [x^{i-1}, 00, 00, 00] where x = 02 and x^i uses multiplication as described before
  - Example: `Rcon[1] = 01000000; Rcon[2] = 02000000;`
    `Rcon[3] = 04000000; Rcon[4] = 08000000;`
    `Rcon[5] = 10000000; ...`



AES: Round Key Generation Algorithm

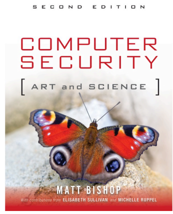
```
roundkeys(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
      word temp;
      for i:= 0 to Nk-1 do
            w[i] = word(key[4*i], key[4*i+1],
                                    key[4*i+2], key[4*i+3]);
      for i := Nk to (Nr+1)*Nb-1 do begin
            temp := w[i-1];
            if (i mod Nk = 0)
                  temp = SubWord(RotWord(temp)) xor Rcon[i/Nk];
            else if (Nk > 6 and i mod Nk = 4)
                  temp = SubWord(temp);
            w[i] = w[i-Nk] xor temp;
      end
end
```



AES: Equivalent Inverse Cipher Implementation

- Add these to the end of the Round Key Generation algorithm:

```
for i = 0 to (Nr+1)*Nb-1 do
     dw[i] = w[i];
for round = 1 to Nr-1 do
     InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
```



AES: Alternate Decryption Algorithm

```
equivdecrypt(byte in[4*Nb], byte out[4*NB], word dw[Nb*(Nr+1)])
begin
      byte state[4,Nb];
      state := in;
      AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1]);
      for round := Nr-1 downto Nr-1 do begin
            InvSubBytes(state);
            InvShiftRows(state);
            InvMixColumns(state);
            AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1]);
      end
      InvSubBytes(state);
      InvShiftRows(state);
      AddRoundKey(state, dw[0b, Nb-1]);
      out := state;
end
```