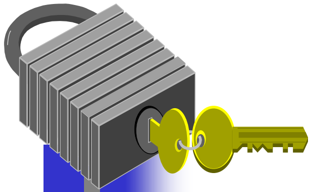
Computer and Information Security

Chapter 3
Symmetric Key Crypto

Chapter 3: Symmetric Key Crypto

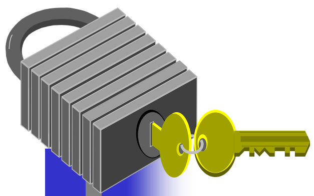
The chief forms of beauty are order and symmetry…
— Aristotle

"You boil it in sawdust: you salt it in glue:
You condense it with locusts and tape:
Still keeping one principal object in view —
To preserve its symmetrical shape."
— Lewis Carroll, *The Hunting of the Snark*



Symmetric Encryption

- Called conventional/private-key single-key encryption
- Sender and recipient share a common key
- All classical encryption algorithms are private-key
- Symmetric Encryption was the only type prior to invention of public-key in 1970's and is most widely used



Conventional Encryption Principles

- An encryption scheme has five ingredients
  - Plain text
  - Encryption algorithms
  - Public and private keys
  - Cipher text
  - Decryption algorithm
- Agents possess their private keys
- Access other public keys from a central repository
- Security depends on the secrecy of the key, not the secrecy of the algorithm

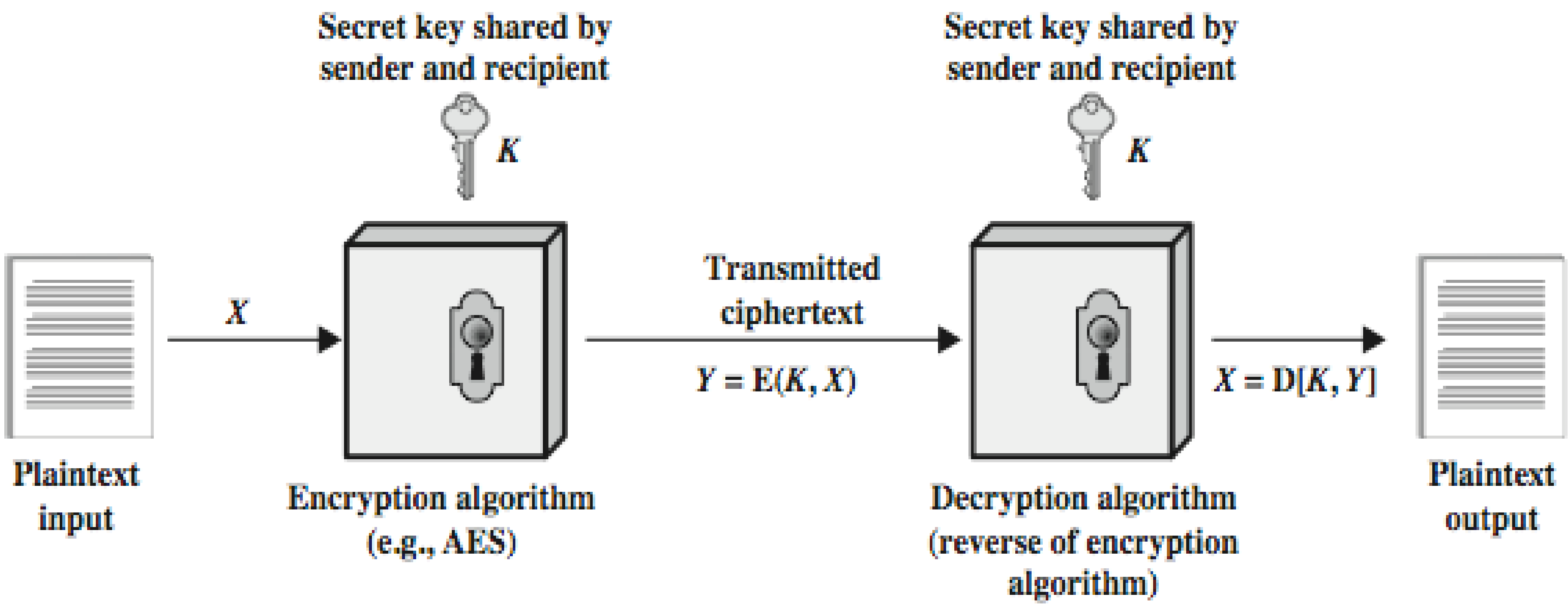
Conventional Encryption

Algorithm Components:

- **Plain Text-** original data or input
- **Encryption Algorithm-** performs substitiutions or transformations on the pliantext
- **Public and Private Keys-** also input determines the substitutions/transpositions
- **Cipher Text-** scrambled message or output
- **Decryption Algorithm-** encryption algorithm run backward, taking the cipher text and producing the plain text.

https://www.youtube.com/watch?v=fNC3jCCGJ0o

Conventional Encryption Principles



[Asymmetric Encryption: https://www.youtube.com/watch?v=E5FEqGYLL0o](https://www.youtube.com/watch?v=E5FEqGYLL0o)

Conventional Encryption

- More rigorous definition
- **Five components to the algorithm**
  - A **Plaintext message space**, $\mathcal{M}$
  - A family of **enciphering transformations**, $E_K: \mathcal{M} \rightarrow \mathcal{C}$, where $K \in \mathcal{K}$
  - A **key space**, $\mathcal{K}$
  - A **ciphertext message space**, $\mathcal{C}$
  - A family of **deciphering transformations**, $D_K: \mathcal{C} \rightarrow \mathcal{M}$, where $K \in \mathcal{K}$

Conventional Encryption

$$M \longrightarrow \boxed{E_K} \longrightarrow C \longrightarrow \boxed{D_K} \longrightarrow M$$

E_K defined by an encrypting algorithm E
D_K defined by an decrypting algorithm D

For given K, D_K is the **inverse** of E_K, i.e.,
$D_K(E_K(M))=M$
for every plain text message M

Requirements

- Two requirements for secure use of symmetric encryption:
  - a strong encryption algorithm
  - a secret key known only to sender / receiver
- Mathematically have:

  $Y = E(K, X)$

  $X = D(K, Y)$
- Assume encryption algorithm is known
- Implies a secure channel to distribute key

Symmetric Encryption

- Security depends on the secrecy of the key, NOT the secrecy of the algorithm
- Do not need to keep the algorith secret- only the key
- This feature makes symmetric encryption feasible for widespread use.

Cryptography

- Classified according to three independent dimensions:
  - The type of operations used for transforming plaintext to ciphertext
    - Substitution
    - Transposition
    - Product
  - The number of keys used
    - **Symmetric** (single key or secret- key or private-key)
    - **Asymmetric** (two-keys, or public-key encryption)
  - The way in which the plaintext is processed
    - Block- a block at a time
    - Stream- one element at a time

Symmetric Key Crypto

- Stream cipher — based on one-time pad
  - Except that key is relatively short
  - Key is stretched into a long **keystream**
  - Keystream is used just like a one-time pad
- Block cipher — based on codebook concept
  - Block cipher key determines a codebook
  - Each key yields a different codebook
  - Employs both "confusion" and "diffusion"

Random Numbers

- Many uses of **random numbers** in cryptography
  - nonces in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- Critical that these values be
  - statistically random, uniform distribution, independent (eg. Same number of 0's and 1's)
  - unpredictability of future values from previous values
- True random numbers provide this
- Care needed with generated random numbers

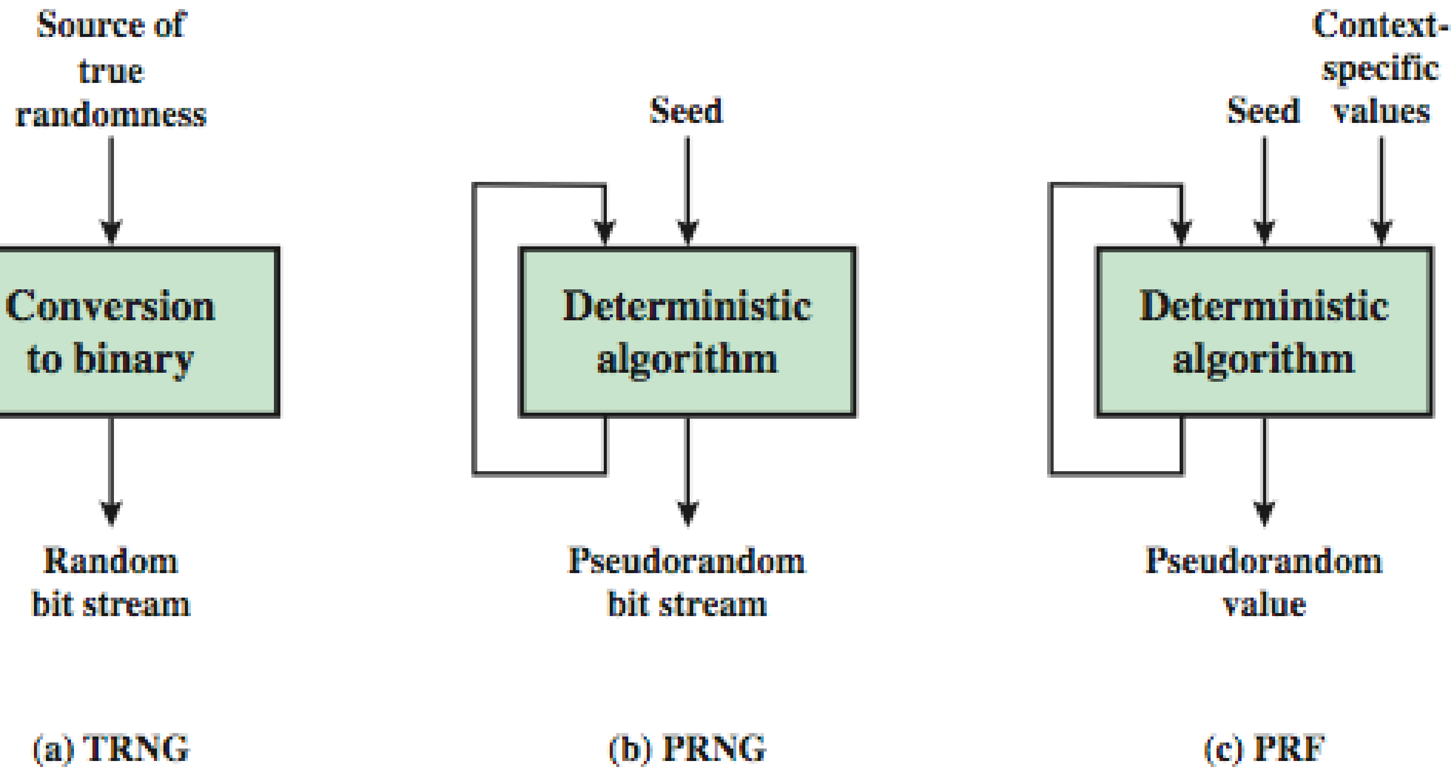
Pseudo Random Number Generators (PRNGs)

- Often use deterministic algorithmic techniques to create "random numbers"
  - although are not truly random
  - can pass many tests of "randomness"
- Known as "pseudorandom numbers"
- Created by "Pseudorandom Number Generators (PRNGs)"

True and Pseudo Random Numbers

- TRNG – true random number generator
  - Takes a source that is random, entropy source, such as the system clock
- PRNG – pseudorandom number generator
  - Takes a fixed value called the seed
  - Produces output using a deterministic algorithm
- PRF - pseudorandom function
  - takes as input a seed plus some context specific values, such as a user ID or an application ID.

True and Pseudo Random Numbers



Stream Ciphers

Stream Ciphers

- Once upon a time, not so very long ago, stream ciphers were the king of crypto
- Today, not as popular as block ciphers
- We'll discuss two stream ciphers…
- A5/1
  - Based on shift registers
  - Used in GSM mobile phone system
- RC4
  - Based on a changing lookup table
  - Used many places

Stream Cipher

- A key is input to a pseudorandom bit generator that produces an apparently random keystream of bits.
- These bits are XOR'd with message to encrypt it,
- They are XOR'd again to decrypt it by the receiver.

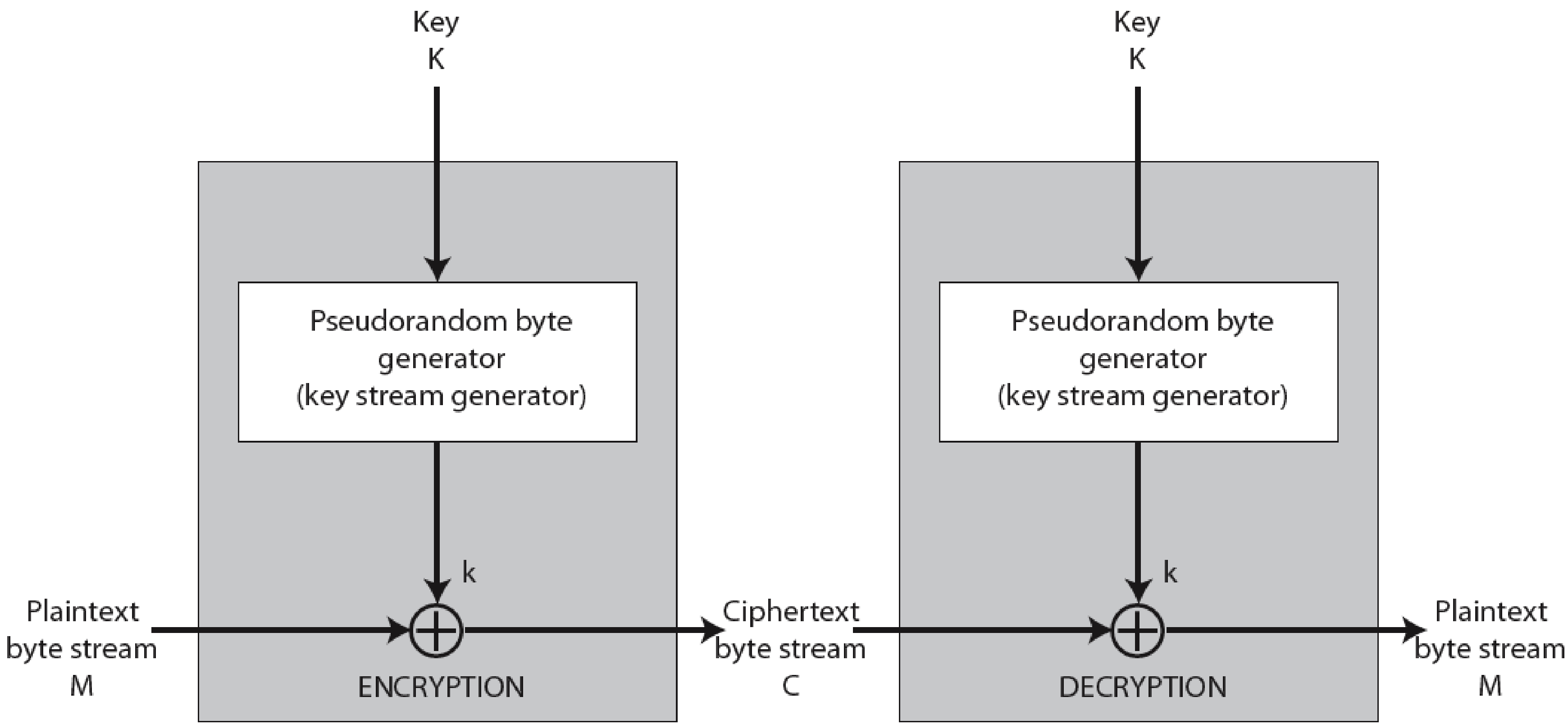
Block and Stream Ciphers

- A **block cipher** inputs a block of elements and produces and output block for each input block.
- A **stream cipher** processes the input elements continuously, producing output one element at a time.
- Block ciphers are more common, but there are applications which use stream ciphers.

Stream Cipher Properties

- Some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
- Properly designed, can be as secure as a block cipher with same size key
- Usually simpler & faster

Stream Cipher Structure



A5/1: Shift Registers

- Used for confidentiality in GSM (Global System for Mobile) cell phones
- A5/1 uses 3 *shift registers*
  - X: 19 bits $(x_0, x_1, x_2, \ldots, x_{18})$
  - Y: 22 bits $(y_0, y_1, y_2, \ldots, y_{21})$
  - Z: 23 bits $(z_0, z_1, z_2, \ldots, z_{22})$
  - *Note total bits =64 (power of 2).

A5/1: Majority Function

- A5/1 also uses a majority function
- A majority function such as maj(x,y,z) is defined as a function from multiple inputs to one output.
- Its value is false when n/2 or more inputs are false and true otherwise.
- Here there are an odd number of bits so there cannot be a tie.

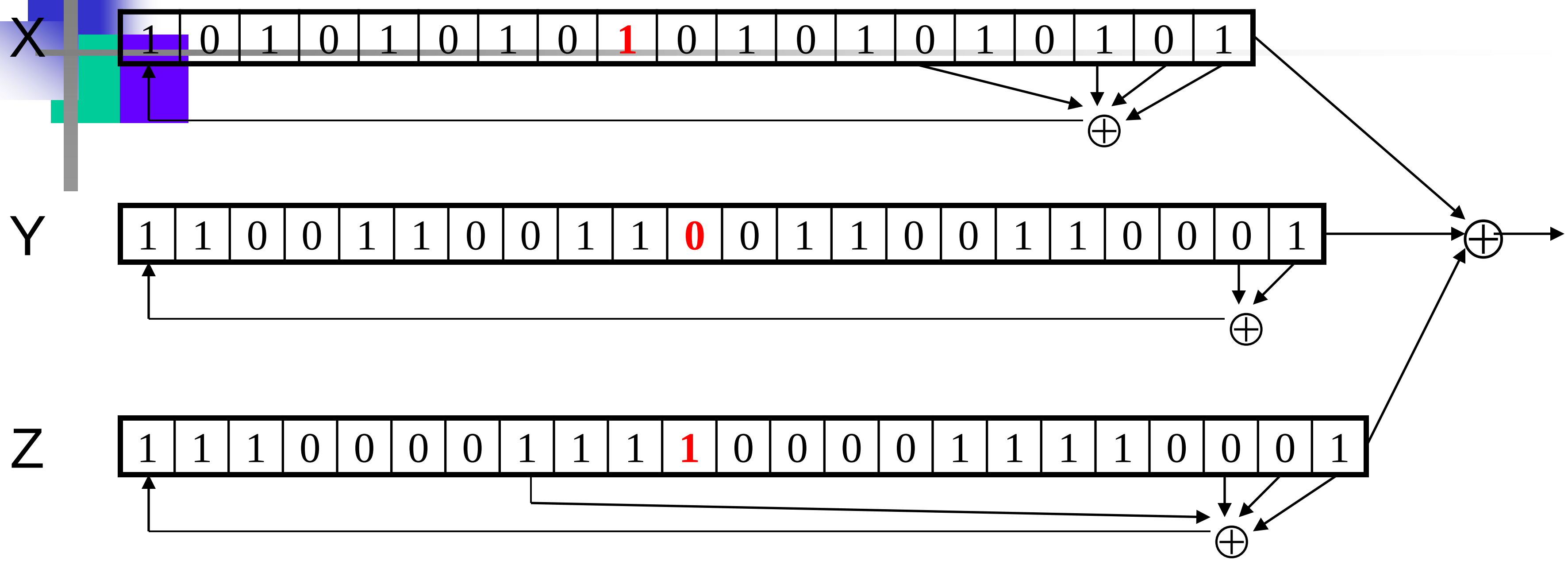
A5/1: Keystream

- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X *steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y *steps*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ and $y_0 = t$
- If $z_{10} = m$ then Z *steps*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$
- Keystream **bit** is $x_{18} \oplus y_{21} \oplus z_{22}$

A5/1

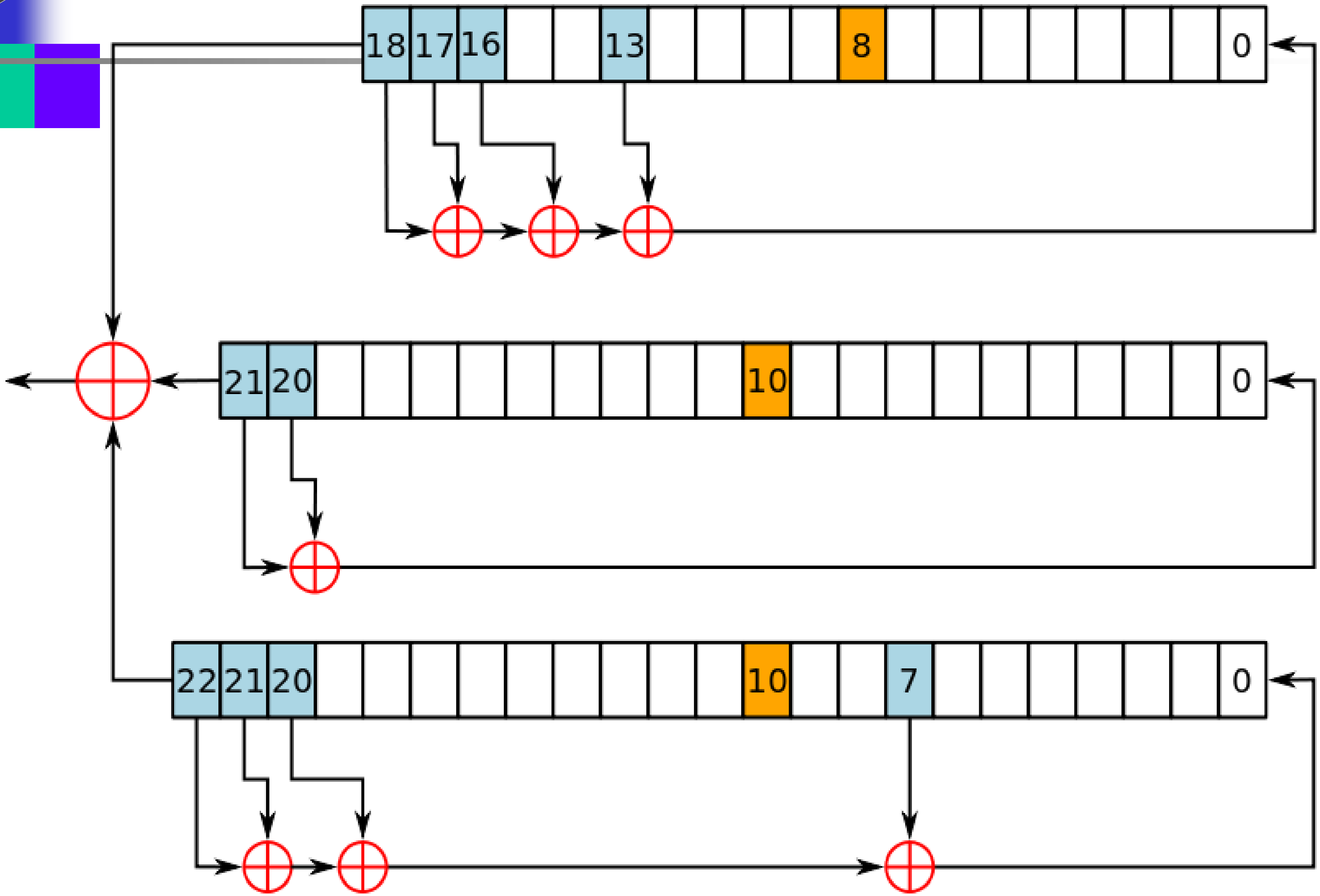
- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on $\mathrm{maj}(x_8, y_{10}, z_{10})$
- Keystream bit is XOR of rightmost bits of registers

A5/1



- In this example, $m = \mathrm{maj}(x_8, y_{10}, z_{10}) = \mathrm{maj}(\mathbf{1},\mathbf{0},\mathbf{1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

A5/1



http://en.wikipedia.org/wiki/Stream_cipher

Shift Register Crypto

- Shift register crypto efficient in hardware
- Often, slow if implement in software
- In the past, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used some
  - Resource-constrained devices

RC4

- A proprietary cipher owned by RSA Security
- A **Ron Rivest** design, simple but effective, based on random permutation
- Variable key size, byte-oriented stream cipher
- Widely used
  - SSL/TLS web security protocol
  - Wireless WEP/WPA LAN security protocols
- Key forms random permutation of all 8-bit values
- Uses that permutation to scramble input info processed a byte at a time
- Kept secret until anonymously posted on the Internet

RC4

A self-modifying lookup table

- Table always contains a permutation of the byte values 0,1,…,255
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a key stream byte from table
- Each step of RC4 produces a **byte**
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware

RC4 Algorithm

- Starts with an array S of numbers 0..255
- Use key to shuffle array
- S forms **internal state** of the cipher

```
for i = 0 to 255 do                    // Initialization
    S[i] = i
    T[i] = K[i mod keylen]
j = 0                                  // Initial permutation of S
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

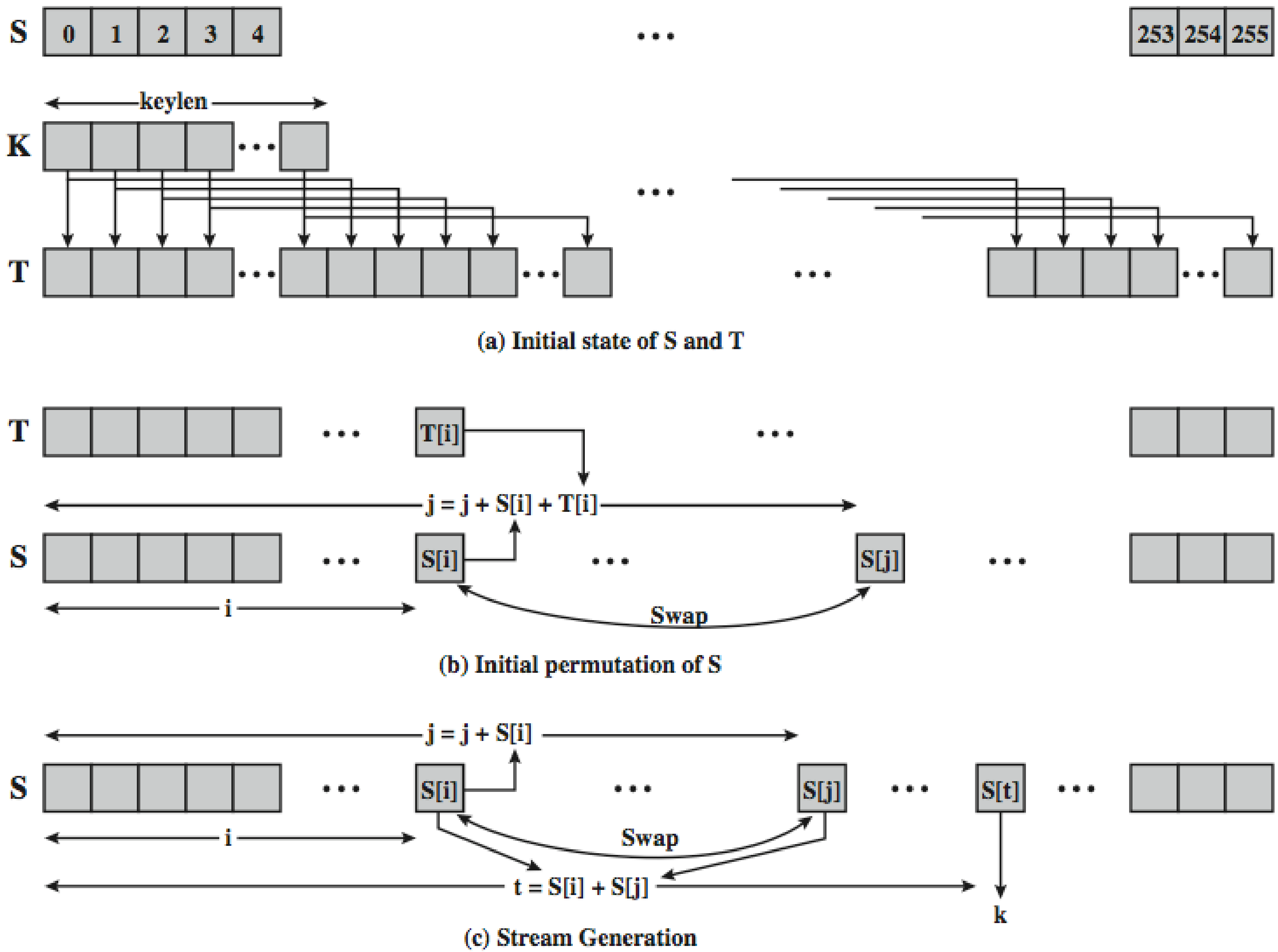
- Total number of possible states is 256!

RC4 Encryption

- Encryption continues shuffling array values
- Sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] with next byte of message to en/decrypt

```
i = j = 0;                        //Stream Generation
while (true)                      //for each message byte M_i
    i = (i + 1) (mod 256);
    j = (j + S[i]) (mod 256);
    swap(S[i], S[j]);
    t = (S[i] + S[j]) (mod 256);
    k= S[t];
    C_i = M_i XOR S[t]   or M_i = C_i XOR S[t]; (to Encrypt/Decrypt)
```

RC4 Overview



RC4 Security

- Claimed secure against known attacks
  - have some analyses, none practical
- Result is very non-linear
- Since RC4 is a stream cipher, must **never reuse a key**
- Concern with WEP, but due to key handling rather than RC4 itself
  - Secure with key length of at least 128 bits

RC4 Initialization

- S[ ] is permutation of 0,1,...,255
- key[ ] contains N bytes of key

```
for i = 0 to 255
      S[i] = i
      K[i] = key[i (mod N)]
next i
j = 0
for i = 0 to 255
      j = (j + S[i] + K[i]) mod 256
      swap(S[i], S[j])
next i
i = j = 0
```

RC4 Keystream

- For each keystream byte, swap elements in table and select byte

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```

- Use keystream bytes like a one-time pad
- **Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists

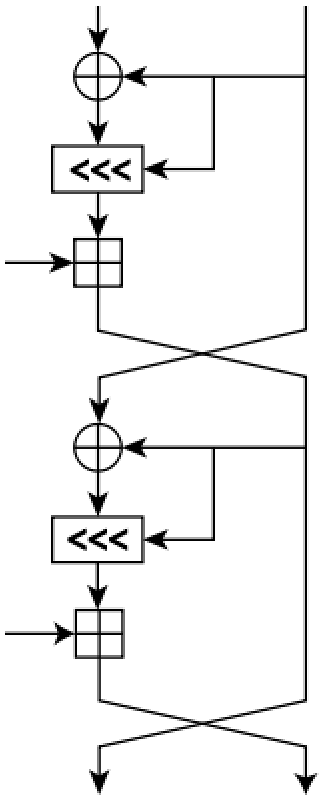
https://www.youtube.com/watch?v=riIp6EQOJOg

RC5

- **RC5 – designed by Ron Rivest (1994)**
  - Block cipher
  - Suitable for hardware and software
  - Fast, simple
  - Adaptable to processors of different word lengths
  - Variable block size (32, 64, 128 bits)
  - Variable number of rounds (0 to 255)
  - Variable-length key (0 to 2040 bits)
  - Low memory requirement, High security
  - Data-dependent rotations
  - Modulo additions and exclusive ORs (XOR)
  - **Feistel-like** structure

RC5

Two half- rounds of RC5



See http://en.wikipedia.org/wiki/RC5 for details

Vigenere Cipher

The **Vigenère cipher** is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword. Its advantage is that it cannot be broken by frequency analysis.

- It is a simple form of polyalphabetic substitution.
- http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher
- Once considered unbreakable, but with modern computers it can be broken.
- See simple examples
- http://www.youtube.com/watch?v=ijC2-JHz6Z4&NR=1&feature=endscreen (4+ min)
- http://www.youtube.com/watch?v=SseaQvc0aXo&feature=related

Vigenere Table

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenere Cipher

Plaintext:	ATTACKATDAWN
Key:	LEMONLEMONLE
Ciphertext:	LXFOPVEFRNHR

- Use key such as LEMON, repeated.
- To encrypt use the row of the key (L) and the column of the plaintext (A) to determine the cipher (L)
- To decrypt use the row of the key (L) and the column of the cipher text (L) to find the column label , which is the plaintext (A)
- Repeat for each letter in the message.

http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

Vigenere Cipher Resources

Here are some other sources for information and interaction with the Vigenere cipher:

- http://www.cs.trincoll.edu/~crypto/historical/vigenere.html
- http://sharkysoft.com/misc/vigenere/
- http://www.simonsingh.net/The_Black_Chamber/vigenere_cipher.html

Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - Shamir declared "the death of stream ciphers"
  - May be greatly exaggerated…
  - http://en.wikipedia.org/wiki/Stream_cipher

Block Ciphers

Block Ciphers

- Block Cipher is a symmetric key cipher operating on fixed-length groups of bits, called *blocks*, with an unvarying transformation. A block cipher encryption algorithm might take (for example) a 128-bit block of plaintext as input, and output a corresponding 128-bit block of cipher text. The exact transformation is controlled using a second input — the secret key.
- Short explanation
  - DES, 3DES, AES, IDEA

(Iterated) Block Cipher

- Plaintext and ciphertext consist of fixed-sized blocks
- Ciphertext obtained from plaintext by iterating a **round function**
- Input to round function consists of ***key*** and ***output*** of previous round
- Usually implemented in software

Feistel Cipher

- Horst Feistel devised the **Feistel Cipher**
  - based on concept of invertible product cipher
- Partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves
- Implements Shannon's S-P net concept
- https://www.youtube.com/watch?v=ySZvE9vOfEQ

Feistel Cipher Structure

- Virtually all conventional block encryption algorithms, including DES have a structure first described by Horst Feistel of IBM in 1973
- The realization of a Feistel Network depends on the choice of the following parameters and design features:

  (see next slide)
- http://www.youtube.com/watch?v=ySZvE9vOfEQ

Feistel Cipher: Encryption

- **Feistel cipher** is a type of block cipher, not a specific block cipher
- Split plaintext block into left and right halves: $P = (L_0,R_0)$
- For each round $i = 1,2,\ldots,n$, compute

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus F(R_{i-1},K_i)$

  where F is **round function** and K_i is **subkey**
- Ciphertext: $C = (L_n,R_n)$

Feistel Cipher: Decryption

- Start with ciphertext $C = (L_n, R_n)$
- For each round $i = n, n-1, \ldots, 1$, compute

  $R_{i-1} = L_i$
  $L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$
  where F is round function and K_i is subkey
- Plaintext: $P = (L_0, R_0)$
- Formula "works" for any function F
  - But only secure for certain functions F

Feistel Cipher Structure

- **Block size:** larger block sizes mean greater security
- **Key Size:** larger key size means greater security
- **Number of rounds:** multiple rounds offer increasing security
- **Subkey generation algorithm:** greater complexity will lead to greater difficulty of cryptanalysis.
- **Fast software encryption/decryption:** the speed of execution of the algorithm becomes a concern

Feistel Cipher

- The plain text block is divided into two halves L_0 and R_0
- The two halves pass through n rounds of processing and then combine to produce the cipher text block.
- At each round a substitution is performed on the left half of the data by applying a round function F to the right half of the data and then XORing it with the left half

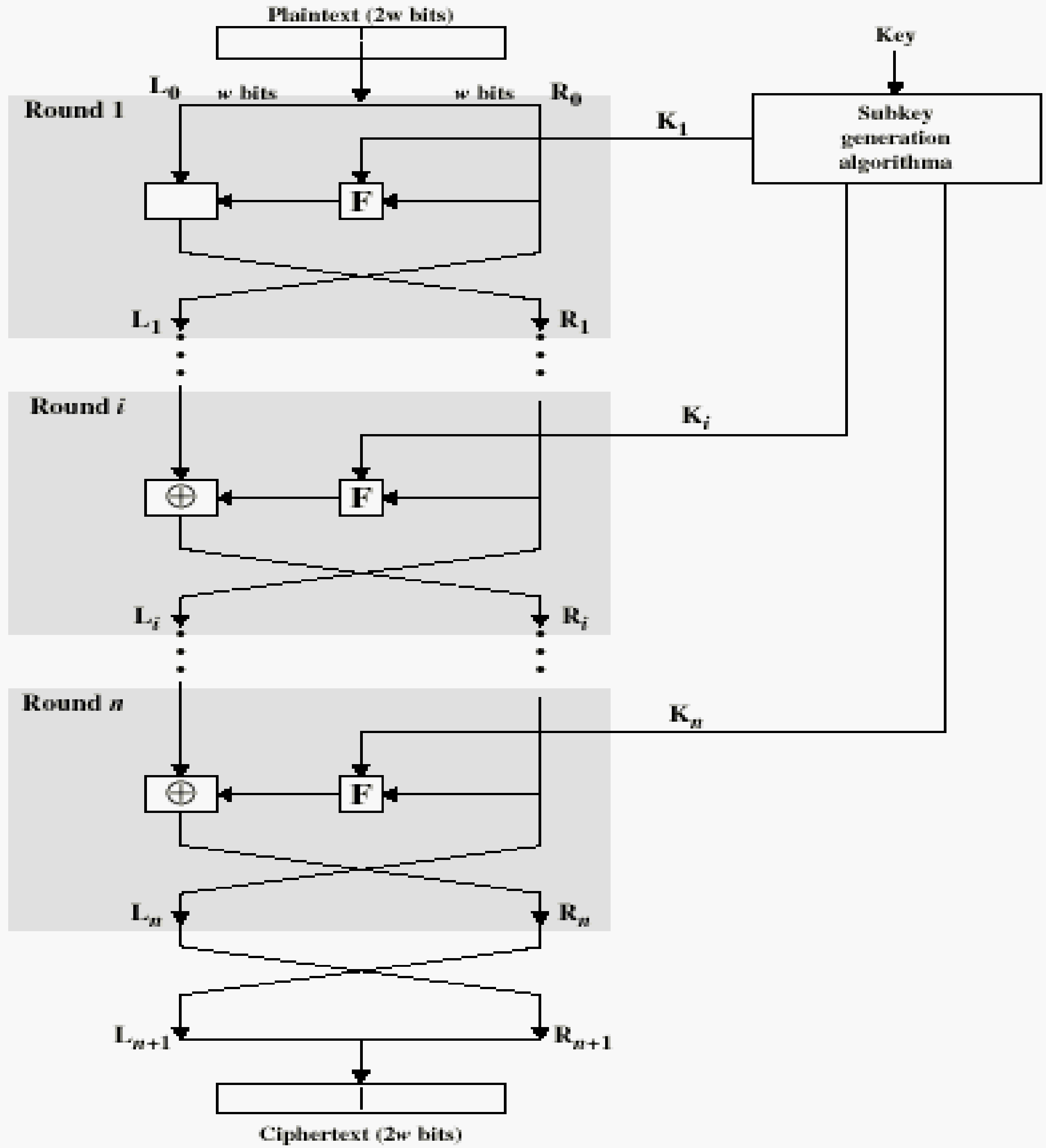


Figure 2.2 Classical Feistel Network

Feistel Structure

- The Feistel Structure is a general example used by all symmetric block ciphers:
- It is a series of rounds, each performing substitutions and permutations using a secret key value

Feistel Cipher Design Elements

- block size
- key size
- number of rounds

Increasing size means greater security, but slows cipher

- subkey generation algorithm
- round function

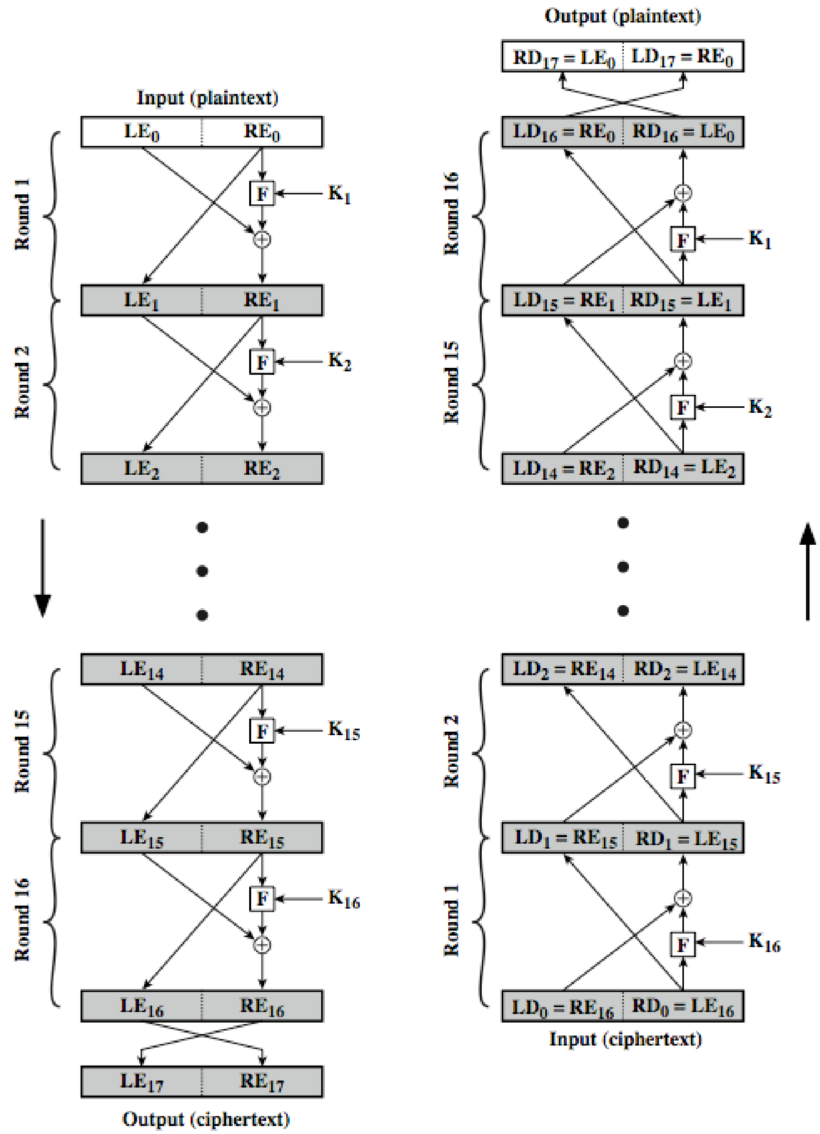
Greater complexity, harder to decrypt

Other Considerations

- fast software en/decryption
- ease of analysis

Feistel Cipher Structure

Encryption



Decryption

Feistel Algorithms

- Encryption and decryption algorithms are essentially the same
- To decrypt:
  - Use the ciphertext as input
  - Use the sub-keys in reverse order (K_n, K_{n-1}...)
- Advantage:
  - Only one algorithm is needed for encryption and decryption

Symmetric Block Encryption Algorithms

- Most common symmetric encryption algorithms are block ciphers.
- Block Ciphers process plaintext input in fixed size blocks and produce a block of equal size cipher text.
  - DES - Data Encryption Standard
  - 3DES - Triple DES
  - AES - Advanced Encryption Standard

Data Encryption Standard

- DES developed in 1970's
- Based on IBM's Lucifer cipher
- DES was U.S. government standard
- DES development was controversial
  - NSA secretly involved
  - Design process was secret
  - Key length reduced from 128 to 56 bits
  - Subtle changes to Lucifer algorithm

https://www.youtube.com/watch?v=G_guTnTcoqg

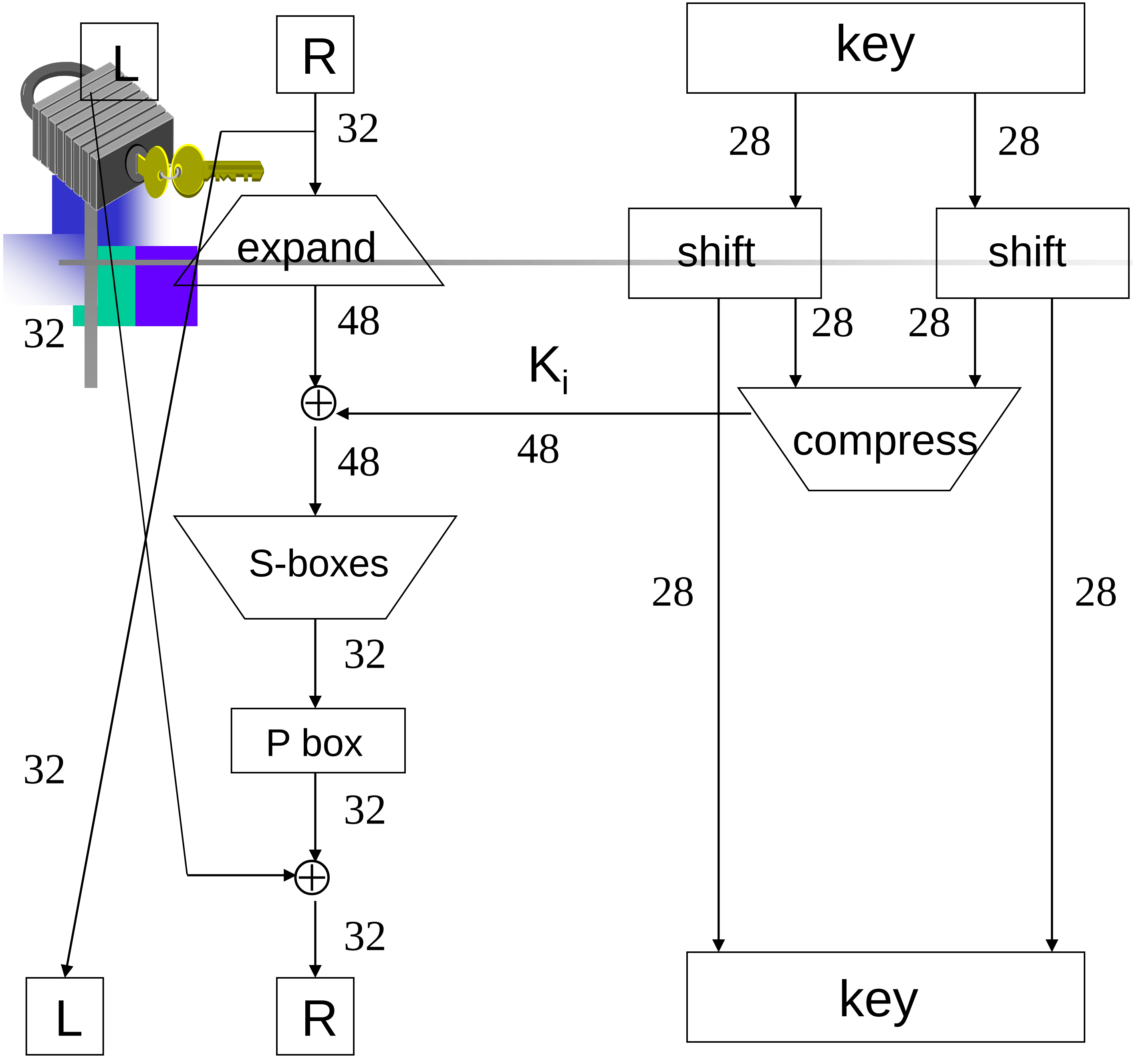
Conventional Symmetric Encryption Algorithms

Data Encryption Standard (DES)- 1977

- The most widely used encryption scheme
- The algorithm is reffered to the Data Encryption Algorithm (DEA)
- DES is a block cipher
- Variation of Feistel Cipher
- The plaintext is processed in 64-bit blocks
- The key is 56-bits in length
- 16 subkeys used in 16 rounds
- <u>No longer used for government transmissions</u>
- <u>Controversy over security</u>

DES Numerology

- DES is a Feistel cipher with…
  - 64 bit block length
  - 56 bit key length
  - 16 rounds
  - 48 bits of key used each round (subkey)
- Each round is simple (for a block cipher)
- Security depends heavily on "S-boxes"
  - Each S-boxes maps 6 bits to 4 bits



One Round of DES

DES Expansion Permutation

- Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- Output 48 bits

31	0	1	2	3	4	3	4	5	6	7	8
7	8	9	10	11	12	11	12	13	14	15	16
15	16	17	18	19	20	19	20	21	22	23	24
23	24	25	26	27	28	27	28	29	30	31	0

DES S-box

- 8 “substitution boxes” or *S-boxes*
- Each S-box <u>maps 6 bits to 4 bits</u>
- S-box number 1

```
input bits (0,5)
↓                               input bits (1,2,3,4)
   | 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
```

DES P-box

- P - permutes or rearranges the bits
- Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- Output 32 bits

15	6	19	20	28	11	27	16	0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8	18	12	29	5	21	10	3	24

DES Subkey

- 56 bit DES key, numbered 0,1,2,…,55
- Left half key bits, `LK`

49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31

- Right half key bits, `RK`

55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3

DES Subkey

- For rounds `i=1,2,...,16`
  - Let LK = (LK circular shift left by r_i)
  - Let RK = (RK circular shift left by r_i)
  - Left half of subkey K_i is of LK bits

```
13 16 10 23  0  4  2 27 14  5 20  9
22 18 11  3 25  7 15  6 26 19 12  1
```

  - Right half of subkey K_i is RK bits

```
12 23  2  8 18 26  1 11 22 16  4 19
15 20 10 27  5 24 17 13 21  7  0  3
```

DES Subkey

- For rounds 1, 2, 9 and 16 the shift r_i is 1, and in all other rounds r_i is 2
- Bits 8,17,21,24 of LK omitted each round
- Bits 6,9,14,25 of RK omitted each round
- **Compression permutation** yields 48 bit subkey K_i from 56 bits of LK and RK
- **Key schedule** generates subkey

DES Last Word (Almost)

- An initial permutation before round 1
- Halves are swapped after last round
- A final permutation (inverse of initial perm) applied to (R_{16},L_{16})
- None of this serves security purpose

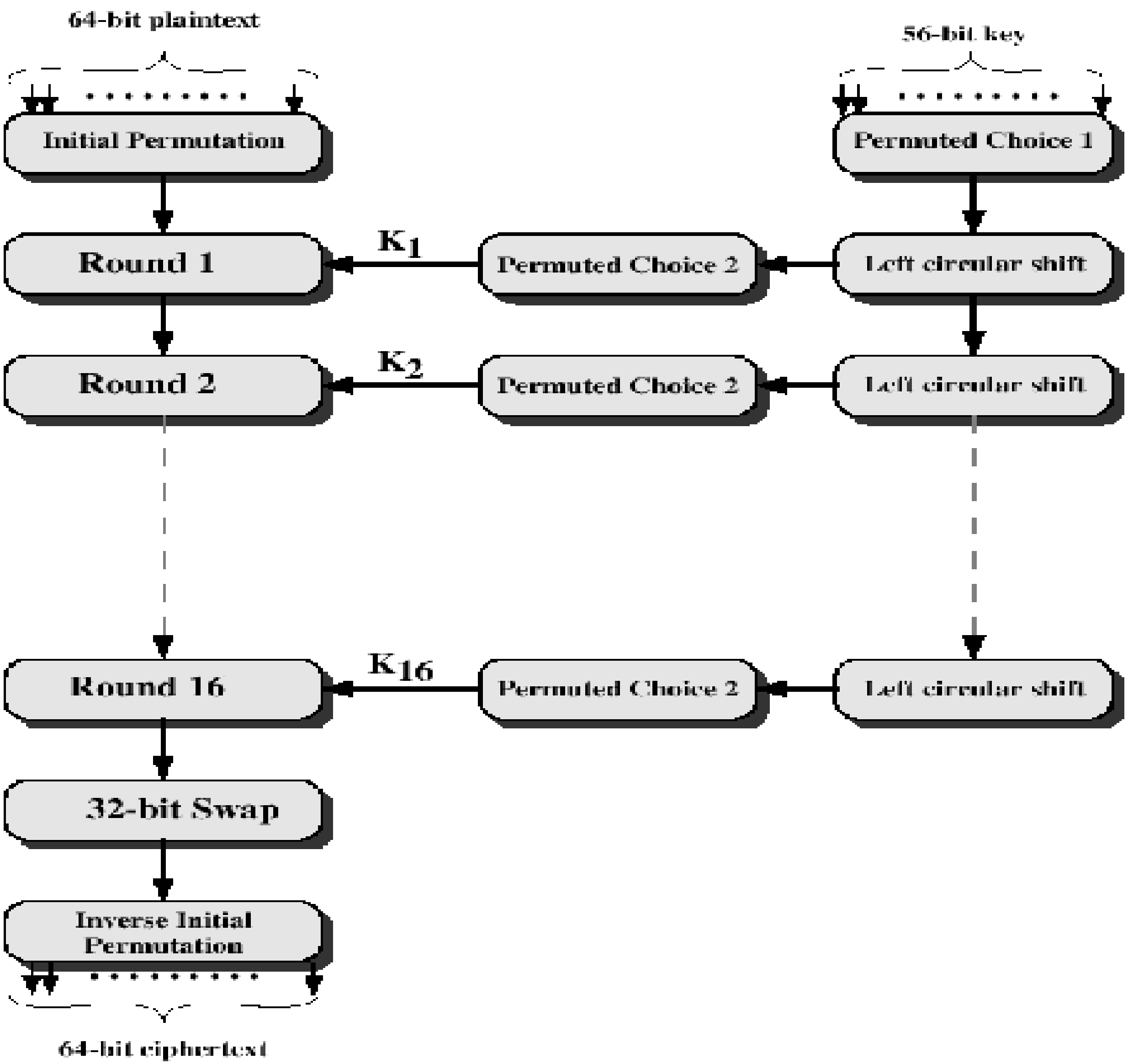


Figure 2.3 General Depiction of DES Encryption Algorithm

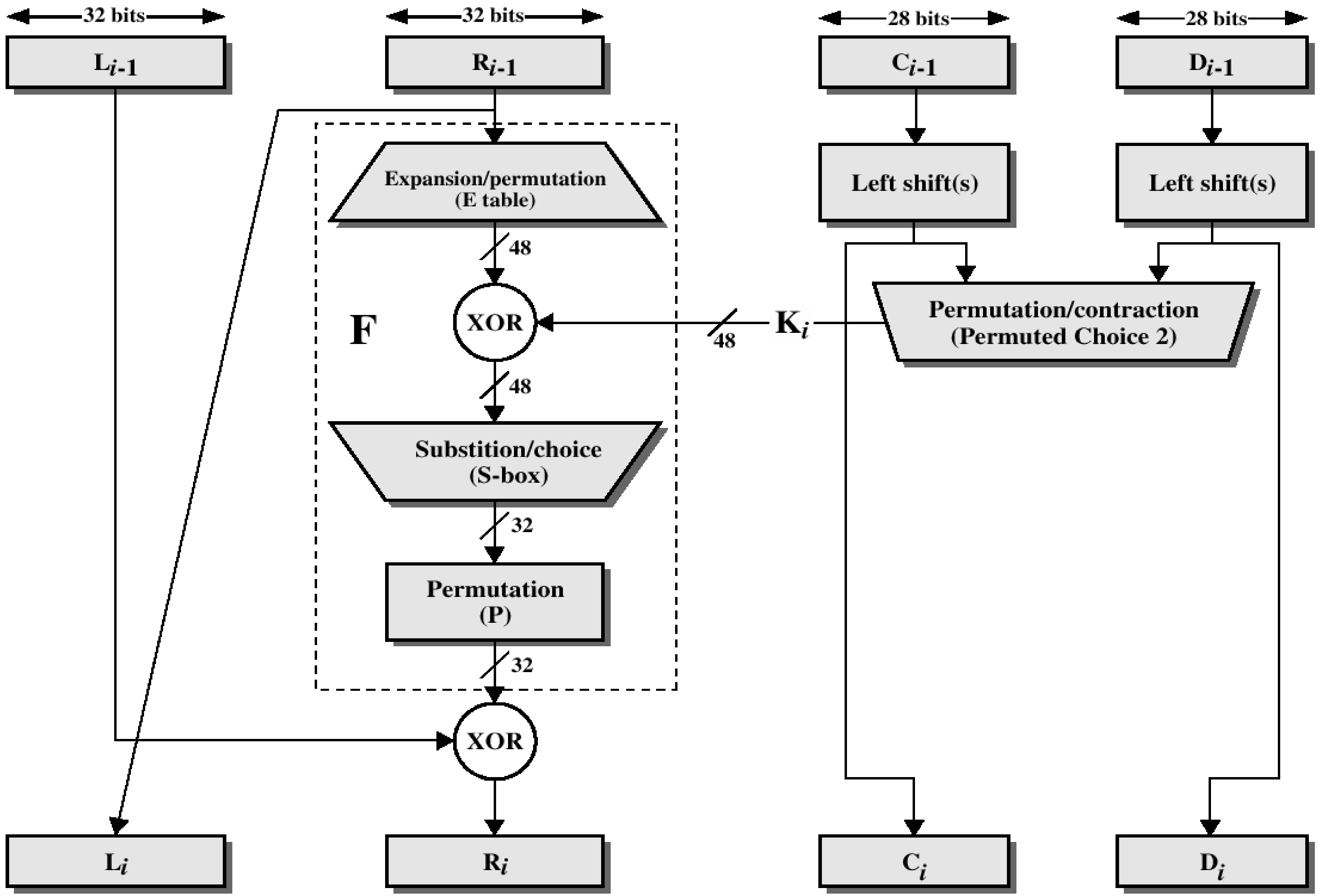
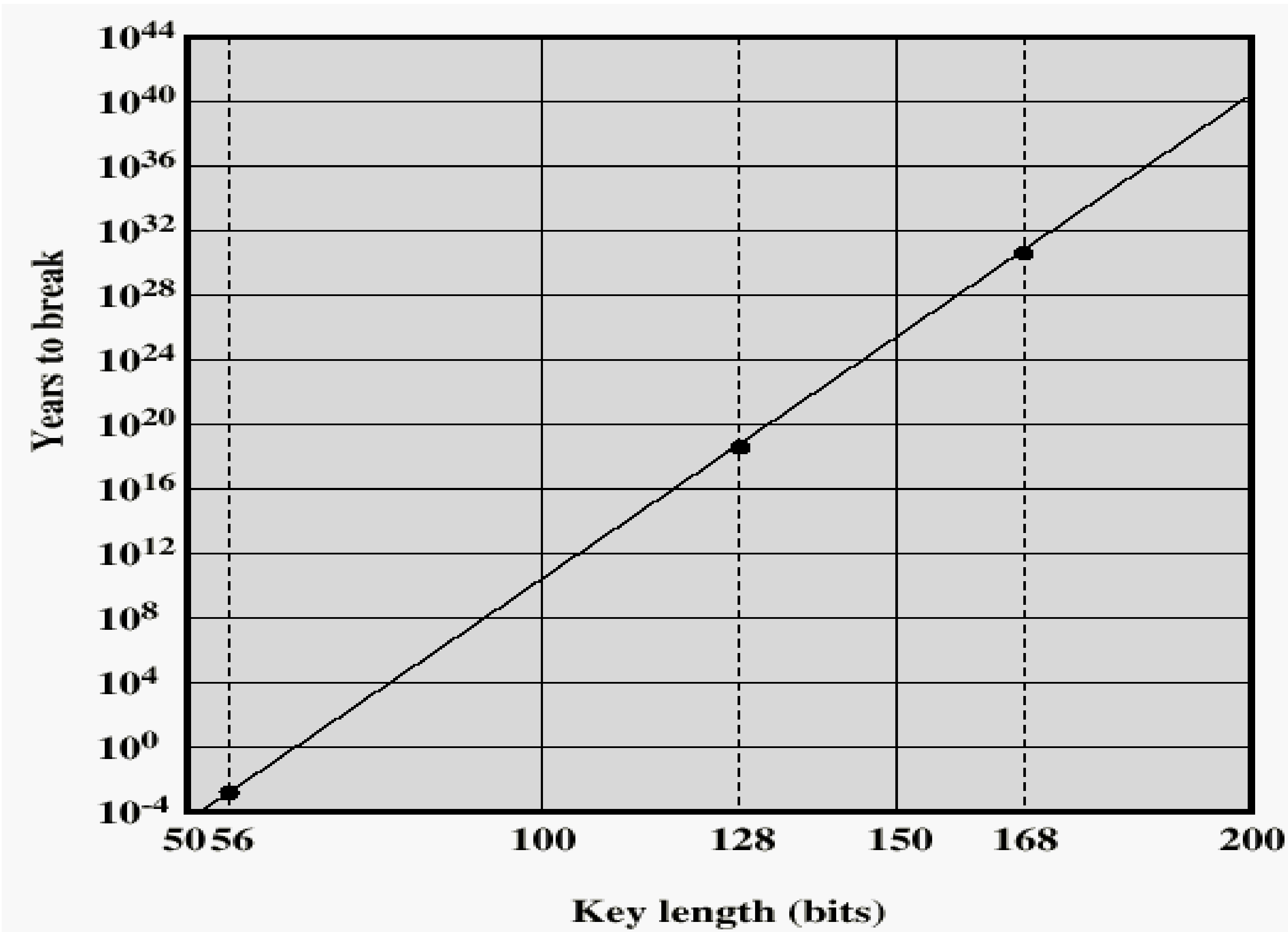


Figure 2.4 Single Round of DES Algorithm

DES

- **The overall processing at each iteration:**
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \quad F(R_{i-1}, K_i)$
- Concerns about:
  - The algorithm (since the design criteria were classified)
  - and the key length (56-bits) vs 128 bits

Time to break a code (10^6 decryptions/µs)



DES Concerns

- Although there are concerns about the DES design- no weakness has yet been discovered.
- With 56 bit keys- brute force is possible as demonstrated by "DES Cracker" in 1998 and machine speeds and costs will continue to improve.
- A 128 bit key is guaranteed to be unbreakable by brute force.

Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- Thirty+ years of intense analysis has revealed no "back door"
- Attacks, essentially exhaustive key search
- **Inescapable conclusions**
  - Designers of DES knew what they were doing
  - Designers of DES were way ahead of their time

Block Cipher Notation

- P = plaintext block
- C = ciphertext block
- Encrypt P with key K to get ciphertext C
  - $C = E(P, K)$
- Decrypt C with key K to get plaintext P
  - $P = D(C, K)$
- Note: $P = D(E(P, K), K)$ and $C = E(D(C, K), K)$
  - But $P \neq D(E(P, K_1), K_2)$ and $C \neq E(D(C, K_1), K_2)$ when $K_1 \neq K_2$

DES Alternatives

- A replacement for DES was needed
- Use multiple encryption with DES implementations – **3DES**
- Design a new alternative- AES is a new cipher alternative
- 3DES
- https://www.youtube.com/watch?v=jQEx_vxLnrE

3DES with 2 Keys

- Use 3 encryptions
- Can use 2 keys with E-D-E sequence
  - $C = E_{K1}(D_{K2}(E_{K1}(P)))$
  - if K1=K2 then can work with single DES
- Standardized in ANSI X9.17 & ISO8732
- No current known practical attacks
  - several proposed impractical attacks might become basis of future attacks
  - Brute force search about 2^{112}

Triple DES

- Use 3 keys and 3 executions of the DES algorithm (encrypt-decrypt-encrypt)

$$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$ encrypt

  - C = ciphertext
  - P = Plaintext
  - EK[X] = encryption of X using key K
  - DK[Y] = decryption of Y using key K

$$P = D_{K1}[E_{K2}[D_{K3}[C]]]$$ decrypt

- Effective key length of 168 bits

Triple DES/DEA

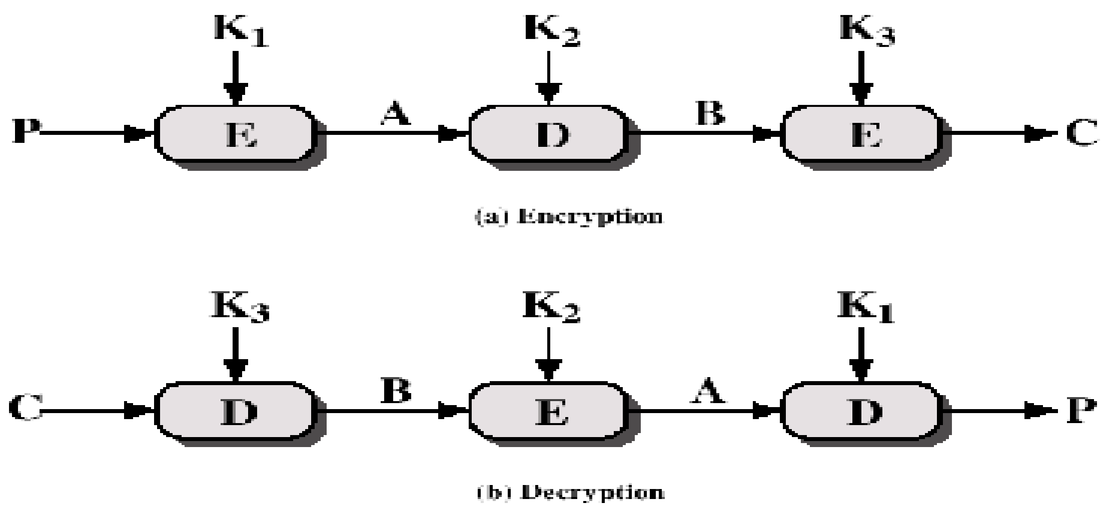


Figure 2.6 Triple DEA

Triple DES

- 3 Key 3DES is the preferred alternative
- Approved for use in financial applications
- Adopted by some Internet applications, (eg. PGP, S/MIME)
- High level of confidence that 3DES is secure and resistent to cryptanalysis.
- Disadvantage – **slow**, small block size

Triple DES

- Today, 56 bit DES key is too small
  - Exhaustive key search is feasible
- But DES is everywhere, so what to do?
- **Triple DES** or **3DES** (112 bit key)
  - $C = E(D(E(P,K_1),K_2),K_1)$
  - $P = D(E(D(C,K_1),K_2),K_1)$
- Why Encrypt-Decrypt-Encrypt with 2 keys?
  - Backward compatible: $E(D(E(P,K),K),K) = E(P,K)$
  - And 112 bits is enough

3DES

- Why not $C = E(E(P,K),K)$?
  - Trick question --- it's still just 56 bit key
- Why not $C = E(E(P,K_1),K_2)$?
- A (semi-practical) **known plaintext** attack
  - Pre-compute table of $E(P,K_1)$ for every possible key K_1 (resulting table has 2^{56} entries)
  - Then for each possible K_2 compute $D(C,K_2)$ until a match in table is found
  - When match is found, have $E(P,K_1) = D(C,K_2)$
  - Result gives us keys: $C = E(E(P,K_1),K_2)$

Advanced Encryption Standard

- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected (pronounced like "Rain Doll" or "Rhine Doll")
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)
- https://www.youtube.com/watch?v=ZhILF5Dhx74

AES Overview

- **Block size:** 128 bits (others in Rijndael)
- **Key length:** 128, 192 or 256 bits (independent of block size)
- 10 to 14 rounds (depends on key length)
- Each round uses 4 functions (3 "layers")
  - ByteSub (nonlinear layer)
  - ShiftRow (linear mixing layer)
  - MixColumn (nonlinear layer)
  - AddRoundKey (key addition layer)

AES ByteSub

- Treat 128 bit block as 4x6 byte array

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ByteSub} \longrightarrow \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}.$$

- ByteSub is AES's "S-box"
- Can be viewed as nonlinear (but invertible) composition of two math operations

AES "S-box"

Last 4 bits of input

First 4 bits of input

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

AES ShiftRow

- Cyclic shift rows

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ShiftRow} \longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

AES MixColumn

- Invertible, linear operation applied to each column

$$\begin{bmatrix} a_{0i} \\ a_{1i} \\ a_{2i} \\ a_{3i} \end{bmatrix} \longrightarrow \texttt{MixColumn} \longrightarrow \begin{bmatrix} b_{0i} \\ b_{1i} \\ b_{2i} \\ b_{3i} \end{bmatrix} \quad \text{for } i = 0, 1, 2, 3$$

- Implemented as a (big) lookup table

AES AddRoundKey

- XOR subkey with block

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Block Subkey

- RoundKey (subkey) determined by **key schedule** algorithm

AES Decryption

To decrypt, process must be invertible

- Inverse of MixAddRoundKey is easy, since "$\oplus$" is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)

AES Cipher

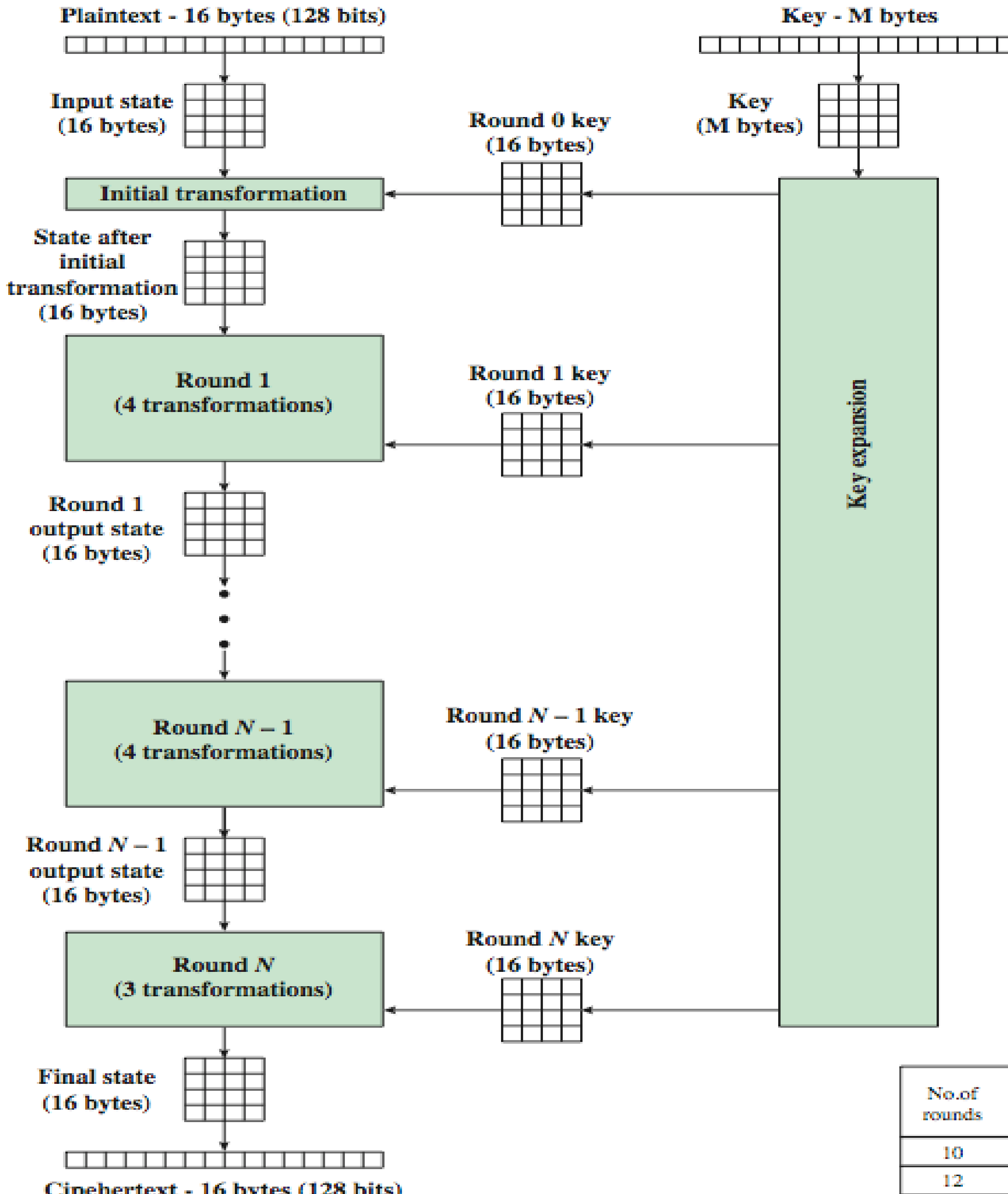
- An **iterative** rather than **feistel** cipher
  - Operates on entire block in every round rather than halves
  - Processes data as block of 4 columns of 4 bytes
- Designed Criteria:
  - Resistant against known attacks
  - Speed and code compactness on many CPUs
  - Design simplicity

AES Cipher Stages

4 Stages are used(1 permutation, 3 substitution):

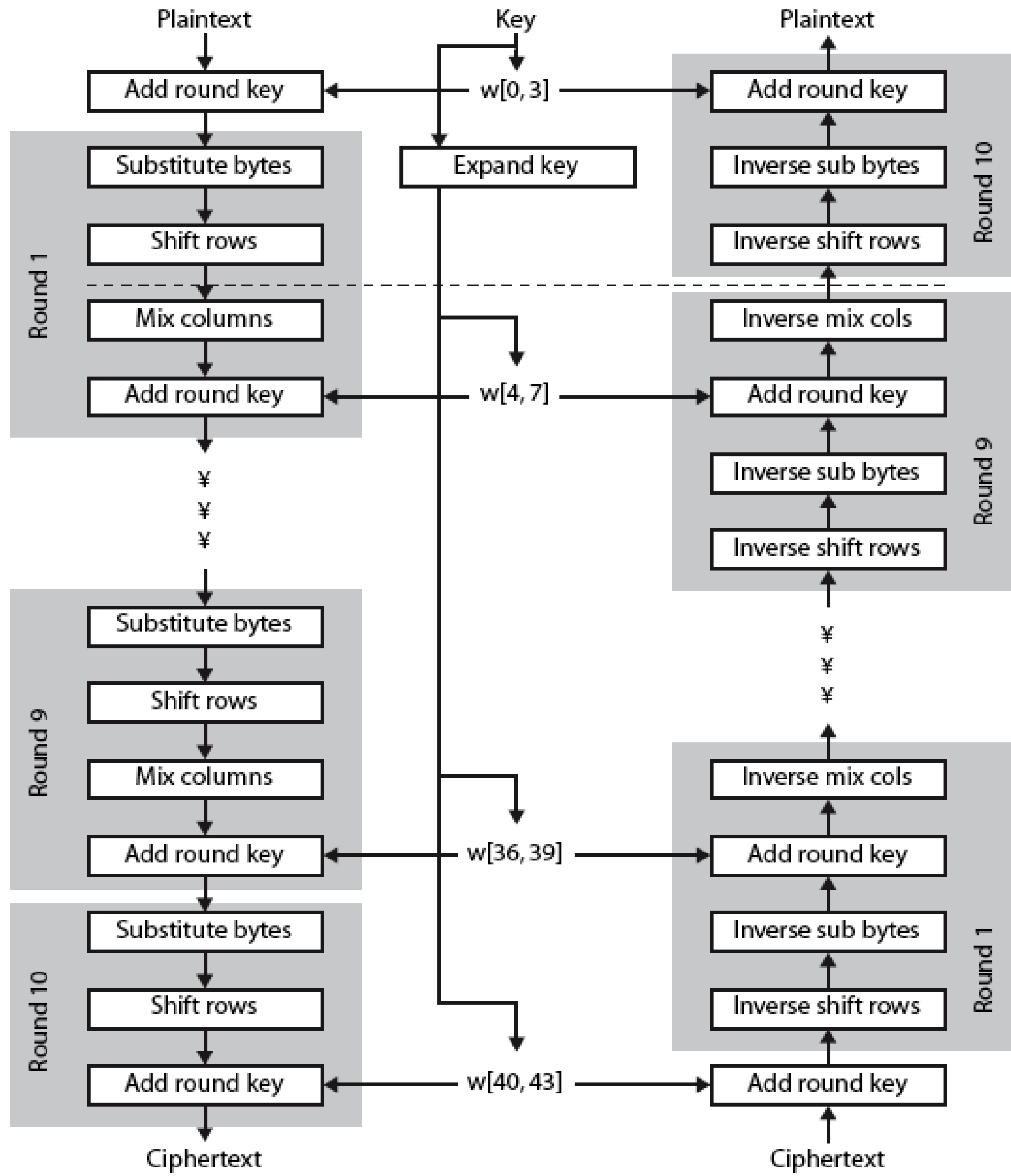
1. Substitute bytes (S-box)
2. Shift rows (row-by-row permutation)
3. Mix columns (substitution using function of all bytes in the column)
4. Add Round Key (bitwise XOR with key)

AES Encryption Process



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

AES Structure



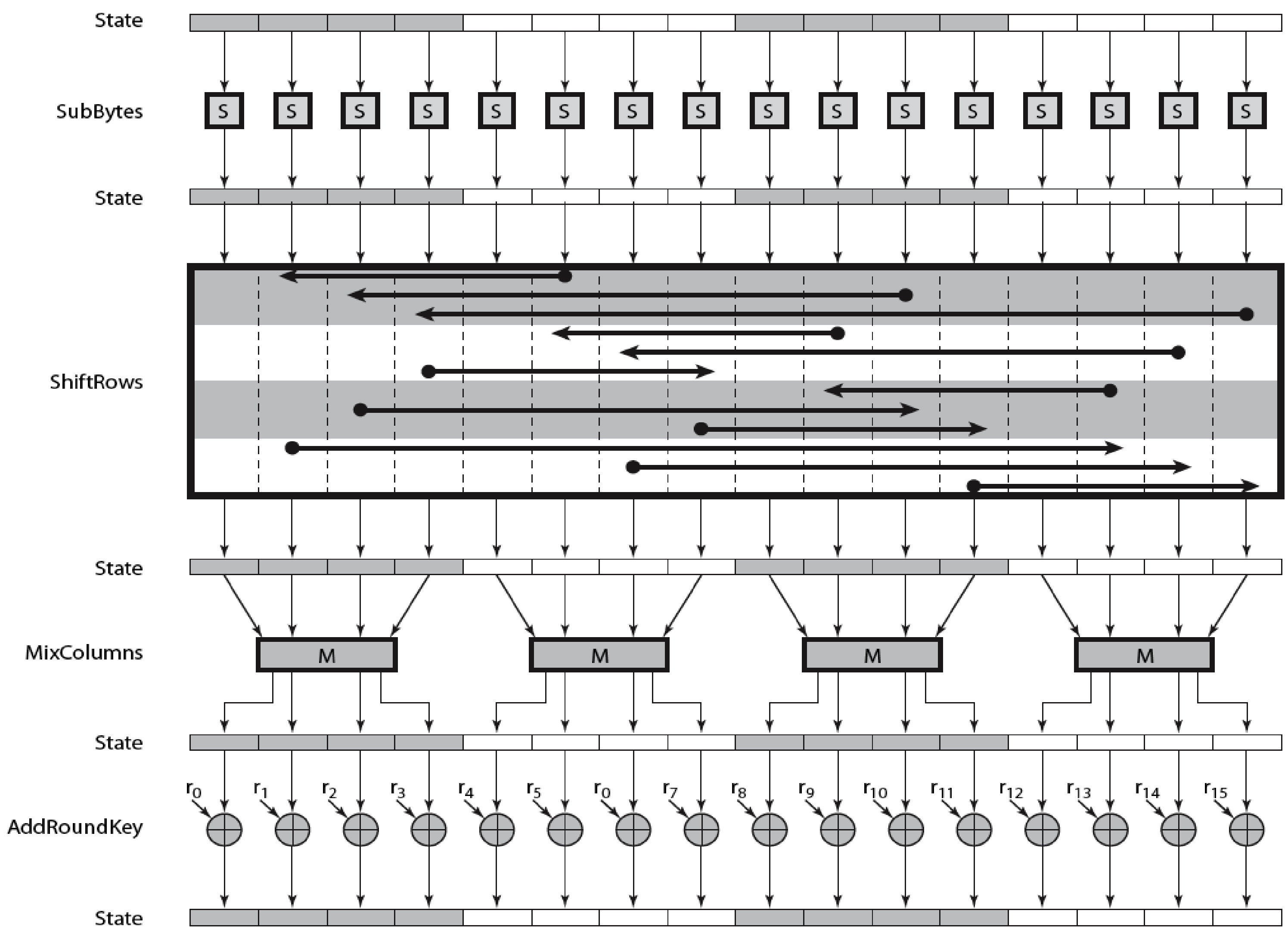
(a) Encryption

(b) Decryption

AES Algorithm

- Input is a single 128 bit block (square matrix)
- Block is copied into the STATE array
- At each stage the STATE array is modified by encryption or decryption
- After the final stage the STATE array is copied to an output matrix.
- The key is also a square matrix of 128 bits

AES Round



State
SubBytes
S S S S S S S S S S S S S S S S
State
ShiftRows
State
MixColumns
M M M M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

A Few Other Block Ciphers

- Briefly…
  - IDEA
  - Blowfish
  - RC6
- More detailed…
  - TEA

Other Symmetric Block Ciphers

- International Data Encryption Algorithm (IDEA)
  - 128-bit key
  - Used in PGP
- Blowfish
  - Easy to implement
  - High execution speed
  - Run in less than 5K of memory
- CAST-128
  - Key size from 40 to 128 bits
  - The round function differs from round to round

IDEA

- Invented by James Massey
  - One of the giants of modern crypto
- IDEA has 64-bit block, 128-bit key
- IDEA uses **mixed-mode arithmetic**
- Combine different math operations
  - IDEA the first to use this approach
  - Frequently used today

Blowfish

- Blowfish encrypts 64-bit blocks
- Key is variable length, up to 448 bits
- Invented by Bruce Schneier
- Almost a Feistel cipher

$$R_i = L_{i-1} \oplus K_i$$
$$L_i = R_{i-1} \oplus F(L_{i-1} \oplus K_i)$$

- The round function F uses 4 S-boxes
  - Each S-box maps 8 bits to 32 bits
- **Key-dependent S-boxes**
  - S-boxes determined by the key

RC6

- Invented by Ron Rivest (of RSA fame)
- Variables
  - Block size
  - Key size
  - Number of rounds
- An AES finalist
- Uses **data dependent rotations**
  - Unusual for algorithm to depend on plaintext

Time for TEA

- Tiny Encryption Algorithm (TEA)
- 64 bit block, 128 bit key
- Assumes 32-bit arithmetic
- Number of rounds is variable (32 is considered secure)
- Uses "weak" round function, so large number of rounds required

TEA Encryption

Assuming 32 rounds:

```
(K[0],K[1],K[2],K[3]) = 128 bit key
(L,R) = plaintext (64-bit block)
delta = 0x9e3779b9
sum = 0
for i = 1 to 32
    sum += delta
    L += ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
    R += ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
next i
ciphertext = (L,R)
```

TEA Decryption

Assuming 32 rounds:

```
(K[0],K[1],K[2],K[3]) = 128 bit key
(L,R) = ciphertext (64-bit block)
delta = 0x9e3779b9
sum = delta << 5
for i = 1 to 32
    R –= ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
    L –= ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
    sum –= delta
next i
plaintext = (L,R)
```

TEA Comments

- **Almost** a Feistel cipher
  - Uses + and - instead of ⊕ (XOR)
- Simple, easy to implement, fast, low memory requirement, etc.
- Possibly a "related key" attack
- eXtended TEA (XTEA) eliminates related key attack (slightly more complex)
- Simplified TEA (STEA) — insecure version used as an example for cryptanalysis

Block Cipher Modes

Multiple Blocks

- How to encrypt multiple blocks?
- Do we need a new key for each block?
  - As bad as (or worse than) a one-time pad!
- Encrypt each block independently?
- Make encryption depend on previous block?
  - That is, can we “chain” the blocks together?
- How to handle partial blocks?
  - We won’t discuss this issue

Modes of Operation

- The different ways an encryption algorithm can be used are modes of operation
- NIST SP 800-38A defines 5 modes:

1. Electronic codebook (ECB) mode
2. Cipher Block Chaining (CBC) mode
3. Cipher Feedback (CFB) mode
4. Output Feedback (OFB) mode
5. Counter (CTR) mode

See also http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Modes of Operation

- In cryptography, **modes of operation** is the procedure of enabling the repeated and secure use of a block cipher under a single key.
- A block cipher by itself allows encryption only of a single data block of the cipher's block length.
- A mode of operation describes the process of encrypting each of these blocks, and generally uses randomization.
- http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Modes of Operation

- Many modes — we discuss 3 most popular
- Electronic Codebook (**ECB**) mode
  - Encrypt each block independently
  - Most obvious, but has a serious weakness
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
  - More secure than ECB, virtually no extra work
- Counter Mode (**CTR**) mode
  - Block ciphers acts like a stream cipher
  - Popular for random access

Modes of Operation

- Block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
  - AES uses 128 bit blocks
  - For larger sizes, break plain text into blocks
- Need some way to en/decrypt arbitrary amounts of data in practice
- have **block** and **stream** modes
- Cover a wide variety of applications
- Can be used with any block cipher

Electronic Codebook Mode (ECB)

- Message is broken into independent blocks which are encrypted
- Each block is a value which is substituted, like a codebook, hence name
- Each block B is encoded or decoded independently of the other blocks:

  $C_i = E_K(P_i)$ $\qquad\qquad$ $B_i = D_K(C_i)$

- Uses: secure transmission of single values

https://www.youtube.com/watch?v=qTjW_6q_61g

Electronic Codebook Mode

- Codebook- for a given key there is a unique cipher text for every b-bit block of plaintext.
- Advantages:
  - Simplicity
  - Tolerates block loss (eg. over network)
  - Used to send a few block of data
- Disadvantage:
  - ECB mode may reveal pattern in text, i.e. blocks that are identical, will be encrypted in the same way

ECB Mode

- Notation: $C = E(P,K)$
- Given plaintext $P_0, P_1, \ldots, P_m, \ldots$
- Most obvious way to use a block cipher:

Encrypt	**Decrypt**
$C_0 = E(P_0, K)$	$P_0 = D(C_0, K)$
$C_1 = E(P_1, K)$	$P_1 = D(C_1, K)$
$C_2 = E(P_2, K)$ …	$P_2 = D(C_2, K)$ …

- For fixed key K, this is "electronic" version of a codebook cipher (without additive)
  - With a different codebook for each key

ECB Cut and Paste

- Suppose plaintext is

    `Alice digs Bob. Trudy digs Tom.`

- Assuming 64-bit blocks and 8-bit ASCII:

    P_0 = "`Alice di`", P_1 = "`gs Bob. `",

    P_2 = "`Trudy di`", P_3 = "`gs Tom. `"

- Ciphertext: C_0, C_1, C_2, C_3
- Trudy cuts and pastes: C_0, C_3, C_2, C_1
- Decrypts as

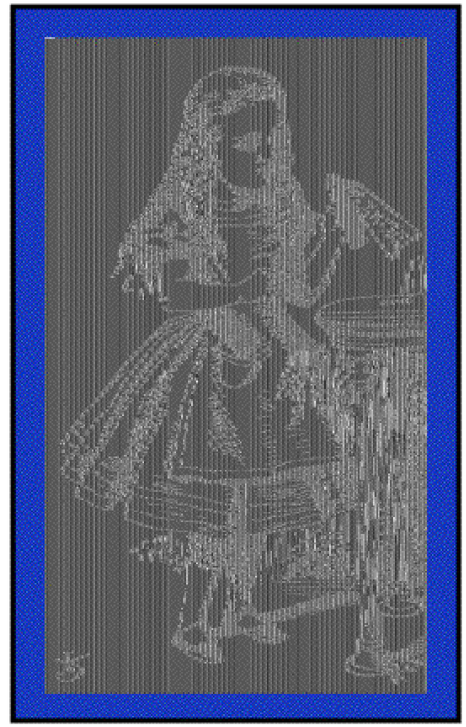
    `Alice digs Tom. Trudy digs Bob`

ECB Weakness

- Suppose $P_i = P_j$
- Then $C_i = C_j$ and Trudy knows $P_i = P_j$
- This gives Trudy some information, even if she does not know P_i or P_j
- Trudy might know P_i
- Is this a serious issue?

Alice Hates ECB Mode

Alice's uncompressed image, and ECB encrypted (TEA)



- Why does this happen?
- Same plaintext yields same ciphertext!

Cipher Block Chaining Mode of Operation

- Message is broken into blocks
- Linked together in encryption operation
- Each previous cipher block is chained with current plaintext block, hence name
- Use Initial Vector (IV) to start process
- Input to encryption algorithm bears no relationship to plaintext block
- Uses: bulk data encryption, authentication

Cipher Block Chaining Mode of Operation

- Cipher Block Chaining Mode (CBC)
  - The 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

$$C_i = E_k[C_{i-1} \oplus P_i]$$

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

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

$$C_{i-1} \oplus D_K[C_i] = C_{i-1} \oplus C_{i-1} \oplus P_i = P_i$$

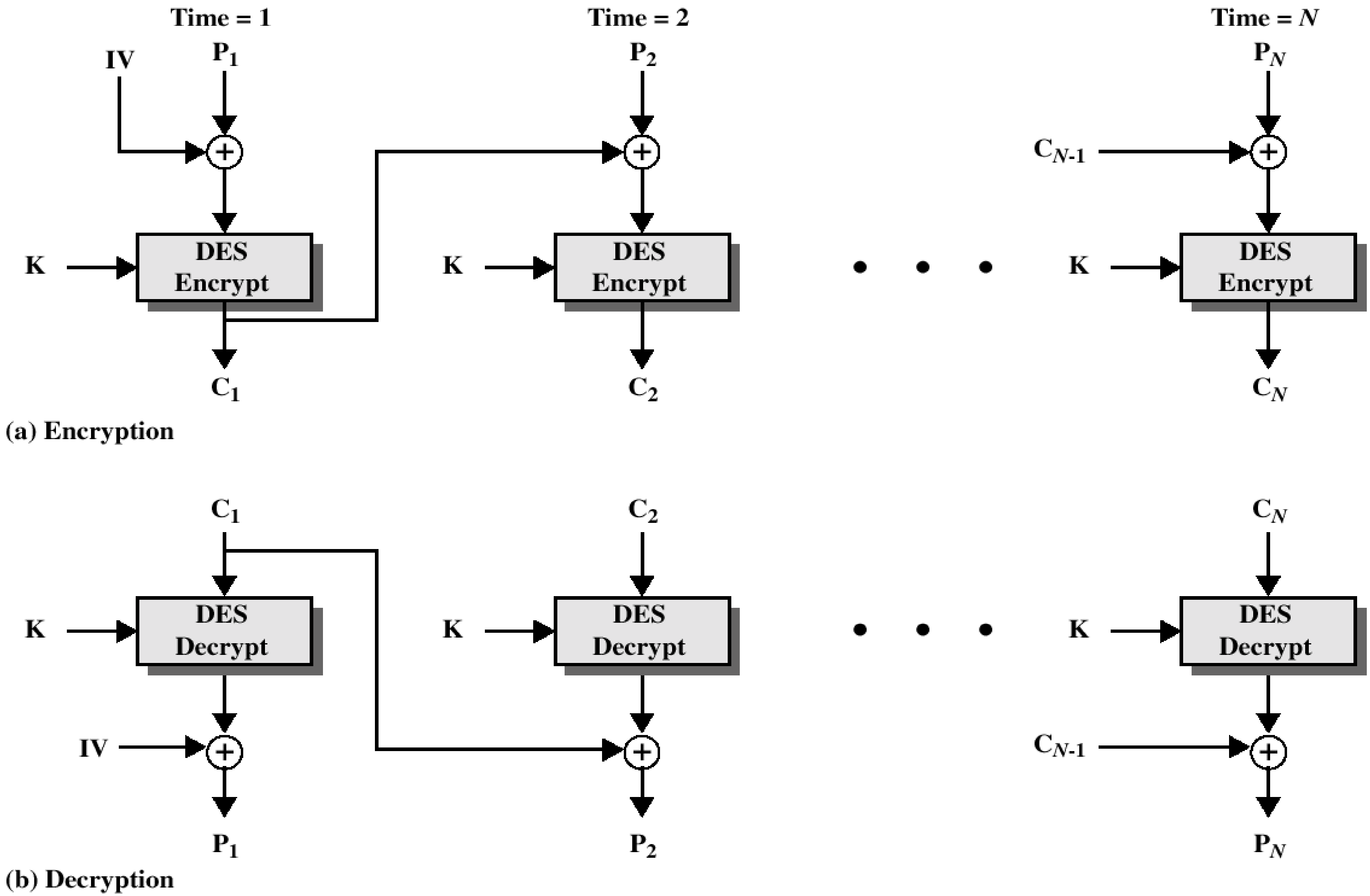


Figure 2.7 Cipher Block Chaining (CBC) Mode

CBC Mode

- Blocks are "chained" together
- A random initialization vector, or IV, is required to initialize CBC mode
- IV is random, but not secret

Encryption

$C_0 = E(IV \oplus P_0, K),$
$C_1 = E(C_0 \oplus P_1, K),$
$C_2 = E(C_1 \oplus P_2, K),\ldots$

Decryption

$P_0 = IV \oplus D(C_0, K),$
$P_1 = C_0 \oplus D(C_1, K),$
$P_2 = C_1 \oplus D(C_2, K),\ldots$

- Analogous to classic codebook *with additive*

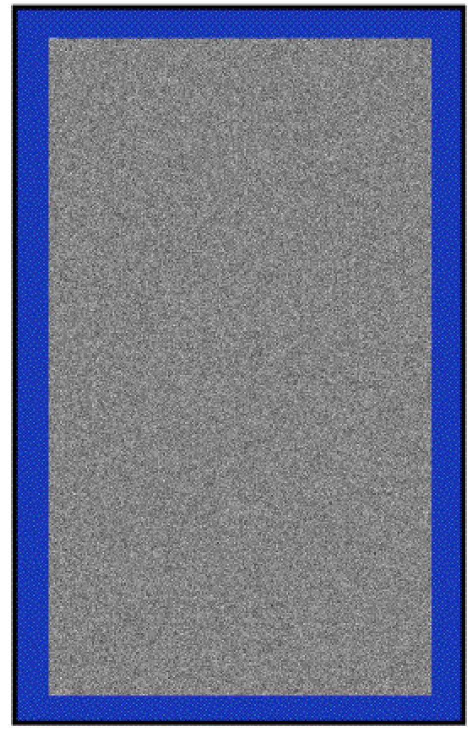
CBC Mode

- Identical plaintext blocks yield different ciphertext blocks — this is good!
- If C_1 is garbled to, say, G then

  $P_1 \neq C_0 \oplus D(G, K)$, $P_2 \neq G \oplus D(C_2, K)$
- But $P_3 = C_2 \oplus D(C_3, K)$, $P_4 = C_3 \oplus D(C_4, K)$,…
- Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

Alice Likes CBC Mode

Alice's uncompressed image, Alice CBC encrypted (TEA)



- Why does this happen?
- Same plaintext yields different ciphertext!

Counter (CTR)

- A "new" mode, though proposed earlier
- Similar to Outback Feedback (OFB) but encrypts a counter value rather than any feedback value
- Must have a different key & counter value for every plaintext block (never reused)

  $O_i = E_K(i)$

  $C_i = P_i \text{ XOR } O_i$
- uses: high-speed network encryptions

Advantages and Limitations of CTR

- Efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- Random access to encrypted data blocks
- Provable security (good as other modes)
- Must ensure never reuse key/counter values, otherwise could break, like OFB

Counter Mode (CTR)

- CTR is popular for random access
- Use block cipher like a stream cipher

Encryption

$C_0 = P_0 \oplus E(IV, K),$

$C_1 = P_1 \oplus E(IV+1, K),$

$C_2 = P_2 \oplus E(IV+2, K),\ldots$

Decryption

$P_0 = C_0 \oplus E(IV, K),$

$P_1 = C_1 \oplus E(IV+1, K),$

$P_2 = C_2 \oplus E(IV+2, K),\ldots$

- CBC can also be used for random access
  - With a significant limitation…

Integrity

Data Integrity

- **Integrity** — detect unauthorized writing (i.e., modification of data)
- Example: Inter-bank fund transfers
  - Confidentiality may be nice, integrity is critical
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc.

MAC

- **Message Authentication Code** (MAC)
  - Used for data **integrity**
  - Integrity **not** the same as confidentiality
- MAC is computed as **CBC residue**
  - That is, compute CBC encryption, saving only final cipher text block, the MAC

MAC Computation

- MAC computation (assuming N blocks)

  $C_0 = E(IV \oplus P_0, K)$,

  $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$,…

  $C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = \text{MAC}$

- MAC sent with IV and plaintext
- Receiver does same computation and verifies that result agrees with MAC
- Note: receiver must know the key K

Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) =$ **MAC**
- Alice sends IV, P_0, P_1, P_2, P_3 and **MAC** to Bob
- Suppose Trudy changes P_1 to X
- Bob computes

  $C_0 = E(IV \oplus P_0, K)$, $\mathit{C_1} = E(C_0 \oplus X, K)$,

  $\mathit{C_2} = E(\mathit{C_1} \oplus P_2, K)$, $\mathit{C_3} = E(\mathit{C_2} \oplus P_3, K) =$ ***MAC*** $\neq$ **MAC**
- That is, error <u>propagates</u> into **MAC**
- Trudy can't make ***MAC*** == **MAC** without K

Confidentiality and Integrity

- Encrypt with one key, MAC with another key
- Why not use the same key?
  - Send last encrypted block (MAC) twice?
  - This cannot add any security!
- Using different keys to encrypt and compute MAC works, even if keys are related
  - But, twice as much work as encryption alone
  - Can do a little better — about 1.5 "encryptions"
- Confidentiality and integrity with same work as one encryption is a research topic

Uses for Symmetric Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later…)
- Anything you can do with a hash function (upcoming chapter…)

Recommended Reading

- Stallings, W. *Cryptography and Network Security: Principles and Practice*, 5th edition. Prentice Hall, 2011
- Scneier, B. *Applied Cryptography*, New York: Wiley, 1996
- Mel, H.X. Baker, D. *Cryptography Decrypted*. Addison Wesley, 2001
- Simon Singh, The Code Book, (on-line)
- http://simonsingh.net/books/the-code-book/the-book/
- David, Kahn, The Codebreakers, 1996.

Videos (Stamp)

- Crypto Basics -Mark Stamp- Info Security
- http://www.youtube.com/watch?v=gnhTDFEQK8A&feature=related
- Double Transposition – One Time Pad
- http://www.youtube.com/watch?v=_8SQljT_g9w&feature=related
- 3-1 Symmetric Key – Stream And Block Ciphers
- http://www.youtube.com/watch?v=1GoP_HfF_v4&feature=related
- 3-2 Symmetric Key – Stream Ciphers- RC4
- http://www.youtube.com/watch?v=riIp6EQOJOg&feature=related
- *3-3 Symmetric Key – Block Ciphers, Feistel cipher
- http://www.youtube.com/watch?v=ySZvE9vOfEQ&feature=related
- 3-4 Symmetric key = Block Ciphers, DES
- https://www.youtube.com/watch?v=G_guTnTcoqg
- 3-5 Symmetric Key -Block ciphers – DES, 3DES
- http://www.youtube.com/watch?v=jQEx_vxLnrE&feature=related

Videos (Stamp)

- 3-6 Symmetric Key –block, AES
- https://www.youtube.com/watch?v=ZhILF5Dhx74
- 3-7 Information Security: Principles and Practices
- https://www.youtube.com/watch?v=aR29pnuJ6fQ
- 3-8 Symmetric Key Crypto- block cipher modes, ECB
- https://www.youtube.com/watch?v=qTjW_6q_61g
- 3-9 Symmetric Key Crypto- block cipher, CBC, CTR
- https://www.youtube.com/watch?v=057hz60lhZw
- 3-10 Symmetric Key Crypto- integrity, message authentication code, MAC
- https://www.youtube.com/watch?v=8XcFiMju_94

More Video Resources

- http://www.FreeSecurityPlus.com
- One time pad demo
- https://www.youtube.com/watch?v=3uJl2zutyO4
- Block and Stream Ciphers
- https://www.youtube.com/watch?v=E_3M41NrtsU

More Resources

- RSA Laboratories http://www.rsa.com/rsalabs/
- http://www.rsa.com/rsalabs/node.asp?id=2174 Stream ciphers
- Search Security – Tutorials and Information
- http://searchsecurity.techtarget.com/tutorial/Information-security-tutorials
- http://searchsecurity.techtarget.com/definition/block-cipher
- Stamp
- http://cs.sjsu.edu/~stamp/crypto/PowerPoint_PDF/5_StreamCiphers.pdf
- Stream vs Block
- http://people.seas.harvard.edu/~salil/cs120/docs/lec13.pdf
- Digital signatures
- https://www.youtube.com/watch?v=HubAvQg6SPM