Secret Key Systems (block encoding)

Encrypting a small block of text (say 64 bits)

General considerations for cipher design:

Secret Key Systems (block encoding)

Encrypting a small block of text (say 64 bits)

General considerations for cipher design:

- Encrypted data should look random.

  As though someone flipped a fair coin 64 times and heads means 1 and tails 0.
  Any change in one bit of output corresponds to a huge change in the input (bits are uncorrelated).
  Should be about as many 1's as 0's usually.

Secret Key Systems (block encoding)

Encrypting a small block of text (say 64 bits)

General considerations for cipher design:

- Encrypted data should look random.

  As though someone flipped a fair coin 64 times and heads means 1 and tails 0.

  Any change in one bit of output corresponds to a huge change in the input (bits are uncorrelated).

  Should be about as many 1's as 0's usually.

- Try to spread the influence of each input bit to all output bits

  Any change in one input bit should have 50% chance of changing any of the output bits (hence many rounds).

Secret Key Systems (block encoding)

Encrypting a small block of text (say 64 bits)

General considerations for cipher design:

- Encrypted data should look random.
  As though someone flipped a fair coin 64 times and heads means 1 and tails 0.
  Any change in one bit of output corresponds to a huge change in the input (bits are uncorrelated).
  Should be about as many 1's as 0's usually.
- Try to spread the influence of each input bit to all output bits
  Any change in one input bit should have 50% chance of changing any of the output bits (hence many rounds).
- Operations should be invertible – hence *xor* and table lookup.
  Use of one key for both encryption and decryption.

Secret Key Systems (block encoding)

Encrypting a small block of text (say 64 bits)

General considerations for cipher design:

- Encrypted data should look random.
  As though someone flipped a fair coin 64 times and heads means 1 and tails 0.
  Any change in one bit of output corresponds to a huge change in the input (bits are uncorrelated).
  Should be about as many 1's as 0's usually.
- Try to spread the influence of each input bit to all output bits
  Any change in one input bit should have 50% chance of changing any of the output bits (hence many rounds).
- Operations should be invertible – hence *xor* and table lookup.
  Use of one key for both encryption and decryption.
- Attacks may be mitigated if they rely on operations that are not efficiently implemented in hardware yet allow normal operation to complete efficiently, even in software (permute).

Secret Key Systems (block encoding)

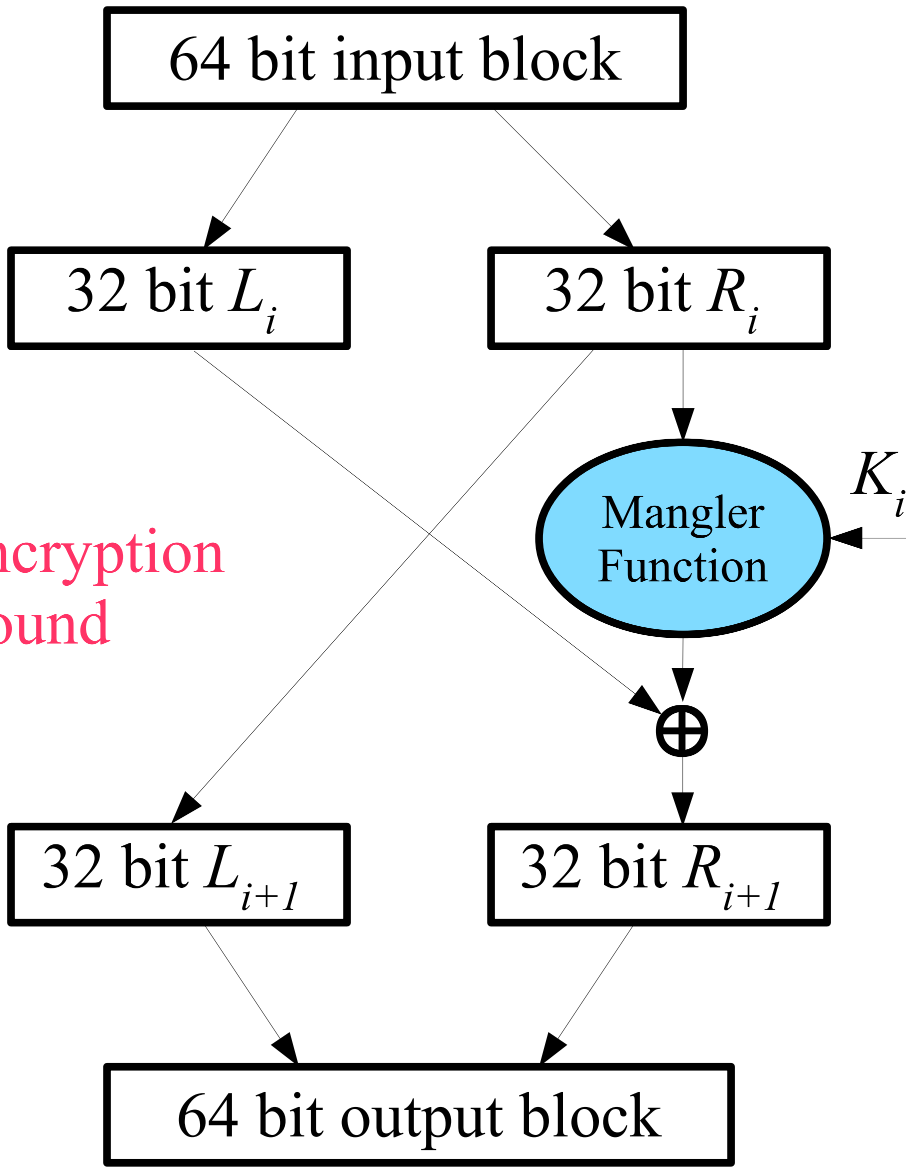
Encrypting a small block of text (say 64 bits)

What is considered a successful attack?

- Suppose some plaintext is known (a crib) and it is desired to find the key.
- If the cipher were generating truly random output, an attack on a key should take 2^{55} tries, on the average, for 56 bit keys.
- If someone can find a way to guarantee finding a key in 2^{50} tries even, then the cipher is considered broken.

Secret Key Systems - DES

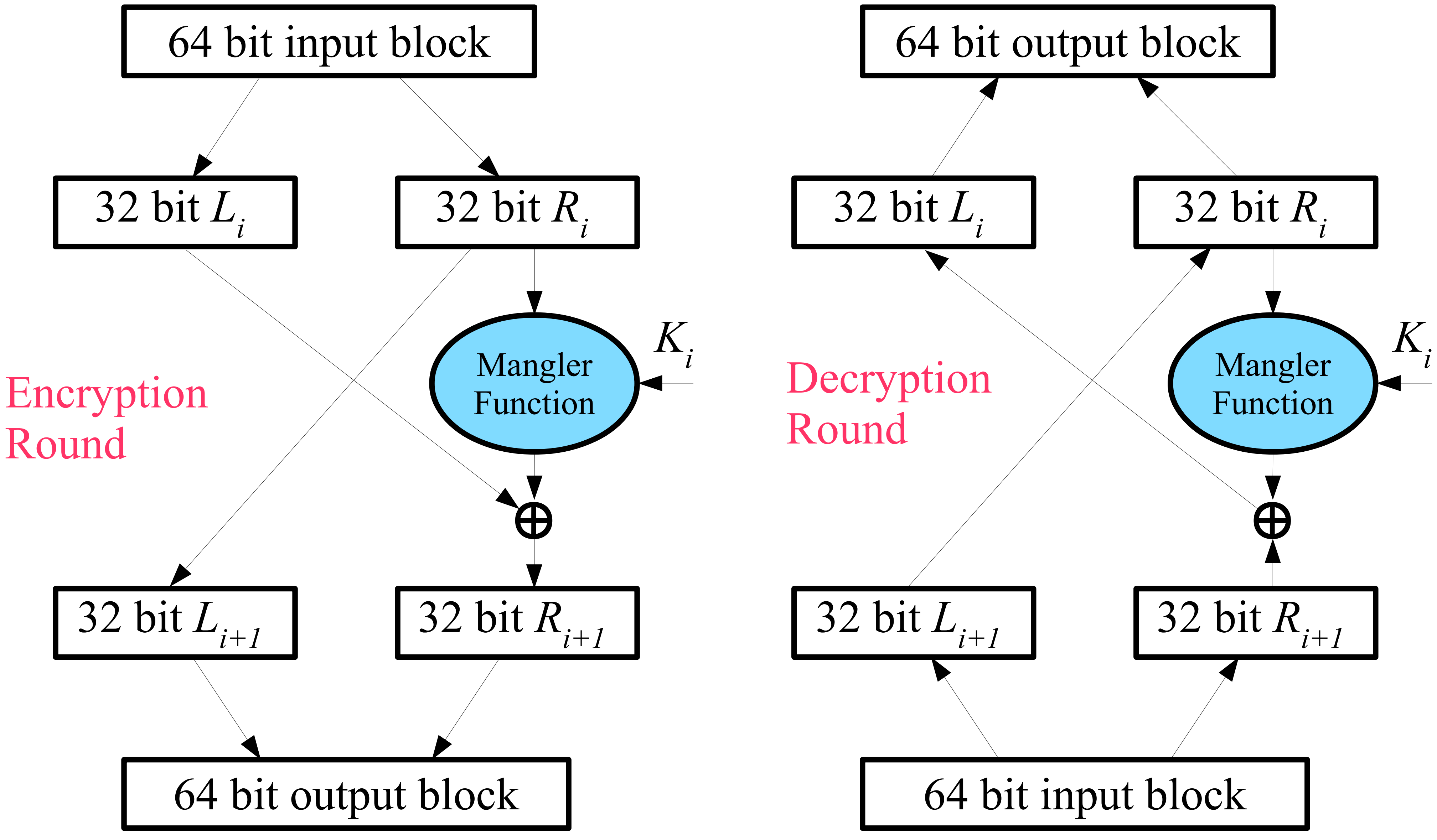
IBM/NSA 1977 - 64 bit blocks, 56 bit key, 8 bits parity



Encryption Round

Secret Key Systems - DES

IBM/NSA 1977 - 64 bit blocks, 56 bit key, 8 bits parity



Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:								
	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

57 49 41 33 25 17 9 1 58 50 42 34 26 18 10 2 59 51 43 35 27 19 11 3 60 52 44 36

D_0:

63 55 47 39 31 23 15 7 62 54 46 38 30 22 14 6 61 53 45 37 29 21 13 5 28 20 12 4

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

57 49 41 33 25 17 9 1 58 50 42 34 26 18 10 2 59 51 43 35 27 19 11 3 60 52 44 36

D_0:

63 55 47 39 31 23 15 7 62 54 46 38 30 22 14 6 61 53 45 37 29 21 13 5 28 20 12 4

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

57 49 41 33 25 17 9 1 58 50 42 34 26 18 10 2 59 51 43 35 27 19 11 3 60 52 44 36

D_0:

63 55 47 39 31 23 15 7 62 54 46 38 30 22 14 6 61 53 45 37 29 21 13 5 28 20 12 4

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

Permutations:

Left half C_i: (9,18,22,25 are missing)

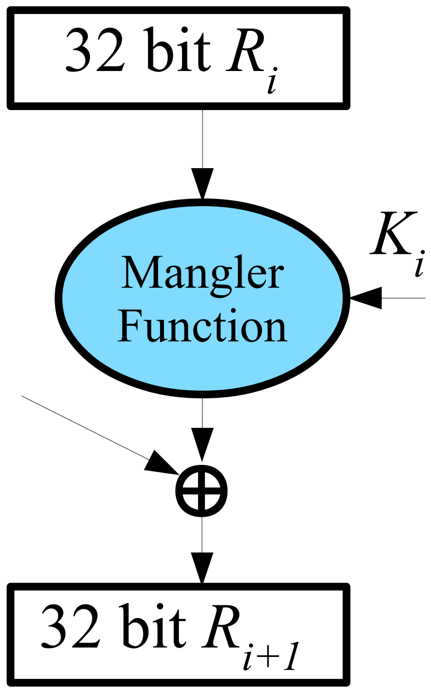
14 17 11 24 1 5 3 28 15 6 21 10 23 19 12 4 26 8 16 7 27 20 13 2

Right half D_i: (35,38,43,54 are missing)

41 52 31 37 47 55 30 40 51 45 33 48 44 49 39 56 34 53 46 42 50 36 29 32

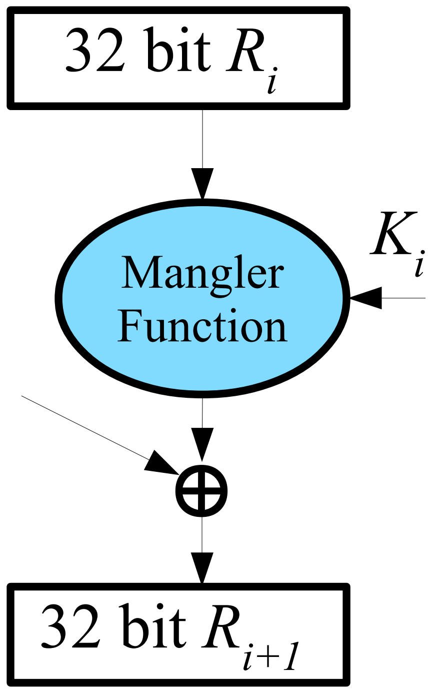
Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

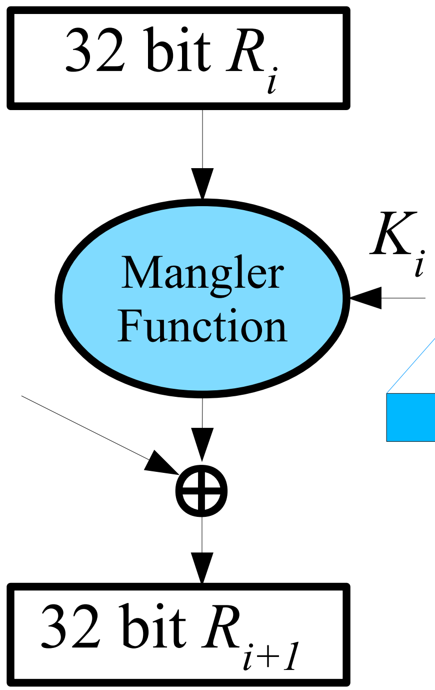
The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



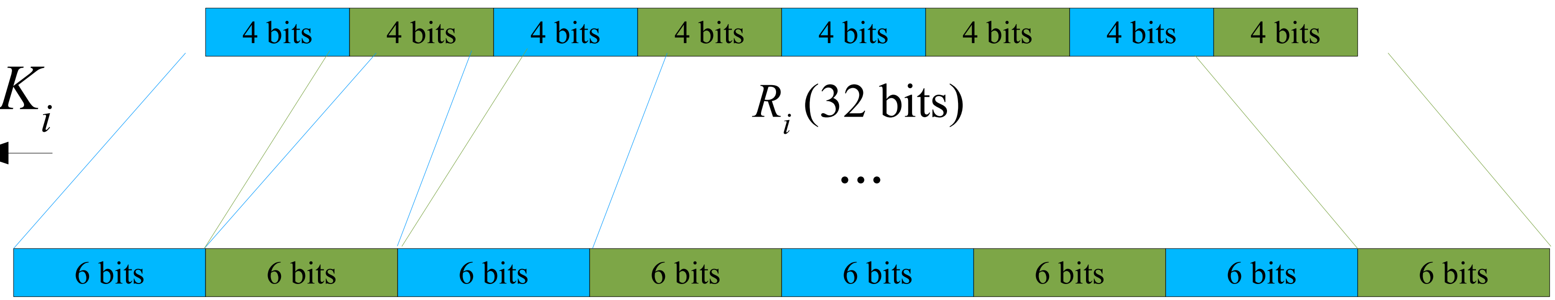
1. Expansion of input bits:

Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits

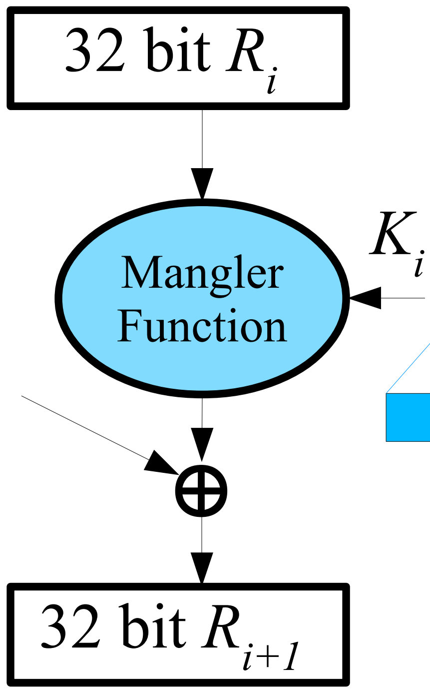


1. Expansion of input bits:

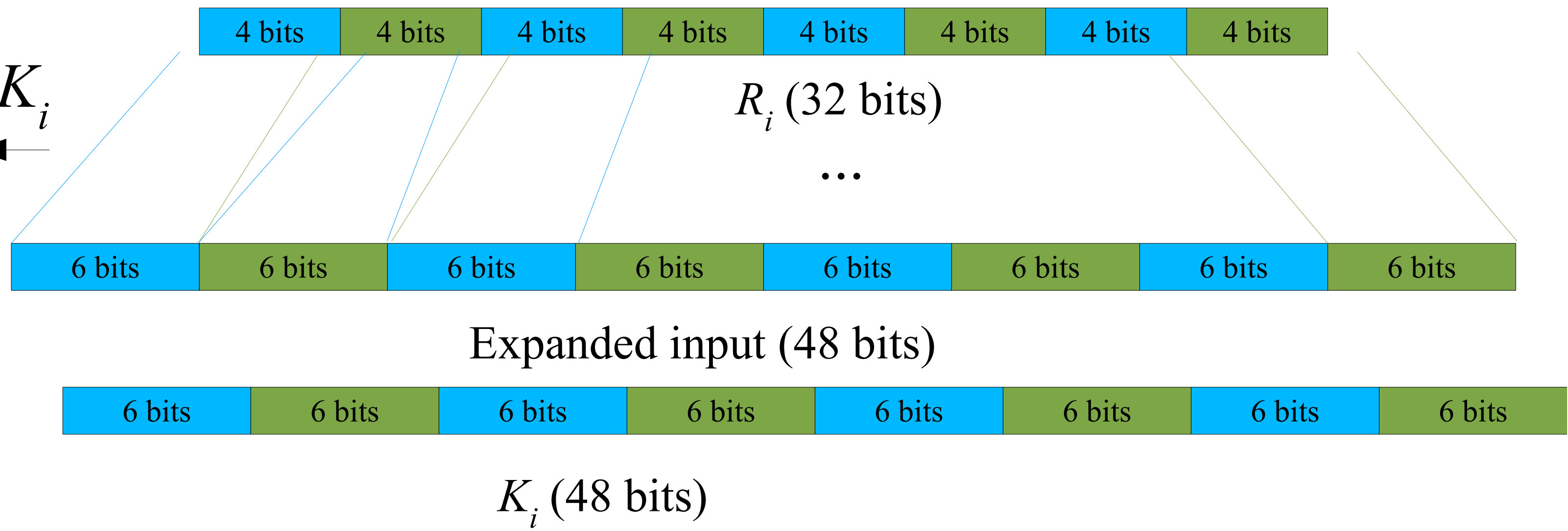


Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits

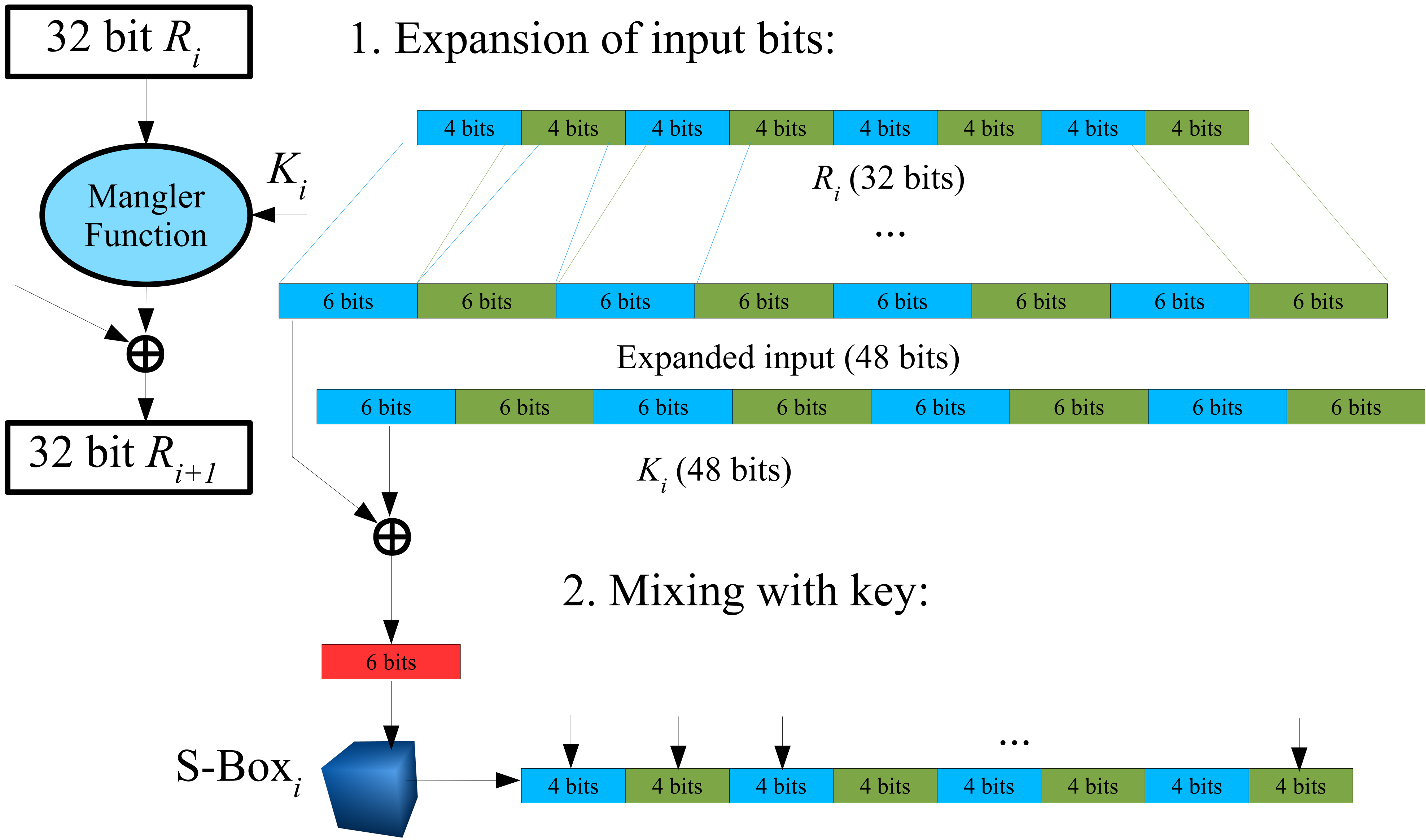


1. Expansion of input bits:



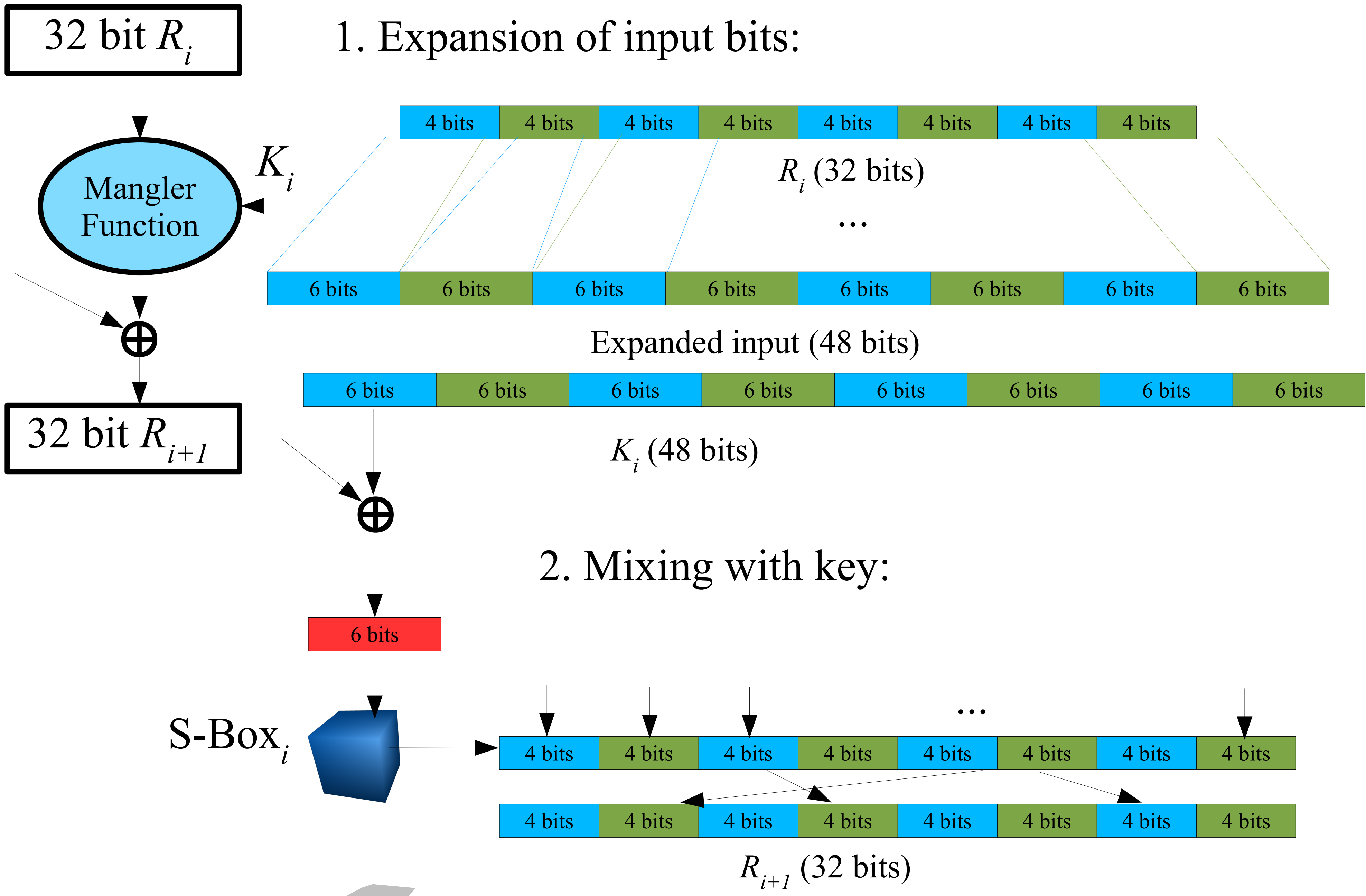
Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

The **S-Box**: maps 6 bit blocks to 4 bit sections

S-Box$_1$ (first 6 bits):

Input bits 2,3,4,5

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

Input bits 1 and 6

Final permutation:

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

Secret Key Systems - DES

Weak and semi-weak keys:

If key is such that C_0 *or* D_0 are:

1) alternating '`F`' and '`E`': `0xFEFEFE...`
2) `0xE0E0E0E0F1F1F1F1`, `0xF1F1F1F1E0E0E0E0`
3) alternating `1`s and `0`s: `0x010101...`

weak: E(E(M)) = M

Secret Key Systems - DES

Weak and semi-weak keys:

If key is such that C_0 *or* D_0 are:

1) alternating '`F`' and '`E`': `0xFEFEFE...`
2) `0xE0E0E0E0F1F1F1F1`, `0xF1F1F1F1E0E0E0E0`
3) alternating `1`s and `0`s: `0x010101...`

weak: E(E(M)) = M

`0x011F011F010E010E` and `0x1F011F010E010E01`

`0x01E001E001F101F1` and `0xE001E001F101F101`

`0X01FE01FE01FE01FE` and `0xFE01FE01FE01FE01`

semi-weak: $E_{K1}(E_{K2}(M)) = M$ (six total pairs)

Secret Key Systems - DES

Weak and semi-weak keys:

If key is such that C_0 *or* D_0 are:

1) alternating '`F`' and '`E`': `0xFEFEFE...`
2) `0xE0E0E0E0F1F1F1F1`, `0xF1F1F1F1E0E0E0E0`
3) alternating `1`s and `0`s: `0x010101...`

weak: E(E(M)) = M

`0x011F011F010E010E` and `0x1F011F010E010E01`

`0x01E001E001F101F1` and `0xE001E001F101F101`

`0X01FE01FE01FE01FE` and `0xFE01FE01FE01FE01`

semi-weak: $E_{K1}(E_{K2}(M)) = M$ (six total pairs)

Discussion:

1. Designed to be resistant to differential attacks – this type of attack was kept secret by NSA/IBM because other current cryptosystems were vulnerable
2. Changing S-Boxes has resulted in provably weaker system

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$m \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow \overset{K_2}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow c$$

encryption

$$c \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow \overset{K_2}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow m$$

decryption

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$\begin{array}{ccccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ m & \rightarrow & \mathrm{E} & \rightarrow & \mathrm{D} & \rightarrow & \mathrm{E} & \rightarrow & c \end{array}$$

encryption

$$\begin{array}{ccccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ c & \rightarrow & \mathrm{D} & \rightarrow & \mathrm{E} & \rightarrow & \mathrm{D} & \rightarrow & m \end{array}$$

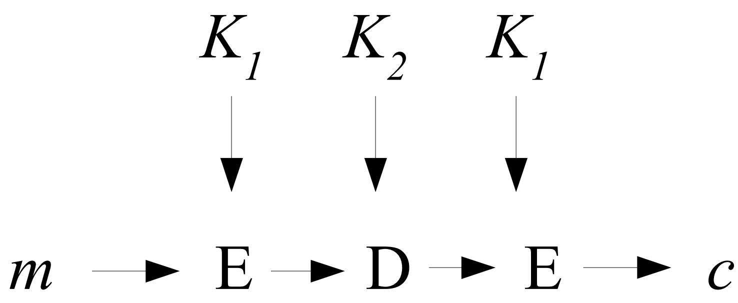
decryption

Why not 2DES

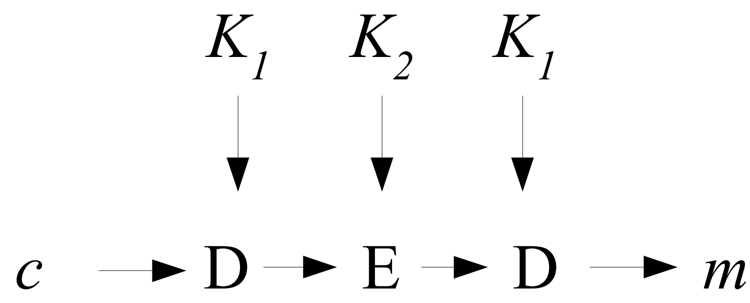
1. Double encryption with the same key still requires searching 2^{56} keys

Secret Key Systems - 3DES

Two keys K_1 and K_2:



encryption



decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known:

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ m \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & c \end{tikzcd}$$

encryption

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ c \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & m \end{tikzcd}$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some $<m,c>$ pairs are known:

2^{56}

m: K_1	E(K_1,m)
101010	1000011
...	...
100010	0101111
001011	0001101

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$\begin{array}{ccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ m & \rightarrow & E & \rightarrow & D & \rightarrow & E & \rightarrow & c \end{array}$$

encryption

$$\begin{array}{ccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ c & \rightarrow & D & \rightarrow & E & \rightarrow & D & \rightarrow & m \end{array}$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known:

m:

	K_1	$E(K_{1,}m)$
2^{56}	101010	1000011
	...	...
	100010	0101111
	001011	0001101

c:

	K_2	$D(K_{2,}c)$
2^{56}	101110	0001101
	...	...
	001110	1000011
	001011	0001101

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
m \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & c
\end{tikzcd}
$$

encryption

$$
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
c \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & m
\end{tikzcd}
$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known:

m:

K_1	$\mathrm{E}(K_1, m)$
101010	1000011
...	...
100010	0101111
001011	0001101

2^{56}

c:

K_2	$\mathrm{D}(K_2, c)$
101110	0001101
...	...
001110	1000011
001011	0001101

2^{56}

Secret Key Systems - 3DES

Two keys K_1 and K_2:

$$m \xrightarrow{} \underset{}{\overset{K_1}{\downarrow}}\!\mathrm{E} \to \overset{K_2}{\downarrow}\!\mathrm{D} \to \overset{K_1}{\downarrow}\!\mathrm{E} \to c$$

encryption

$$c \to \overset{K_1}{\downarrow}\!\mathrm{D} \to \overset{K_2}{\downarrow}\!\mathrm{E} \to \overset{K_1}{\downarrow}\!\mathrm{D} \to m$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some $<m,c>$ pairs are known:

m: (2^{56} rows)

K_1	$E(K_1,m)$
101010	1000011
...	...
100010	0101111
001011	0001101

c: (2^{56} rows)

K_2	$D(K_2,c)$
101110	0001101
...	...
001110	1000011
001011	0001101

Test matches on other $<m,c>$ pairs

$m \to E(K_1,m)$

$D(K_2,c) \leftarrow c$

Secret Key Systems - 3DES

Why not 2DES

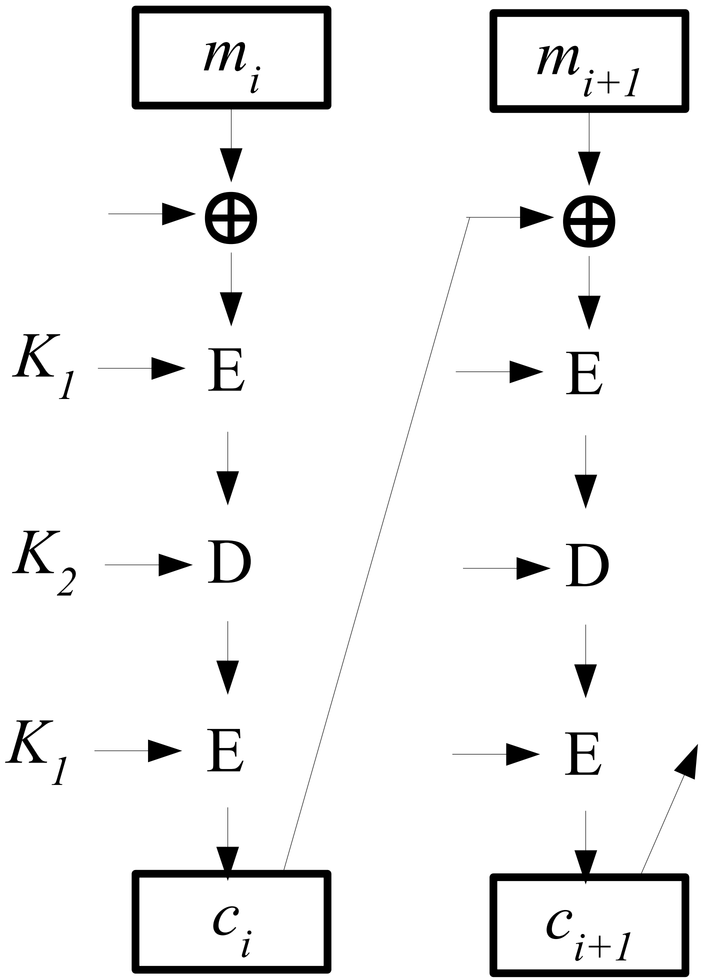
3. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known: How many <*m*,*c*> pairs do you need?

   2^{64} possible blocks
   2^{56} table entries
   each block has probability $2^{-8} = 1/256$ of showing in a table
   probability a block is in both tables is 2^{-16}
   average number of matches is $2^{-16} * 2^{64} = 2^{48}$
   average number of matches for two <*m*,*c*> pairs is about 2^{32}
   for three <*m*,*c*> pairs is about 2^{16}
   for four <*m*,*c*> pairs is about 2^{0}

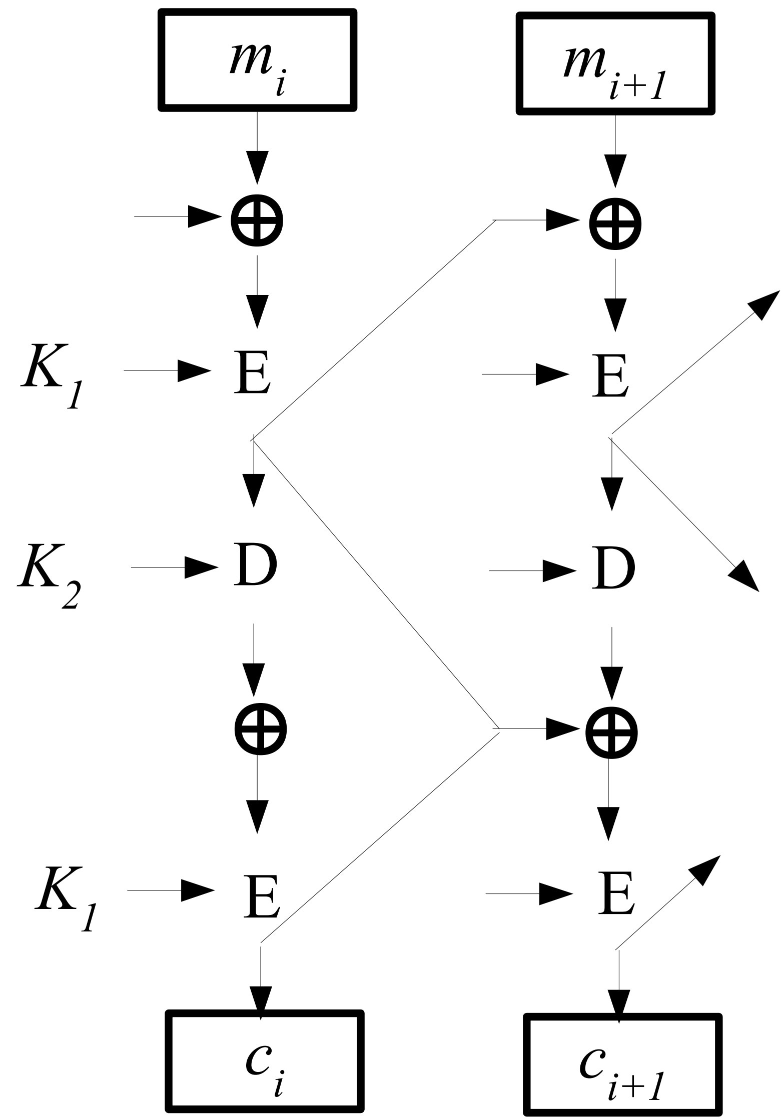
4. Triple encryption with two different keys
   - 112 bits of key is considered enough
   - No added security with three keys, encrypting three times

Secret Key Systems - 3DES

CBC with 3DES:



outside



inside

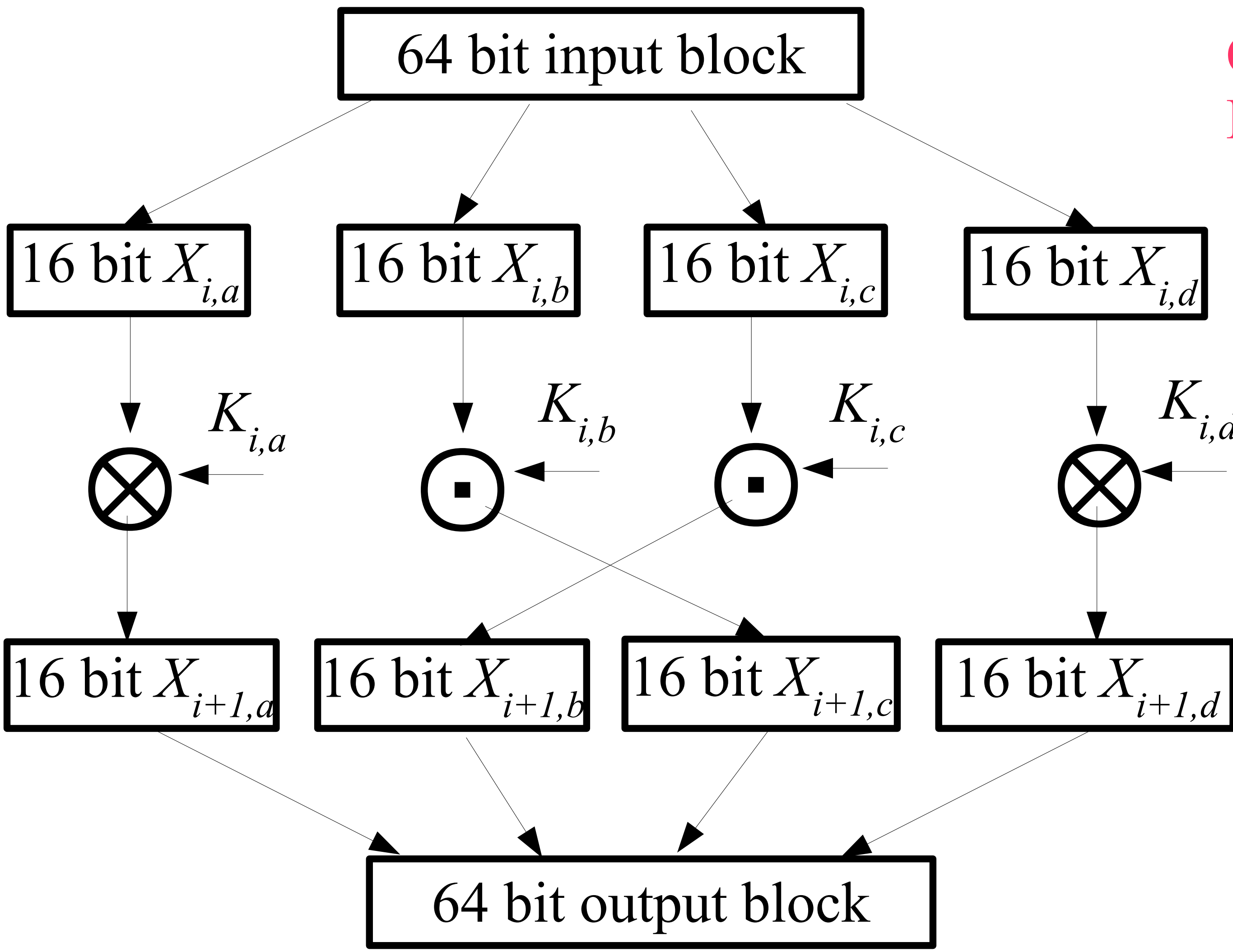
Secret Key Systems - 3DES

CBC with 3DES:

1. On the outside – same attack as with CBC – change a block with side effect of garbling another
2. On the inside – attempt at changing a block results in all blocks garbled to the end of the message.
3. On the inside – use three times as much hardware to pipeline encryptions resulting in DES speeds.
4. On the outside – EDE simply is a drop-in replacement for what might have been there before.

Secret Key Systems - IDEA

ETH Zuria 1991 - 64 bit blocks, 128 bit key, 8.5 rounds



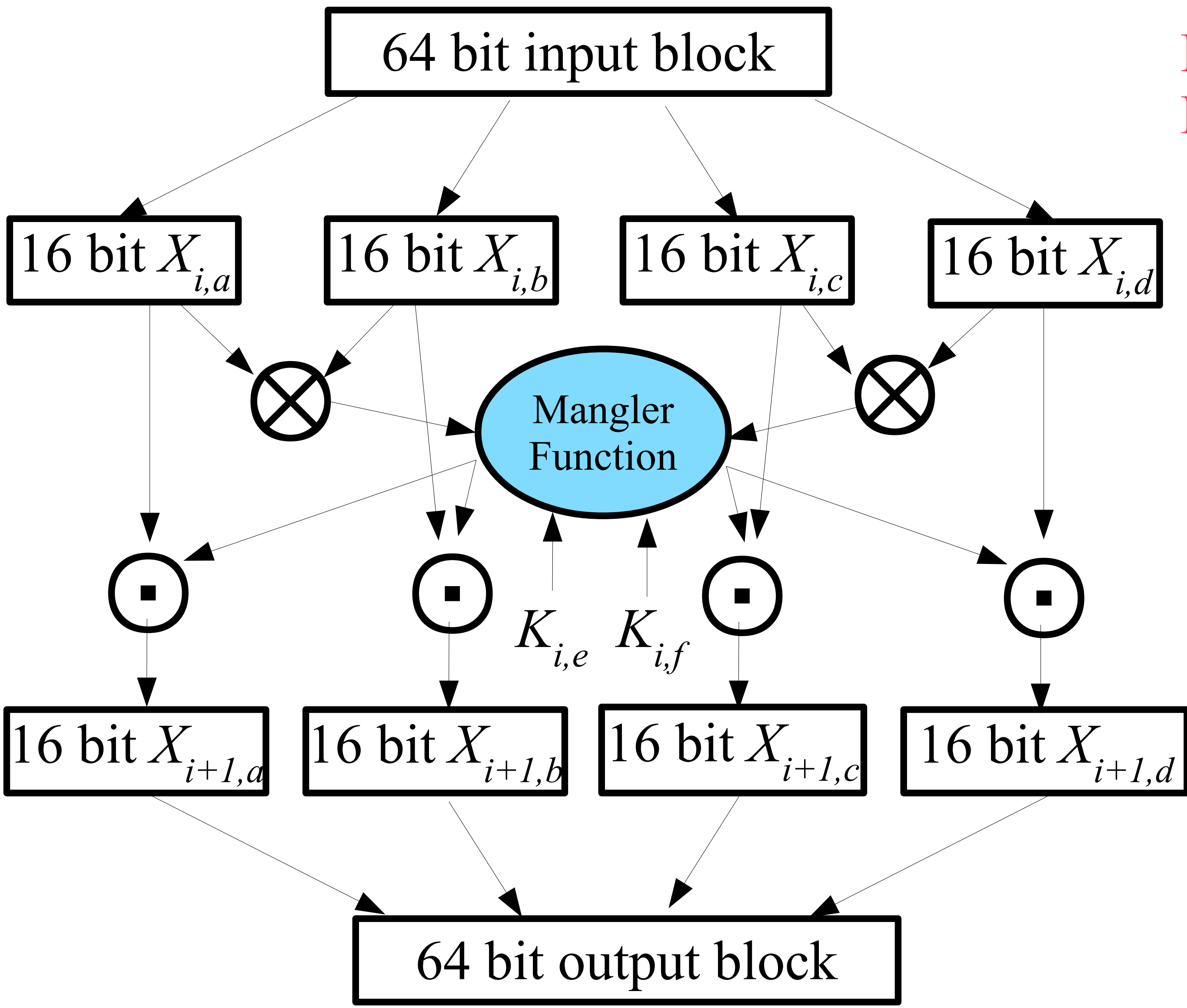
Odd Encryption Round

$\otimes$ Multiplication mod $2^{16}+1$. 0 is treated as 2^{16} (invertible)

$\odot$ Addition but throw away carries

Secret Key Systems - IDEA

ETH Zuria 1991 - 64 bit blocks, 128 bit key, 8.5 rounds

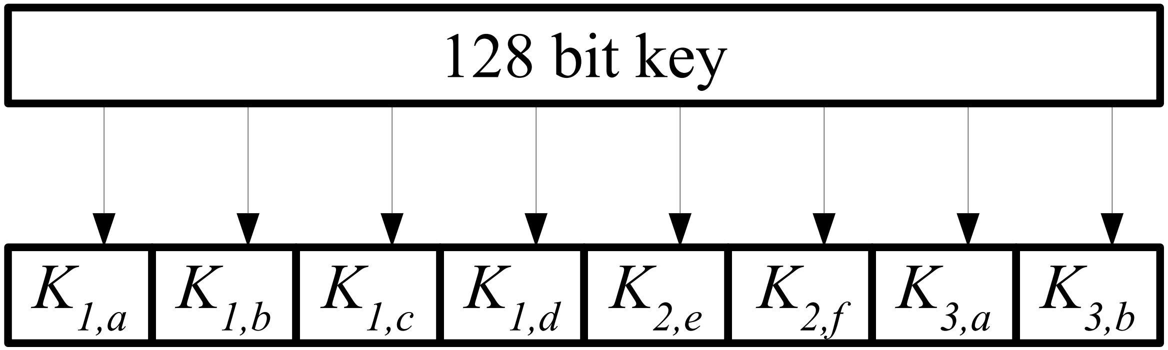


Even Encryption Round

$$Y_{in} = X_{i,a} \otimes X_{i,b}$$
$$Z_{in} = X_{i,c} \otimes X_{i,d}$$
$$Y_{out} = ((K_{i,e} \otimes Y_{in}) \odot Z_{in} \otimes K_{i,f}$$
$$Z_{out} = (K_{i,e} \otimes Y_{in}) \odot Y_{out}$$
$$X_{i+1,a} = X_{i,a} \odot Y_{out}$$
$$X_{i+1,b} = X_{i,b} \odot Y_{out}$$
$$X_{i+1,c} = X_{i,c} \odot Z_{out}$$
$$X_{i+1,d} = X_{i,d} \odot Z_{out}$$

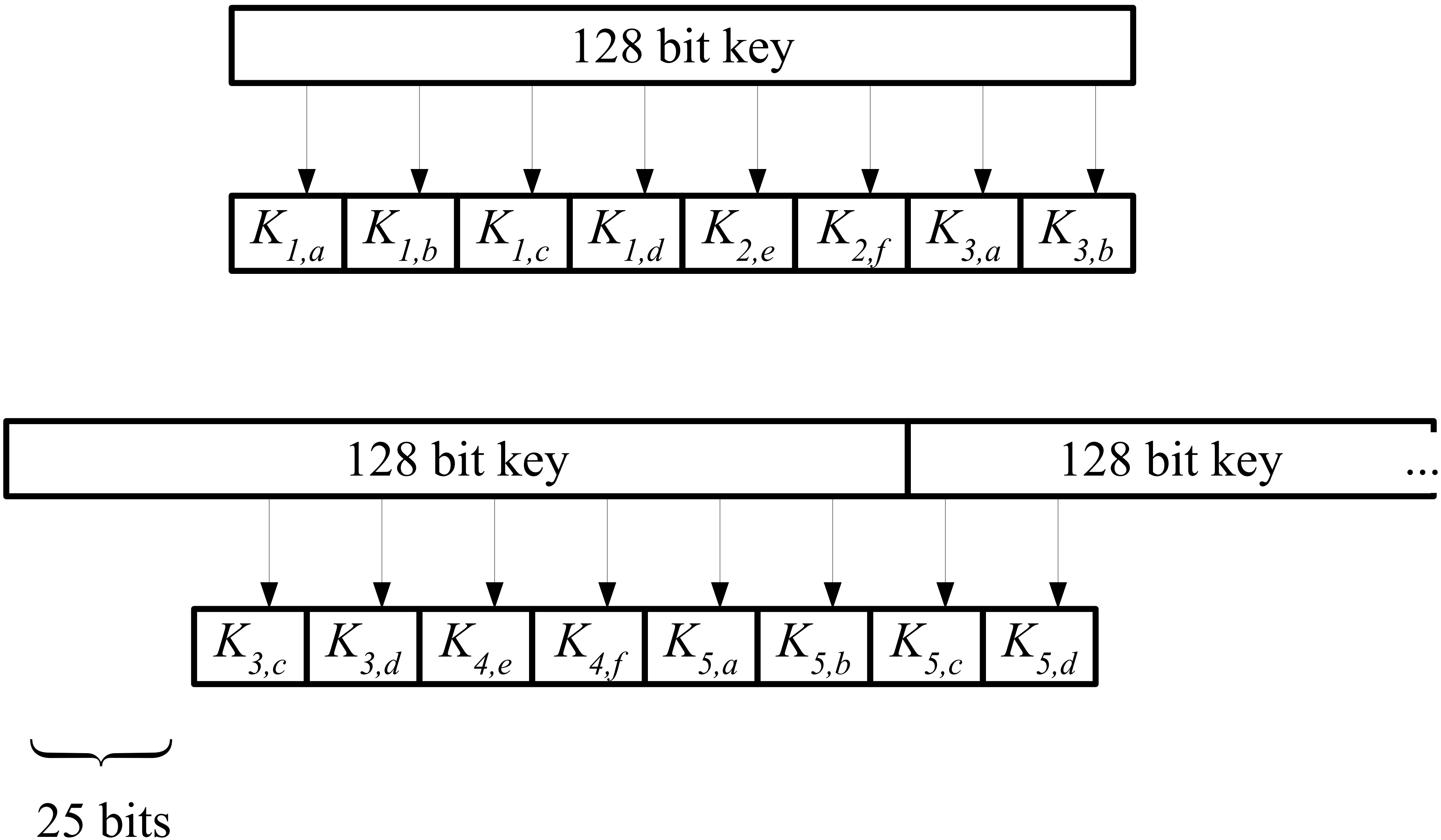
Secret Key Systems - IDEA

Key generation   52 16 bit keys needed (2 per even round 4 per odd):



Secret Key Systems - IDEA

Key generation   52 16 bit keys needed (2 per even round 4 per odd):



Secret Key Systems - IDEA

IDEA has been proven to be successful against many cipher attack methods

It appears to have no algebraic or linear type of weakness
(linear: develop equations relating plaintext, ciphertext, key bits whose probability of holding over the space of all possible values of their variables is close to 1 or 0 – then use these equations to get key bits from known ciphertex-plaintext pairs)

Immune to differential attack under some circumstances

IDEA was 'broken' in 2011 using a meet-in-the-middle attack
IDEA was 'broken' in 2012 using a narrow-bicliques attack (variant of a meet-in-the-middle attack) reducing cryptographic strength by 2 bits.

Patents limited use until 2012 when all expired.

There exist some weak keys but they are rare

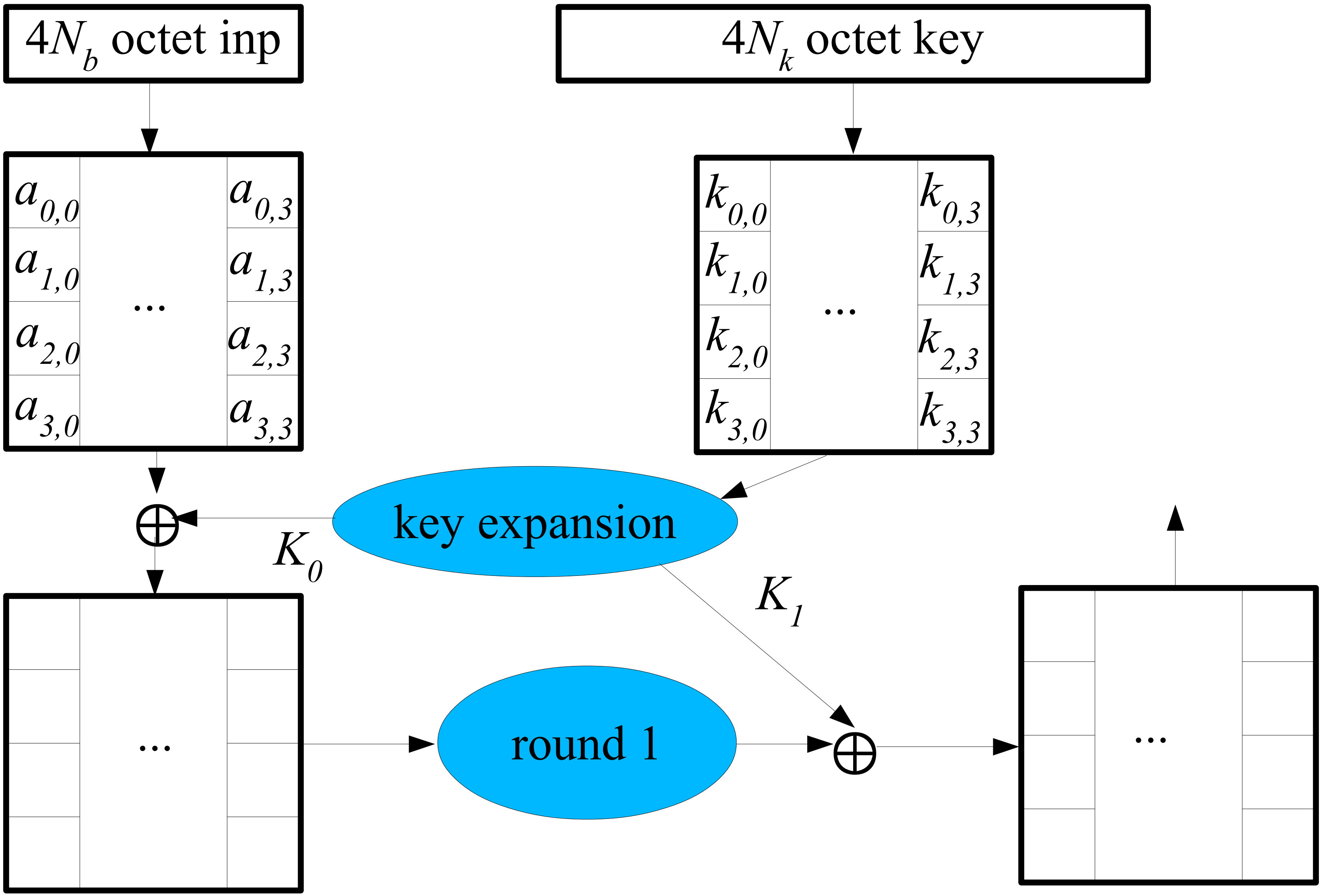
Secret Key Systems – For Fun

Secure Shell uses the following ciphers for encryption and authentication:

	Cipher	SSH1	SSH2
	DES	yes	no
	3DES	yes	yes
	IDEA	yes	no
Encryption:	Blowfish	yes	yes
	Twofish	no	yes
	Arcfour	no	yes
	AES		yes
Authentication:	RSA	yes	no
	DSA	no	yes

Secret Key Systems - AES

NIST (2001) parameterized key size (128 bits to 256 bits)



Secret Key Systems - AES

The State: An array of four rows and N_b columns – each element is a byte.

Initially: next block of $4N_b$ input bytes.

Execution: all operations are performed on the State.

Example: $N_b = 4$

$$\begin{array}{|c|c|c|c|}\hline in_0 & in_4 & in_8 & in_{12} \\ \hline in_1 & in_5 & in_9 & in_{13} \\ \hline in_2 & in_6 & in_{10} & in_{14} \\ \hline in_3 & in_7 & in_{11} & in_{15} \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

Secret Key Systems - AES

Addition: modulo 2 addition (xor) of polynomials of maximum degree 7

Examples:

$(x^6+x^4+x^2+x^1+1) + (x^7+x^1+1) = x^7+x^6+x^4+x^2$ (polynomial)

$01010111 \oplus 10000011 = 11010100$ (binary notation)

$0x57 \oplus 0x83 = D4$ (hexadecimal)

Secret Key Systems - AES

Multiplication of two degree 7 polynomials (bytes):

Just like ordinary multiplication except mod $m(x) = (x^8+x^4+x^3+x^1+1)$

Reason: for each byte there will be an inverse: $a \times a^{-1} = 1 \bmod m(x)$

Basis: $x \times b = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

Shift b left by 1, if result has a degree 8 bit, xor with $m(x)$

This operation is called **xtime(x) = (x << 1) ⊕ (((x >> 7) & 1) * 0x11b)**

Secret Key Systems - AES

Multiplication of two degree 7 polynomials (bytes):

Just like ordinary multiplication except mod $m(x) = (x^8+x^4+x^3+x^1+1)$

Reason: for each byte there will be an inverse: $a \times a^{-1} = 1 \bmod m(x)$

Basis: $x \times b = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

Shift b left by 1, if result has a degree 8 bit, xor with $m(x)$

This operation is called **xtime(x) = (x << 1) ⊕ (((x >> 7) & 1) * 0x11b)**

Example:

$\text{xtime}(x^7 + x^5 + x^3 + x^2 + x) = x^6 + x^2 + x + 1$ or

$101011100 \oplus 100011011 = 01000111$ or $\text{xtime}(\text{0xAE}) = \text{0x47}$

Example:

$(x^6 + x^4 + x^2 + x^1 + 1) \otimes (x^4 + x + 1) = \text{0x57} \otimes \text{0x13}$

$(x^6 + x^4 + x^2 + x^1 + 1) \otimes x = \text{xtime}(\text{0x57}) = \text{0xAE}$

$(x^6 + x^4 + x^2 + x^1 + 1) \otimes x^2 = \text{xtime}(\text{0xAE}) = \text{0x47}$

$(x^6 + x^4 + x^2 + x^1 + 1) \otimes x^3 = \text{xtime}(\text{0x47}) = \text{0x8E}$

$(x^6 + x^4 + x^2 + x^1 + 1) \otimes x^4 = \text{xtime}(\text{0x8E}) = \text{0x7}$

$\text{0x57} \otimes \text{0x13} = \text{0x7} \oplus \text{0xAE} \oplus \text{0x57} = \text{0xFE}$

Secret Key Systems - AES

Byte substitutions (S-Box):

Find the inverse of a polynomial mod *m*(*x*):

$$a(x) \otimes b(x) \quad \oplus \quad m(x) \otimes c(x) \ = \ 1$$

Example:

$0x57 \otimes 0xBF = 1$ so 0xBF is the inverse of 0x57

S-Box number:

Apply the transformation on the right to the inverse

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Example:

getSbox(0x57) = 0x5B

Secret Key Systems - AES

The S-Box:

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Secret Key Systems - AES

The Inverse S-Box:

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Secret Key Systems - AES

Four term polynomials with 8 bit coefficients:

State columns will be represented by such polynomials.
We will want to transform columns by multiplication, modulo a polynomial, as was done to get the S-Box. However, in this case the maximum degree is 3 and the coefficients are bytes, not bits.
The idea is to multiply a column polynomial by the fixed polynomial $a(x) = 3x^3 + x^2 + x + 2 \bmod x^4 + 1$

Equivalently: $a(x) \otimes b(x) = d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

where
$$d_0 = (2 \otimes b_0) \oplus (3 \otimes b_1) \oplus (1 \otimes b_2) \oplus (1 \otimes b_3)$$
$$d_1 = (1 \otimes b_0) \oplus (2 \otimes b_1) \oplus (3 \otimes b_2) \oplus (1 \otimes b_3)$$
$$d_2 = (1 \otimes b_0) \oplus (1 \otimes b_1) \oplus (2 \otimes b_2) \oplus (3 \otimes b_3)$$
$$d_3 = (3 \otimes b_0) \oplus (1 \otimes b_1) \oplus (1 \otimes b_2) \oplus (2 \otimes b_3)$$

Observe: d_i is influenced by all four bytes of original column

Example: $a(x) \otimes (\text{0xB}x^3 + \text{0xD}x^2 + \text{0x9}x + \text{0xE}) = 1$

Secret Key Systems - AES

```
void Cipher () {              // Nr is the number of rounds
    int i, j, round=0;

    // Copy the input PlainText to state array.
    state = in;

    AddRoundKey(0);

    for (round=1 ; round < Nr ; round++) {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }

    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);

    // Copy the state array to the Output array.
    out = state;
}
```

Secret Key Systems - AES

SubBytes ( ):

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

$\mathrm{gSx}(s_{0,0})$	$\mathrm{gSx}(s_{0,1})$	$\mathrm{gSx}(s_{0,2})$	$\mathrm{gSx}(s_{0,3})$
$\mathrm{gSx}(s_{1,0})$	$\mathrm{gSx}(s_{1,1})$	$\mathrm{gSx}(s_{1,2})$	$\mathrm{gSx}(s_{1,3})$
$\mathrm{gSx}(s_{2,0})$	$\mathrm{gSx}(s_{2,1})$	$\mathrm{gSx}(s_{2,2})$	$\mathrm{gSx}(s_{2,3})$
$\mathrm{gSx}(s_{3,0})$	$\mathrm{gSx}(s_{3,1})$	$\mathrm{gSx}(s_{3,2})$	$\mathrm{gSx}(s_{3,3})$

Function $\mathrm{gSx}(a)$ maps a to the character it indexes in the S-Box

Secret Key Systems - AES

ShiftRows ( ):

$$
\begin{array}{|c|c|c|c|}
\hline
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline
s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline
s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline
s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline
\end{array}
\longrightarrow
\begin{array}{|c|c|c|c|}
\hline
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline
s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline
s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline
s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline
\end{array}
$$

N_b \ *Row*	1	2	3
4	1	2	3
6	1	2	3
8	1	3	4

Secret Key Systems - AES

MixColumns ( ) :

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$tmp = s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus s_{3,c}$$
$$s_{0,c} = \mathrm{xtime}(s_{0,c} \oplus s_{1,c}) \oplus tmp \oplus s_{0,c}$$
$$s_{1,c} = \mathrm{xtime}(s_{1,c} \oplus s_{2,c}) \oplus tmp \oplus s_{1,c}$$
$$s_{2,c} = \mathrm{xtime}(s_{2,c} \oplus s_{3,c}) \oplus tmp \oplus s_{2,c}$$
$$s_{3,c} = \mathrm{xtime}(s_{0,c} \oplus s_{3,c}) \oplus tmp \oplus s_{3,c}$$

Replace state columns by matrix multiplication ($\otimes$ and $\oplus$) above

Columns are considered as polynomials over GF(2^8) and multiplied mod x^4+1 by a fixed polynomial given by

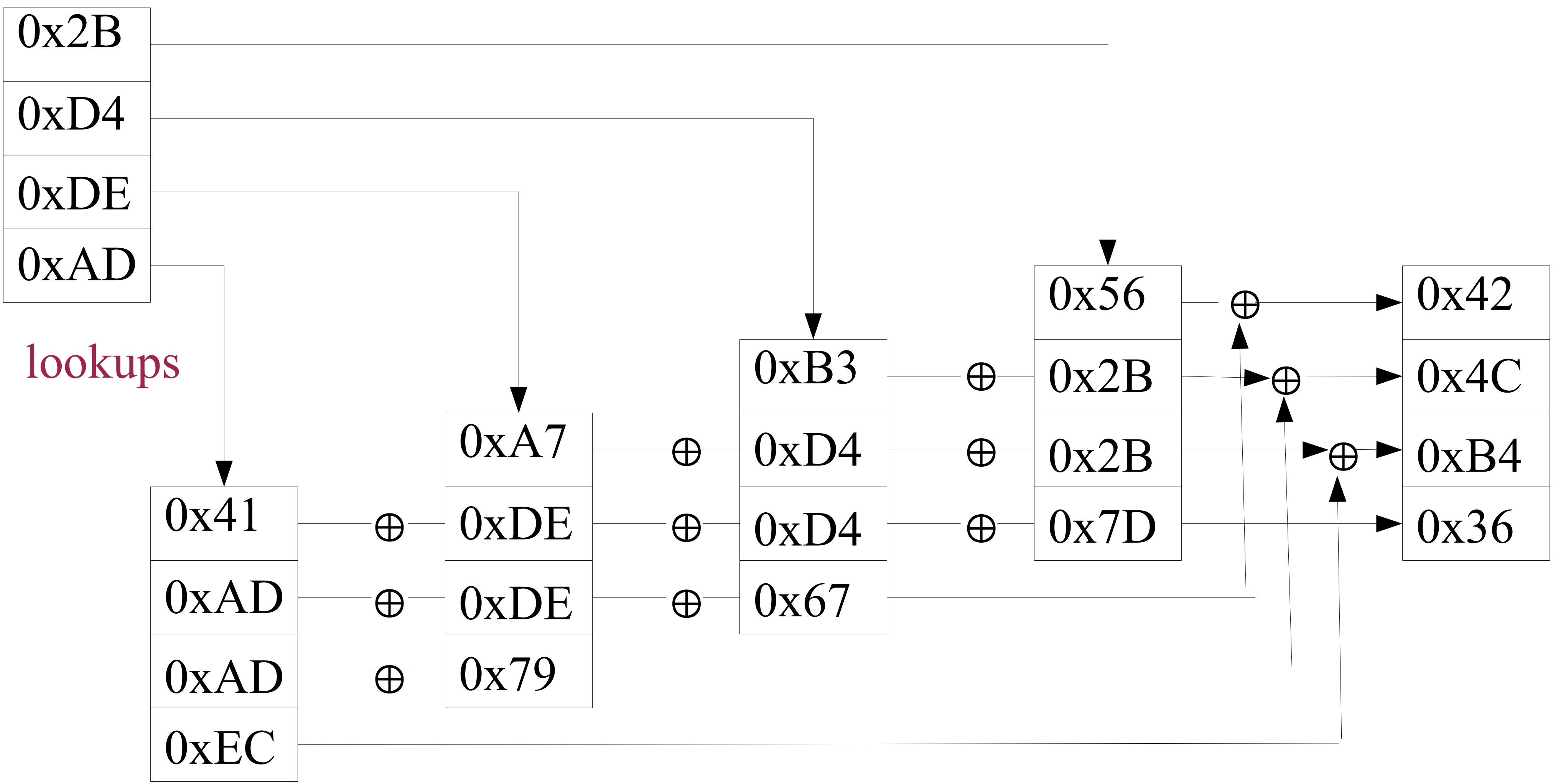
$$3x^3 + x^2 + x + 2$$

Example:

$$\begin{bmatrix} \mathrm{0xD4} \\ \mathrm{0xBF} \\ \mathrm{0x5D} \\ \mathrm{0x30} \end{bmatrix} \rightarrow \begin{bmatrix} \mathrm{0x04} \\ \mathrm{0x66} \\ \mathrm{0x81} \\ \mathrm{0xE5} \end{bmatrix}$$

Secret Key Systems - AES

MixColumns ( ) :



Secret Key Systems - AES

AddRoundKey ( ):

$$
\begin{bmatrix}
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3}
\end{bmatrix}
\oplus
\begin{bmatrix}
w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} \\
w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\
w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \\
w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\
s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\
s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2}
\end{bmatrix}
$$

first round only - generally it's

$$w_{i,r+c}$$

where c is the column and r is the round

Secret Key Systems - AES

Key Schedule: example for N_k=4

$w_{0,0}, w_{1,0}, w_{2,0}, w_{3,0}$	...	$w_{0,3}, w_{1,3}, w_{2,3}, w_{3,3}$	$w_{0,4}, w_{1,4}, w_{2,4}, w_{3,4}$

1st round

2nd round

All rounds: 32*N_k bits for a round key

Secret Key Systems - AES

Key Expansion: example for $N_k = 4$

RotWord ( ): changes $[\ a_0, a_1, a_2, a_3\ ]$ to $[\ a_3, a_2, a_1, a_0\ ]$

Rcon[i]: the word $[\ x^{i-1}, 0, 0, 0\ ] \bmod x^4 + 1$, where $x = 2$

SubWord ( ): maps each byte of $[\ a_0, a_1, a_2, a_3\ ]$ using S-Box values

Secret Key Systems - AES

Rcon[i]: also precomputed as a lookup table

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0		01	02	04	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a
	1	2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39
	2	72	e4	d3	bd	61	c2	9f	25	4a	94	33	66	cc	83	1d	3a
	3	74	e8	cb	8d	01	02	04	08	10	20	40	80	1b	36	6c	d8
	4	ab	4d	9a	2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef
	5	c5	91	39	72	e4	d3	bd	61	c2	9f	25	4a	94	33	66	cc
	6	83	1d	3a	74	e8	cb	8d	01	02	04	08	10	20	40	80	1b
X	7	36	6c	d8	ab	4d	9a	2f	5e	bc	63	c6	97	35	6a	d4	b3
	8	7d	fa	ef	c5	91	39	72	e4	d3	bd	61	c2	9f	25	4a	94
	9	33	66	cc	83	1d	3a	74	e8	cb	8d	01	02	04	08	10	20
	a	40	80	1b	36	6c	d8	ab	4d	9a	2f	5e	bc	63	c6	97	35
	b	6a	d4	b3	7d	fa	ef	c5	91	39	72	e4	d3	bd	61	c2	9f
	c	25	4a	94	33	66	cc	83	1d	3a	74	e8	cb	8d	01	02	04
	d	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a	2f	5e	bc	63
	e	c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39	72	e4	d3	bd
	f	61	c2	9f	25	4a	94	33	66	cc	83	1d	3a	74	e8	cb	

Secret Key Systems - AES

Key Expansion: example for $N_k = 4$

RotWord ( ): changes $[\ a_0, a_1, a_2, a_3\ ]$ to $[\ a_3, a_2, a_1, a_0\ ]$

Rcon[i]: the word $[\ x^{i-1}, 0, 0, 0\ ] \bmod x^4 + 1$, where $x = 2$

SubWord ( ): maps each byte of $[\ a_0, a_1, a_2, a_3\ ]$ using S-Box values

First Round: the original key (16 bytes if $N_k = 4$)

Other Rounds:

$[\ w_{0,l}, w_{1,l}, w_{2,l}, w_{3,l}\ ]$ smallest l is N_k-1

$[\ w_{3,l}, w_{2,l}, w_{1,l}, w_{0,l}\ ]$ apply RotWord

$[\ S(w_{0,l}), S(w_{1,l}), S(w_{2,l}), S(w_{3,l})\ ]$ apply SubWord

$[\ S(w_{0,l}) \oplus \mathrm{Rcon}[(l+1)/N_k], S(w_{1,l}), S(w_{2,l}), S(w_{3,l})\ ]$ use Rcon

$$[\ S(w_{0,l}) \oplus \mathrm{Rcon}[(l+1)/N_k] \oplus w_{0,l}\ , S(w_{1,l}) \oplus w_{1,l}\ , S(w_{2,l}) \oplus w_{2,l}\ , S(w_{3,l}) \oplus w_{3,l}]$$

$[\ w_{0,l+1}, w_{1,l+1}, w_{2,l+1}, w_{3,l+1}\ ]$ next key word

Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i–Nk]	w[i]= temp XOR w[i–Nk]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943
10	7a96b943					23a33939	5935807a
11	5935807a					2a6c7605	7359f67f
12	7359f67f	59f67f73	cb42d28f	04000000	cf42d28f	f2c295f2	3d80477d
13	3d80477d					7a96b943	4716fe3e
14	4716fe3e					5935807a	1e237e44
15	1e237e44					7359f67f	6d7a883b
16	6d7a883b	7a883b6d	dac4e23c	08000000	d2c4e23c	3d80477d	ef44a541
17	ef44a541					4716fe3e	a8525b7f
18	a8525b7f					1e237e44	b671253b
19	b671253b					6d7a883b	db0bad00
20	db0bad00	0bad00db	2b9563b9	10000000	3b9563b9	ef44a541	d4d1c6f8
21	d4d1c6f8					a8525b7f	7c839d87
22	7c839d87					b671253b	caf2b8bc
23	caf2b8bc					db0bad00	11f915bc

Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

24	11f915bc	f915bc11	99596582	20000000	b9596582	d4d1c6f8	6d88a37a
25	6d88a37a					7c839d87	110b3efd
26	110b3efd					caf2b8bc	dbf98641
27	dbf98641					11f915bc	ca0093fd
28	ca0093fd	0093fdca	63dc5474	40000000	23dc5474	6d88a37a	4e54f70e
29	4e54f70e					110b3efd	5f5fc9f3
30	5f5fc9f3					dbf98641	84a64fb2
31	84a64fb2					ca0093fd	4ea6dc4f
32	4ea6dc4f	a6dc4f4e	2486842f	80000000	a486842f	4e54f70e	ead27321
33	ead27321					5f5fc9f3	b58dbad2
34	b58dbad2					84a64fb2	312bf560
35	312bf560					4ea6dc4f	7f8d292f
36	7f8d292f	8d292f7f	5da515d2	1b000000	46a515d2	ead27321	ac7766f3
37	ac7766f3					b58dbad2	19fadc21
38	19fadc21					312bf560	28d12941
39	28d12941					7f8d292f	575c006e
40	575c006e	5c006e57	4a639f5b	36000000	7c639f5b	ac7766f3	d014f9a8
41	d014f9a8					19fadc21	c9ee2589
42	c9ee2589					28d12941	e13f0cc8
43	e13f0cc8					575c006e	b6630ca6

Secret Key Systems - AES

Example:

Input:

32	88	31	E0
43	5A	31	37
F6	30	98	07
A8	8D	A2	34

Key:

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

Secret Key Systems - AES

Example:

Input:

32	88	31	E0
43	5A	31	37
F6	30	98	07
A8	8D	A2	34

$\oplus$

19	A0	9A	E9
3D	F4	C6	F8
E3	E2	8D	48
BE	2B	2A	08

Key:

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

beginning of first round - input placed into the state and key has been added to state

Secret Key Systems - AES

Example:

State:

19	A0	9A	E9
3D	F4	C6	F8
E3	E2	8D	48
BE	2B	2A	08

S-Box

D4	E0	B8	1E
27	BF	B4	41
11	98	5D	52
AE	F1	E5	30

ShiftRows

D4	E0	B8	1E
BF	B4	41	27
5D	52	11	98
30	AE	F1	E5

MixColumns

04	E0	48	28
66	CB	F8	06
81	19	D3	26
E5	9A	7A	4C

Secret Key Systems - AES

Performance Notes:

1. Many operations are table look ups so they are fast
2. Parallelism can be exploited
3. Key expansion only needs to be done one time until the key is changed
4. The S-box minimizes the correlation between input and output bits
5. There are no known weak keys

Secret Key Systems - AES

Attacks:

- **Extended Sparse Linearization** -
  Derive a system of quadratic simultaneous equations and solve,
  128 bit key: 8000 quadratic equations, 1600 variables
  256 bit key: 22400 quadratic equations, 4480 variables
  given plaintext, to get the key – not practical although 2^{100} vs. 2^{128}

Secret Key Systems - AES

Attacks:

- **Extended Sparse Linearization** -
  Derive a system of quadratic simultaneous equations and solve,
  128 bit key: 8000 quadratic equations, 1600 variables
  256 bit key: 22400 quadratic equations, 4480 variables
  given plaintext, to get the key – not practical although 2^{100} vs. 2^{128}
- **Related Key** -
  Attacker may be able to observe behavior of cipher in the case of several keys, not initially known, but with some understanding of the mathematical relationship connecting the keys - e.g. the number of 1s equals the number of 0s – 2^{70} time for an 11 round version of 256-bit AES but attack on full version is not reported.

Secret Key Systems - AES

Attacks:

- **Extended Sparse Linearization** -
  Derive a system of quadratic simultaneous equations and solve,
  128 bit key: 8000 quadratic equations, 1600 variables
  256 bit key: 22400 quadratic equations, 4480 variables
  given plaintext, to get the key – not practical although 2^{100} vs. 2^{128}
- **Related Key** -
  Attacker may be able to observe behavior of cipher in the case of several keys, not initially known, but with some understanding of the mathematical relationship connecting the keys - e.g. the number of 1s equals the number of 0s – 2^{70} time for an 11 round version of 256-bit AES but attack on full version is not reported.
- **Titled Attack** -
  If attacker can stop the execution of encryption, apply a "difference" to the state, and roll back the encryption. 2^{32} vs. 2^{128}

Secret Key Systems - AES

Attacks:

- **Extended Sparse Linearization** -
  Derive a system of quadratic simultaneous equations and solve,
  128 bit key: 8000 quadratic equations, 1600 variables
  256 bit key: 22400 quadratic equations, 4480 variables
  given plaintext, to get the key – not practical although 2^{100} vs. 2^{128}
- **Related Key** -
  Attacker may be able to observe behavior of cipher in the case of several keys, not initially known, but with some understanding of the mathematical relationship connecting the keys - e.g. the number of 1s equals the number of 0s – 2^{70} time for an 11 round version of 256-bit AES but attack on full version is not reported.
- **Titled Attack** -
  If attacker can stop the execution of encryption, apply a "difference" to the state, and roll back the encryption. 2^{32} vs. 2^{128}
- **Side Channel** -
  Vulnerable in some implementations – later in the course

Secret Key Systems - AES

Number of rounds:

N_k \ N_b	4	6	8
4	10	12	14
6	12	12	14
8	14	14	14