Cryptosystems and Symmetric Encryption/Decryption)

- Need for improved Security

Some attack types on Network

- **1. Disclosure (İfşaat)**
- **2. Traffic Analysis(Trafik Analizi)**
- **3. Masquerade (Gerçeği gizleme)**
- **4. Content Modification (İçerik Değiştirme**)
- **5. Sequence Modification (Sıra Değiştirme)**
- **6. Timing Modification (Zamanlamayı Değiştirme)**
- **7. Repudiation (İnkarcılık)**

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM
11	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		74	326 ms	8:58:39 PM
12	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		91	920 µs	8:58:39 PM
13	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	172 ms	8:58:39 PM

```
 0: 00 00 E8 2F 77 2A 00 20 AF 24 7F 25 08 00 45 00 |.../w*. .$.%..E.
10: 00 3B 1C 00 40 00 20 06 BA CC C0 A8 01 64 C0 A8 |.;..@. ......d..
20: 01 3C 00 6E 04 2B 00 0D 0D 56 00 BF 06 DF 50 18 |.<.n.+...V....P.
30: 22 2B D3 06 00 00 2B 4F 4B 20 75 73 65 72 20 61 |"+....+OK user a
40: 63 63 65 70 74 65 64 0D 0A                      |ccepted..
```

- MSG 3,4,5   3 way hanshake
- 6 ,7 POP3 Mail server msg
- 8 Client’s Logon name

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM
11	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		74	326 ms	8:58:39 PM
12	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		91	920 µs	8:58:39 PM
13	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	172 ms	8:58:39 PM

```
 0: 00 00 E8 2F 77 2A 00 20 AF 24 7F 25 08 00 45 00 |.../w*. .$.%..E.
10: 00 3B 1C 00 40 00 20 06 BA CC C0 A8 01 64 C0 A8 |.;..@. ......d..
20: 01 3C 00 6E 04 2B 00 0D 0D 56 00 BF 06 DF 50 18 |.<.n.+...V....P.
30: 22 2B D3 06 00 00 2B 4F 4B 20 75 73 65 72 20 61 |"+....+OK user a
40: 63 63 65 70 74 65 64 0D 0A                      |ccepted..
```

- MSG 9   logon name
- 11 POP3 Mail client msg
- 12 response of server

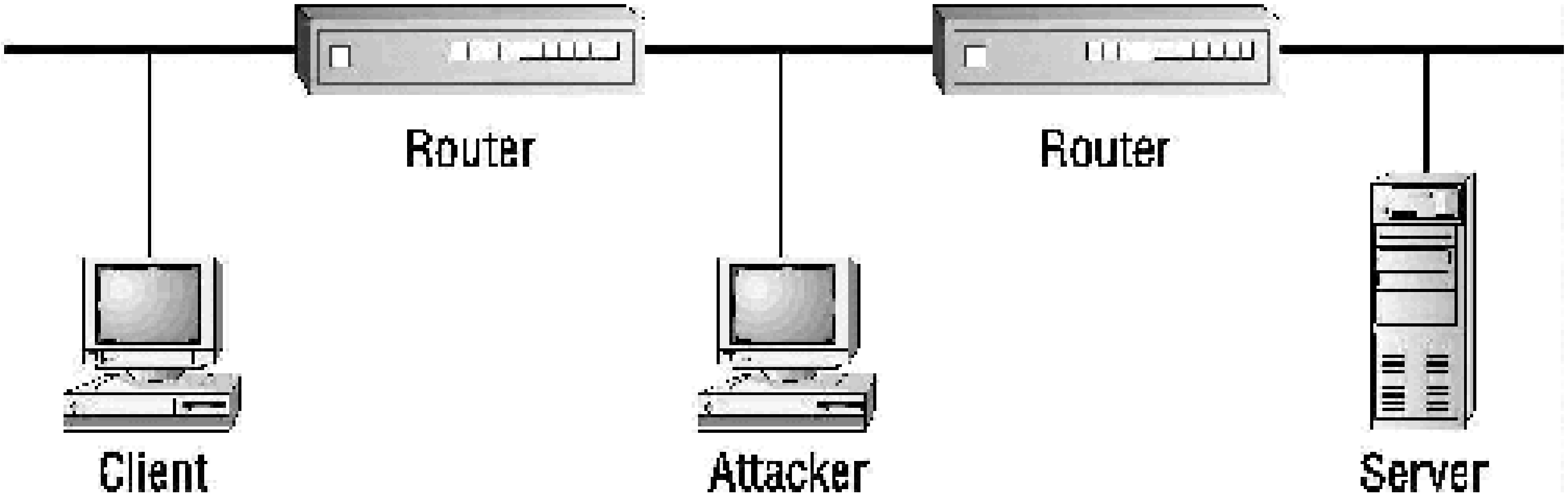
Passive monitoring clear text

Clear text protocols

- **FTP** Authentication is clear text.
- **Telnet** Authentication is clear text.
- **SMTP** Contents of mail messages are delivered as clear text.
- **HTTP** Page content and the contents of fields within forms are sent clear text.
- **IMAP** Authentication is clear text.
- **SNMPv1** Authentication is clear text.

Good authentication required

- Session hijacking



- Verifying destination
- C2MYAZZ (server spoofing Attack)
- DNS Poisoning

Encryption Techniques

Many savages at the present day regard their names as vital parts of themselves, and therefore take great pains to conceal their real names, lest these should give to evil-disposed persons a handle by which to injure their owners.

—*The Golden Bough,* Sir James George Frazer

- For Encryped communication;
  - Encryption Algorithm(E)
  - Decryiption Algorithm (D)
  - Key(K),

Some Basic Terminology

- **plaintext** - original message
- **ciphertext** - coded message
- **cipher** - algorithm for transforming plaintext to ciphertext
- **key** - info used in cipher known only to sender/receiver
- **encipher (encrypt)** - converting plaintext to ciphertext
- **decipher (decrypt)** - recovering ciphertext from plaintext
- **cryptography** - study of encryption principles/methods
- **cryptanalysis (codebreaking)** - study of principles/ methods of deciphering ciphertext *without* knowing key
- **cryptology** - field of both cryptography and cryptanalysis

Requirements

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

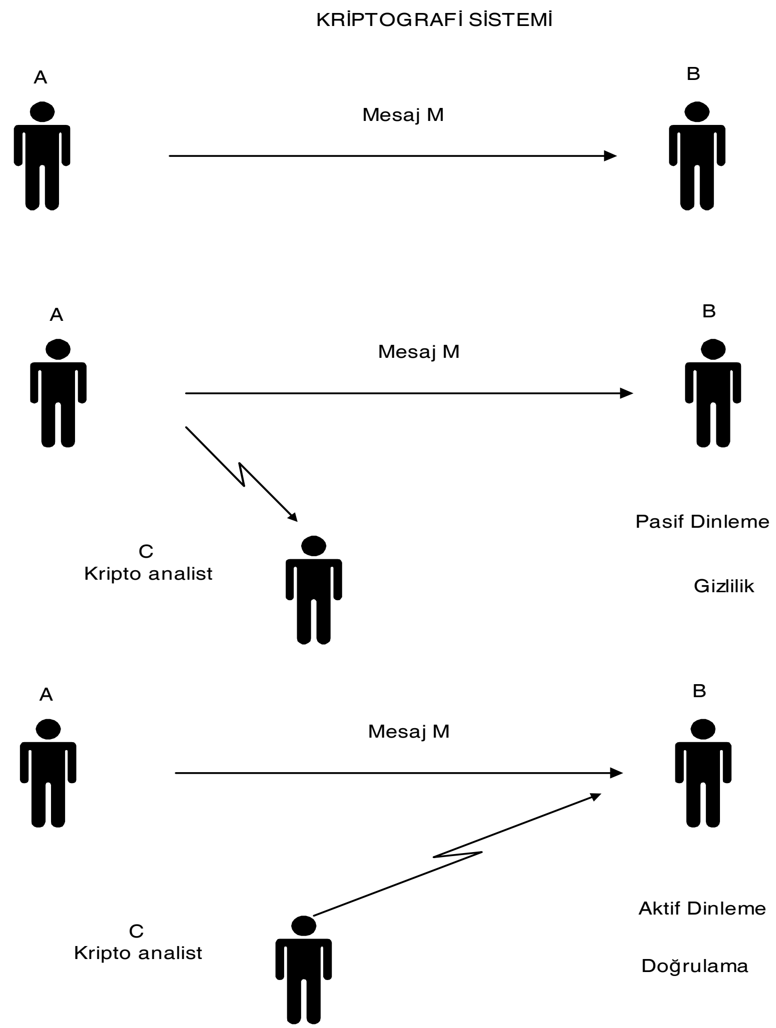
  $c = \mathrm{E}_K(m)$ one to one function

  $m = \mathrm{D}_K(c)$ decryption function
- assume encryption algorithm is known
- implies a secure channel to distribute key

Cryptography

- characterize cryptographic system by:
  - type of encryption operations used
    - substitution / transposition / product
  - number of keys used
    - single-key or private / two-key or public
  - way in which plaintext is processed
    - block / stream

Cryptosystem



Cryptosystem

- Alphabet A
- Plain text space P
- Ciphertext space C
- Key space K
- Encryption Func. E
- Decryption Func. D
- A Cryptosystem is formed as (P,C,K,E,D)
- $for \ \forall\ k \in K$, $D_k \in D$ there is a n$E_k \in E$ functions, such as;
- $\forall\ E_k : P \rightarrow C$ and $\forall\ D_k : C \rightarrow P$ and $D_k(E_k(x)) = x$ for $\forall\ x \in P$

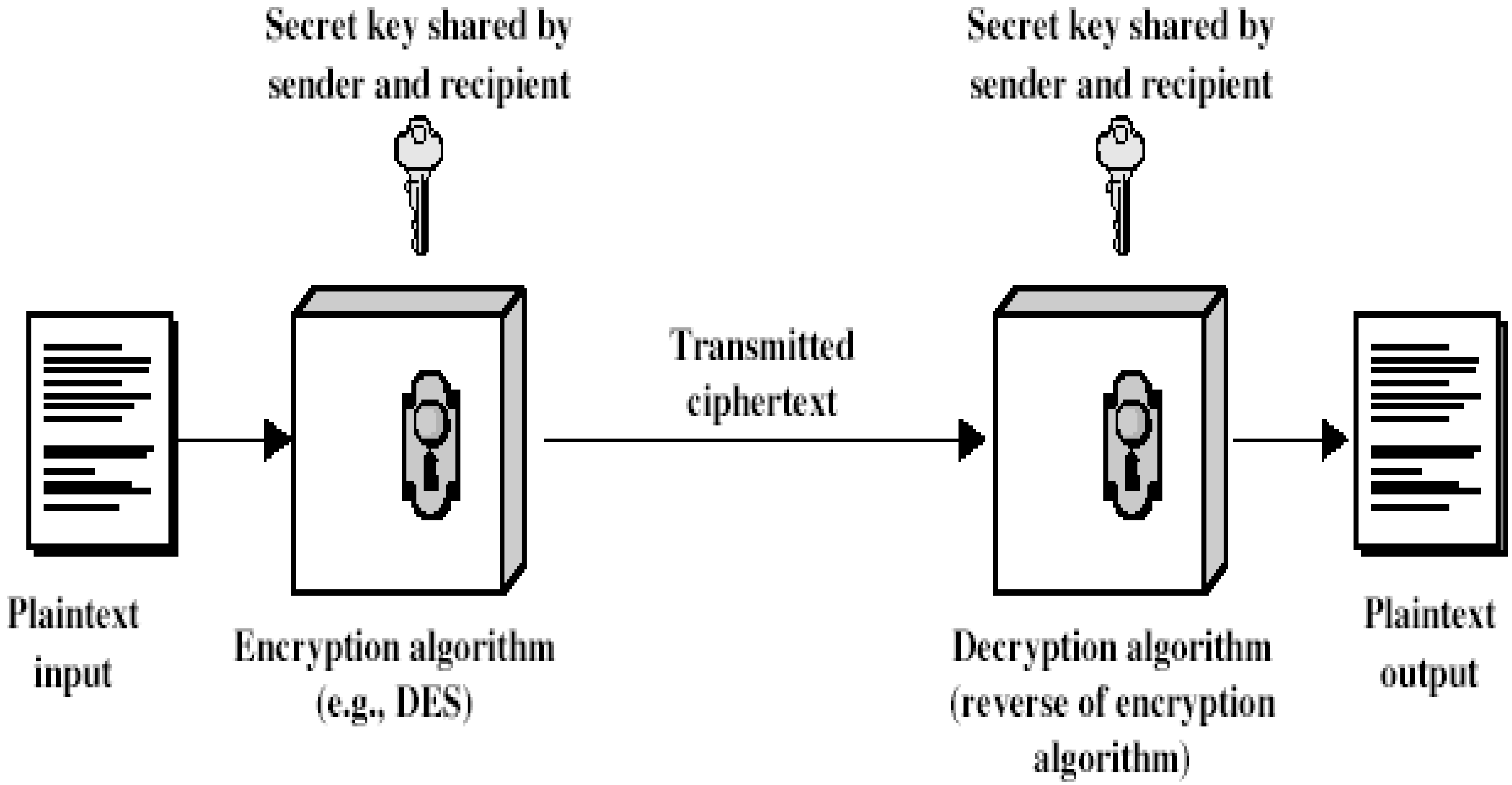
Cryptosystem

- is classified according to;
- Types of operations used for transforming from plaintext to ciphertext
  - Substitution, transposition
- Number of used keys
  - Symmetric, asymmetric
- Processing method of plaintext
  - Block cipher, stream cipher

Symmetric Encryption

- or conventional / private-key / single-key
- sender and recipient share a common key
- all classical encryption algorithms are private-key
- was only type prior to invention of public-key in 1970's
- and by far most widely used

Symmetric Cipher Model



Cryptanalysis

- objective to recover key not just message
- general approaches:
  - cryptanalytic attack
  - brute-force attack

Cryptanalytic Attacks

- **ciphertext only**
  - only know algorithm & ciphertext, is statistical, know or can identify plaintext
- **known plaintext**
  - know/suspect plaintext & ciphertext
- **chosen plaintext**
  - select plaintext and obtain ciphertext
- **chosen ciphertext**
  - select ciphertext and obtain plaintext
- **chosen text**
  - select plaintext or ciphertext to en/decrypt

More Definitions

- **unconditional security**
  - no matter how much computer power or time is available, the cipher cannot be broken since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext
- **computational security**
  - given limited computing resources (eg time needed for calculations is greater than age of universe), the cipher cannot be broken

Brute Force Search

- always possible to simply try every key
- most basic attack, proportional to key size
- assume either know / recognise plaintext

Key length(bit)	Number of keys	Required time in speed of 1 analysis/μs	Required time in speed of 10^6analsis/μs
24	$2^{24} = 1.6 \times 10^7$	2^{23} μs = 8.4 sec	8.4 μsec.
32	$2^{32} = 4.3 \times 10^9$	2^{31} μs = 35.8 min	2.15 milisec.
48	$2^{48} = 2.8 \times 10^{14}$	2^{47} μs = 4.46 years	2.35 min.
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4x 10^{24} years	5.4x 10^{18} yearsl
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} μs = 5.9x 10^{36} years	5.9x 10^{30} years
26 character permutation	$26! = 4 \times 10^{26}$	$2\text{x}10^{26}$ μs = 6.4x 10^{12} y	6.4x 10^6 years

History of Cryptography

- Used 4000 years ago
- Egypts used pictures for cipher

Hieroglyphic encipherments of proper names and titles, with cipher hieroglyphs at left, plain equivalents at right

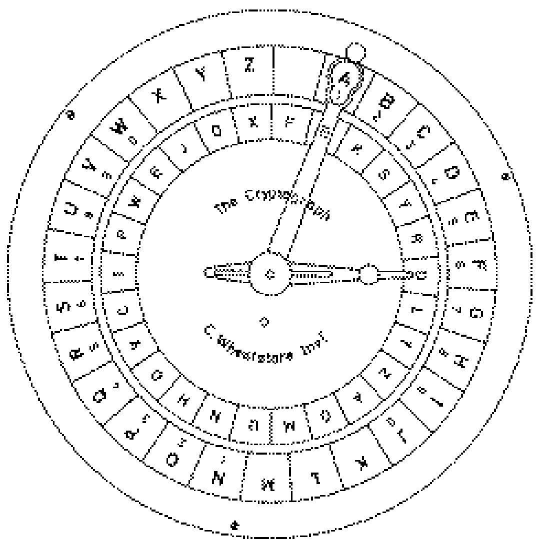
History of Cryptography

- Hebrews used encrypted words in holy books
- Julius Caesar used a basic substitution cipher nearly 2000 years ago
- Roger Bacon presenteed some methods in 1200.
- Geoffrey Chaucer used cipher in his works
- Leon Alberti used a cipher wheel in 1460
- Blaise de Vigenère published a book on cryptography in 1855. (multiple alphabet exchanges)
- Cyrptography is used for mostly military and diplomacy

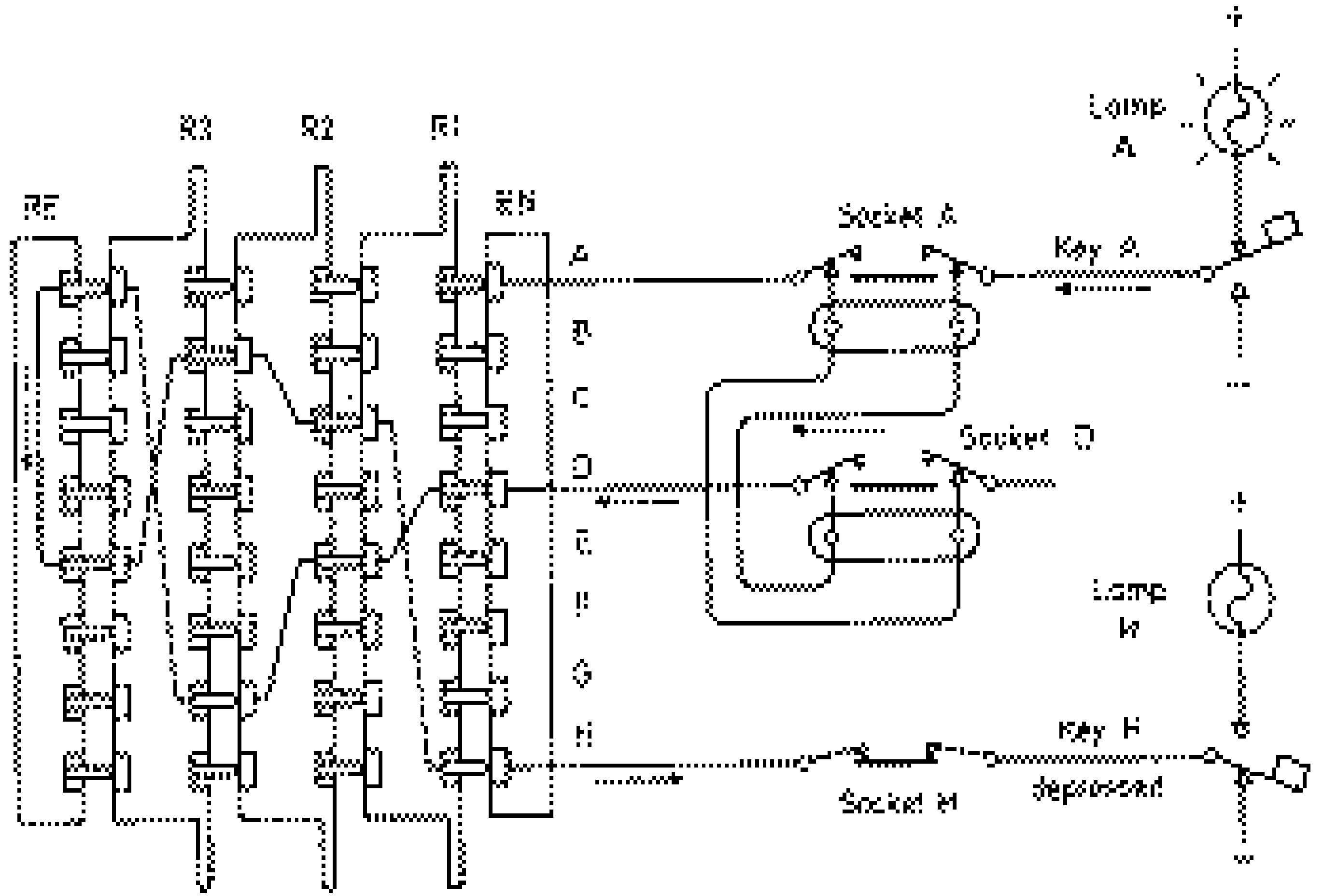
Machine Ciphers

- Jefferson cylinder was developed in 1790

- Wheatstone disc is designed by Wadsworth in1817



- **Enigma Rotor machine is used in world war II**



Classical Substitution Ciphers

- where letters of plaintext are replaced by other letters or by numbers or symbols
- or if plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns

Caesar Cipher

- earliest known substitution cipher
- by Julius Caesar
- first attested use in military affairs
- replaces each letter by 3rd letter on
- example:

  meet me after the toga party

  PHHW PH DIWHU WKH WRJD SDUWB

Caesar Cipher

- can define transformation as:

  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

  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

- mathematically give each letter a number

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
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

- then have Caesar cipher as:

  $c = \mathrm{E}(p) = (p + k) \bmod (26)$

  $p = \mathrm{D}(\mathrm{c}) = (\mathrm{c} - k) \bmod (26)$

Cryptanalysis of Caesar Cipher

- only have 26 possible ciphers
  - A maps to A,B,..Z
- could simply try each in turn
- a **brute force search**
- given ciphertext, just try all shifts of letters
- do need to recognize when have plaintext
- eg. break ciphertext "GCUA VQ DTGCM"

Monoalphabetic Cipher

- rather than just shifting the alphabet
- could shuffle (jumble) the letters arbitrarily
- each plaintext letter maps to a different random ciphertext letter
- hence key is 26 letters long

Plain: abcdefghijklmnopqrstuvwxyz
Cipher: DKVQFIBJWPESCXHTMYAUOLRGZN

Plaintext: ifwewishtoreplaceletters
Ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA

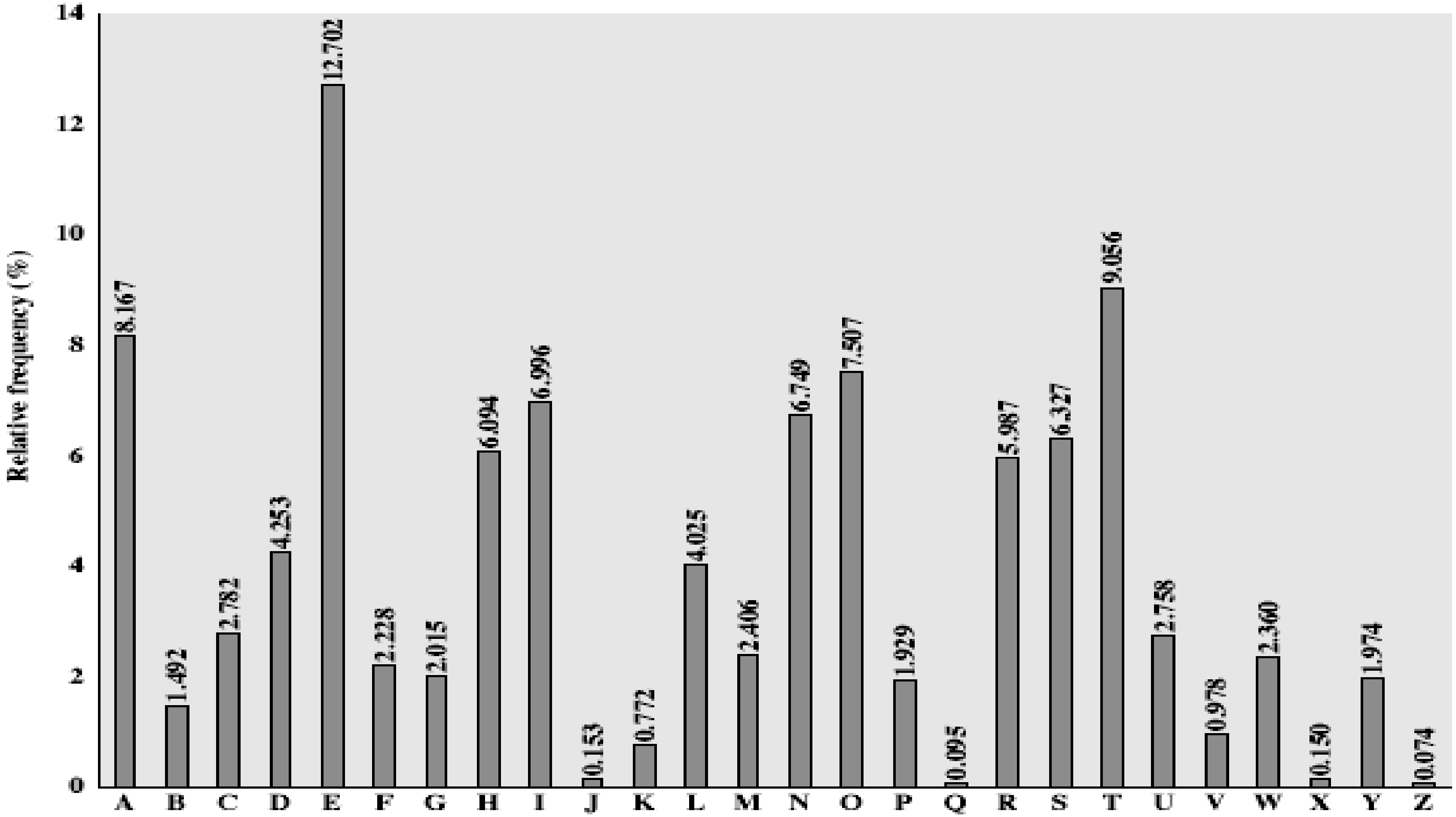
Monoalphabetic Cipher Security

- now have a total of 26! = 4 x 1026 keys
- with so many keys, might think is secure
- but would be **!!!WRONG!!!**
- problem is language characteristics

Language Redundancy and Cryptanalysis

- human languages are **redundant**
- eg "th lrd s m shphrd shll nt wnt"
- letters are not equally commonly used
- in English E is by far the most common letter
  - followed by T,R,N,I,O,A,S
- other letters like Z,J,K,Q,X are fairly rare
- have tables of single, double & triple letter frequencies for various languages

English Letter Frequencies



Use in Cryptanalysis

- key concept - monoalphabetic substitution ciphers do not change relative letter frequencies
- discovered by Arabian scientists in 9^{th} century
- calculate letter frequencies for ciphertext
- compare counts/plots against known values
- if caesar cipher look for common peaks/troughs
  - peaks at: A-E-I triple, NO pair, RST triple
  - troughs at: JK, X-Z
- for monoalphabetic must identify each letter
  - tables of common double/triple letters help

Example Cryptanalysis

- given ciphertext:

  UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
  VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
  EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

- count relative letter frequencies (see text)
- guess P & Z are e and t
- guess ZW is th and hence ZWP is the
- proceeding with trial and error finally get:

  it was disclosed yesterday that several informal but
  direct contacts have been made with political
  representatives of the viet cong in moscow

Playfair Cipher

- not even the large number of keys in a monoalphabetic cipher provides security
- one approach to improving security was to encrypt multiple letters
- the **Playfair Cipher** is an example
- invented by Charles Wheatstone in 1854, but named after his friend Baron Playfair

Playfair Key Matrix

- a 5X5 matrix of letters based on a keyword
- fill in letters of keyword (sans duplicates)
- fill rest of matrix with other letters
- eg. using the keyword MONARCHY

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

Encrypting and Decrypting

- plaintext is encrypted two letters at a time
  1. if a pair is a repeated letter, insert filler like 'X’
  2. if both letters fall in the same row, replace each with letter to right (wrapping back to start from end, ar encrypted as RM)
  3. if both letters fall in the same column, replace each with the letter below it (again wrapping to top from bottom, mu encrypted as CM)
  4. otherwise each letter is replaced by the letter in the same row and in the column of the other letter of the pair (hs becomes BP)

Security of Playfair Cipher

- security much improved over monoalphabetic
- since have 26 x 26 = 676 digrams
- would need a 676 entry frequency table to analyse (verses 26 for a monoalphabetic)
- and correspondingly more ciphertext
- was widely used for many years
  - eg. by US & British military in WW1
- it **can** be broken, given a few hundred letters
- since still has much of plaintext structure

Polyalphabetic Ciphers

- **polyalphabetic substitution ciphers**
- improve security using multiple cipher alphabets
- make cryptanalysis harder with more alphabets to guess and flatter frequency distribution
- use a key to select which alphabet is used for each letter of the message
- use each alphabet in turn
- repeat from start after end of key is reached

Vigenère Cipher

- simplest polyalphabetic substitution cipher
- effectively multiple caesar ciphers
- key is multiple letters long $K = k_1\ k_2 \ldots k_d$
- i^{th} letter specifies i^{th} alphabet to use
- use each alphabet in turn
- repeat from start after d letters in message
- decryption simply works in reverse

Example of Vigenère Cipher

- write the plaintext out
- write the keyword repeated above it
- use each key letter as a caesar cipher key
- encrypt the corresponding plaintext letter
- eg using keyword *deceptive*

key: deceptivedeceptivedeceptive
plaintext: wearediscoveredsaveyourself
ciphertext:ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Aids

- simple aids can assist with en/decryption
- a **Saint-Cyr Slide** is a simple manual aid
  - a slide with repeated alphabet
  - line up plaintext 'A' with key letter, eg 'C'
  - then read off any mapping for key letter
- can bend round into a **cipher disk**
- or expand into a **Vigenère Tableau**

Security of Vigenère Ciphers

- have multiple ciphertext letters for each plaintext letter
- hence letter frequencies are obscured
- but not totally lost
- start with letter frequencies
  - see if look monoalphabetic or not
- if not, then need to determine number of alphabets, since then can attach each

Kasiski Method

- method developed by Babbage / Kasiski
- repetitions in ciphertext give clues to period
- so find same plaintext an exact period apart
- which results in the same ciphertext
- of course, could also be random fluke
- eg repeated “VTW” in previous example
- suggests size of 3 or 9
- then attack each monoalphabetic cipher individually using same techniques as before

Autokey Cipher

- ideally want a key as long as the message
- Vigenère proposed the **autokey** cipher
- with keyword is prefixed to message as key
- knowing keyword can recover the first few letters
- use these in turn on the rest of the message
- but still have frequency characteristics to attack
- eg. given key *deceptive*

  key: deceptivewearediscoveredsav

  plaintext: wearediscoveredsaveyourself

  ciphertext:ZICVTWQNGKZEIIGASXSTSLVVWLA

One-Time Pad

- if a truly random key as long as the message is used, the cipher will be secure
- called a One-Time pad
- is unbreakable since ciphertext bears no statistical relationship to the plaintext
- since for **any plaintext** & **any ciphertext** there exists a key mapping one to other
- can only use the key **once** though
- problems in generation & safe distribution of key

Transposition Ciphers

- now consider classical **transposition** or **permutation** ciphers
- these hide the message by rearranging the letter order
- without altering the actual letters used
- can recognise these since have the same frequency distribution as the original text

Rail Fence cipher

- write message letters out diagonally over a number of rows
- then read off cipher row by row
- eg. write message out as: meet me after the toga party

```
m e m a t r h t g p r y
 e t e f e t e o a a t
```

- giving ciphertext

```
MEMATRHTGPRYETEFETEOAAT
```

Row Transposition Ciphers

- a more complex transposition
- write letters of message out in rows over a specified number of columns
- then reorder the columns according to some key before reading off the rows

Key: 3 4 2 1 5 6 7
Plaintext: a t t a c k p
o s t p o n e
d u n t i l t
w o a m x y z
Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ

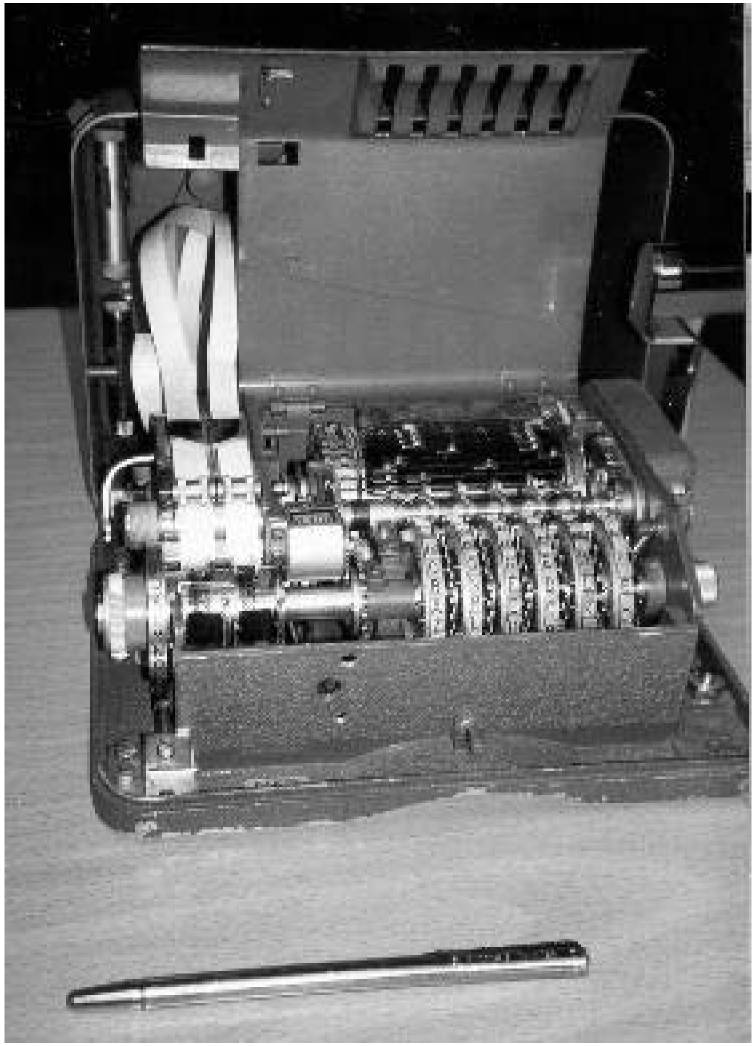
Product Ciphers

- ciphers using substitutions or transpositions are not secure because of language characteristics
- hence consider using several ciphers in succession to make harder, but:
  - two substitutions make a more complex substitution
  - two transpositions make more complex transposition
  - but a substitution followed by a transposition makes a new much harder cipher
- this is bridge from classical to modern ciphers

Rotor Machines

- before modern ciphers, rotor machines were most common complex ciphers in use
- widely used in WW2
  - German Enigma, Allied Hagelin, Japanese Purple
- implemented a very complex, varying substitution cipher
- used a series of cylinders, each giving one substitution, which rotated and changed after each letter was encrypted
- with 3 cylinders have 26^3=17576 alphabets

Hagelin Rotor Machine



Steganography

- an alternative to encryption
- hides existence of message
  - using only a subset of letters/words in a longer message marked in some way
  - using invisible ink
  - hiding in LSB in graphic image or sound file
- has drawbacks
  - high overhead to hide relatively few info bits

Finite Fields and Number Theory

- will now introduce finite fields
- of increasing importance in cryptography
  - AES, Elliptic Curve, IDEA, Public Key
- concern operations on “numbers”
  - where what constitutes a “number” and the type of operations varies considerably
- start with concepts of groups, rings, fields from abstract algebra

Group

- a set of elements or “numbers”
- with some operation whose result is also in the set (closure)
- obeys:
  - associative law: $(a.b).c = a.(b.c)$
  - has identity e: $e.a = a.e = a$
  - has inverses a^{-1}: $a.a^{-1} = e$
- if commutative $a.b = b.a$
  - then forms an **abelian group**

Cayley table of a group (* operation)

*	e	a	b	c
e	e	a	b	c
a	a	b	c	e
b	b	c	e	a
c	c	e	a	b

$a^1=a$, $a^2=b$, $a^3=c$ ve $a^4=e$ ‘
$b=a^2=a^6=a^{-2}$.
Each elements of {e,a,b,c} can be defined as a^n
a is a generqator of group.

Cyclic Group

- define **exponentiation** as repeated application of operator
  - example: $a^3 = a.a.a$
- and let identity be: $e=a^0$
- a group is cyclic if every element is a power of some fixed element
  - ie $b = a^k$ for some a and every b in group
- a is said to be a generator of the group

Ring

- a set of “numbers”
- with two operations (addition and multiplication) which form:
- an abelian group with addition operation
- and multiplication:
  - has closure
  - is associative
  - distributive over addition: `a(b+c) = ab + ac`
- if multiplication operation is commutative, it forms a **commutative ring**
- if multiplication operation has an identity and no zero divisors, it forms an **integral domain**

Field

- a set of numbers
- with two operations which form:
  - abelian group for addition
  - abelian group for multiplication (ignoring 0)
  - ring
- have hierarchy with more axioms/laws
  - group -> ring -> field

Modular Arithmetic

- define **modulo operator** "`a mod n`" to be remainder when a is divided by n
- use the term **congruence** for: `a = b mod n`
  - when divided by *n,* a & b have same remainder
  - eg. 100 = 34 mod 11
- b is called a **residue** of a mod n
  - since with integers can always write: `a = qn + b`
  - usually chose smallest positive remainder as residue
    - ie. `0 <= b <= n-1`
  - process is known as **modulo reduction**
    - `eg. -12 mod 7 = -5 mod 7 = 2 mod 7 = 9 mod 7`

Divisors

- say a non-zero number `b` **divides** `a` if for some `m` have `a=mb` (`a`, `b`, `m` all integers)
- that is `b` divides into `a` with no remainder
- denote this $b|a$
- and say that `b` is a **divisor** of `a`
- eg. all of 1,2,3,4,6,8,12,24 divide 24
- if $a|1$, $a = \pm 1$
- if $a|b$ and $b|a$, $a = \pm b$ dir.
- any $b \neq 0$ divides zero.
- if, $b|g$ and $b|h$, for any integer m,n $b|(mg + nh)$

Modular Arithmetic Operations

- is 'clock arithmetic'
- uses a finite number of values, and loops back from either end
- modular arithmetic is when do addition & multiplication and modulo reduce answer
- can do reduction at any point, ie
  - `a+b mod n = [a mod n + b mod n] mod n`
- if, $n|(a-b)$ then $a \equiv b$ ***mod*** n .
- $a \equiv b$ ***mod*** n , means $b \equiv a$ ***mod*** n .
- $a \equiv b$ ***mod*** n and $b \equiv c$ ***mod*** n, means $a \equiv c$ ***mod n***

Modulo 8 Addition Example

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Properties of Mod operation

- **Add** (a+b) mod n = [(a mod n) + (b mod n)] mod n
- **Sub** (a-b) mod n = [(a mod n) - (b mod n)] mod n
- **Mult.** axb mod n = [(a mod n) x (b mod n)] mod n
- Derived with repeated adition
- Neither a nor b not equal zero may be a.b modn =0
  - Ex. 2.5 mod 10
- **Div.** a/b mod n
- Like multiply inverse of b: a/b = a.b^{-1} mod n
- İf n prime there is b^{-1} mod n and b.b^{-1} = 1 mod n
  - Ex. 2.3=1 mod 5 so,4/2=4.3=2 mod 5 .

Modular Arithmetic

- can do modular arithmetic with any group of integers: $Z_n = \{0, 1, \ldots , n-1\}$
- Z_n is defined as residue class [r] = { a : a integer; such as a = r mod n
- form a commutative ring for addition
- with a multiplicative identity
- note some peculiarities
  - if $(a+b)=(a+c) \bmod n$
    then $b=c \bmod n$
  - but if $(a.b)=(a.c) \bmod n$
    then $b=c \bmod n$ only if a is relatively prime to n

$\in$

properties

property	description
Commutative	(a + b) mod n = (b + a) mod n (a x b) mod n = (b x a) mod n
Associative	[(a+b) +c] mod n = [a +(b + c)] mod n [(axb) x c] mod n = [a x(b x c)] mod n
Distributive	[ax(b + c)] mod n = [(axb) + (axc)] mod n
Identity element	(0 + a) mod n = a mod n (1xa) mod n = a mod n
Additive invers(-a)	For $\forall$a Z_n there is a b such as; a + b = 0 mod n .

if order of finite field is p^n, (p prime number)
This field called **Galois Field modulo p** and shown as **GF(p^n)**

Greatest Common Divisor (GCD)

- a common problem in number theory
- GCD (a,b) of a and b is the largest number that divides evenly into both a and b
  - eg GCD(60,24) = 12
- often want **no common factors** (except 1) and hence numbers are **relatively prime**
  - eg GCD(8,15) = 1
  - hence 8 & 15 are relatively prime

Euclidean Algorithm

- an efficient way to find the GCD(a,b)
- uses theorem that:
  - GCD(a,b) = GCD(b, a mod b)
- Euclidean Algorithm to compute GCD(a,b) is:
- ***GCD (a,n) is given by:***
- ***let g0=n   g1=a***
- ***gi+1 = gi-1 mod gi***
- ***when gi=0 then GCD(a,n) = gi-1***
- ex. Find GCD (56,98) ‘.
- g0=98   g1=56   g2 = 98 mod 56 = 42   g3 = 56 mod 42 = 14   g4 = 42 mod 14 = 0 as a result GCD(56,98)=14

Example GCD(1970,1066)

```
1970 = 1 x 1066 + 904      gcd(1066, 904)
1066 = 1 x 904 + 162       gcd(904, 162)
904 = 5 x 162 + 94         gcd(162, 94)
162 = 1 x 94 + 68          gcd(94, 68)
94 = 1 x 68 + 26           gcd(68, 26)
68 = 2 x 26 + 16           gcd(26, 16)
26 = 1 x 16 + 10           gcd(16, 10)
16 = 1 x 10 + 6            gcd(10, 6)
10 = 1 x 6 + 4             gcd(6, 4)
6 = 1 x 4 + 2              gcd(4, 2)
4 = 2 x 2 + 0              gcd(2, 0)
```

Galois Fields

- finite fields play a key role in cryptography
- can show number of elements in a finite field **must** be a power of a prime p^n
- known as Galois fields
- denoted $GF(p^n)$
- in particular often use the fields:
  - $GF(p)$
  - $GF(2^n)$

Galois Fields GF(p)

- GF(p) is the set of integers {0,1, … , p-1} with arithmetic operations modulo prime p
- these form a finite field
  - since have multiplicative inverses
- hence arithmetic is "well-behaved" and can do addition, subtraction, multiplication, and division without leaving the field GF(p)

GF(7) Multiplication Example

$\times$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

GF(7)additive and multip. İnverse

For $\forall w \in Z_p$ there is z in Z_p and $w \boldsymbol{x} z = 1 \bmod p$

w	-w	w^{-1}
0	0	-
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

Polynomial Arithmetic

- can compute using polynomials

  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = \sum a_i x^i$

  - nb. not interested in any specific value of x
  - which is known as the indeterminate
- several alternatives available
  - ordinary polynomial arithmetic
  - poly arithmetic with coords mod p
  - poly arithmetic with coords mod p and polynomials mod m(x)

Ordinary Polynomial Arithmetic

- add or subtract corresponding coefficients
- multiply all terms by each other
- eg

  let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$

  $f(x) + g(x) = x^3 + 2x^2 - x + 3$

  $f(x) - g(x) = x^3 + x + 1$

  $f(x) \text{ x } g(x) = x^5 + 3x^2 - 2x + 2$

Polynomial Arithmetic with Modulo Coefficients

- when computing value of each coefficient do calculation modulo some value
  - forms a polynomial ring
- could be modulo any prime
- but we are most interested in mod 2
  - ie all coefficients are 0 or 1
  - eg. let $f(x) = x^3 + x^2$ and $g(x) = x^2 + x + 1$

$$f(x) + g(x) = x^3 + x + 1$$

$$f(x) \text{ x } g(x) = x^5 + x^2$$

Polynomial Division

- can write any polynomial in the form:
  - $f(x) = q(x)\, g(x) + r(x)$
  - can interpret $r(x)$ as being a remainder
  - $r(x) = f(x) \bmod g(x)$
- if have no remainder say $g(x)$ divides $f(x)$
- if $g(x)$ has no divisors other than itself & 1 say it is **irreducible** (or prime) polynomial
- arithmetic modulo an irreducible polynomial forms a field

Polynomial GCD

- can find greatest common divisor for polys
  - $c(x)$ = GCD($a(x), b(x)$) if $c(x)$ is the poly of greatest degree which divides both $a(x), b(x)$
- can adapt Euclid's Algorithm to find it:

  EUCLID[$a(x), b(x)$]

  1. $A(x) = a(x)$; $B(x) = b(x)$

  2. if $B(x) = 0$ **return** $A(x) = \gcd[a(x), b(x)]$

  3. $R(x) = A(x) \bmod B(x)$

  4. A(x) ¨ B(x)

  5. B(x) ¨ R(x)

  6. goto 2

Modular Polynomial Arithmetic

- can compute in field $GF(2^n)$
  - polynomials with coefficients modulo 2
  - whose degree is less than n
  - hence must reduce modulo an irreducible poly of degree n (for multiplication only)
- form a finite field
- can always find an inverse
  - can extend Euclid's Inverse algorithm to find

Example GF(2^3)

Table 4.6 Polynomial Arithmetic Modulo ($x^3 + x + 1$)

	+	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

	×	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

(b) Multiplication

Computational Considerations

- since coefficients are 0 or 1, can represent any such polynomial as a bit string
- addition becomes XOR of these bit strings
- multiplication is shift & XOR
  - cf long-hand multiplication
- modulo reduction done by repeatedly substituting highest power with remainder of irreducible poly (also shift & XOR)

Computational Example

- in $GF(2^3)$ have (x^2+1) is 101_2 & (x^2+x+1) is 111_2
- so addition is
  - $(x^2+1) + (x^2+x+1) = x$
  - 101 XOR 111 = 010_2
- and multiplication is
  - $(x+1).(x^2+1) = x.(x^2+1) + 1.(x^2+1)$
    $= x^3+x+x^2+1 = x^3+x^2+x+1$
  - 011.101 = (101)<<1 XOR (101)<<0 =
    1010 XOR 101 = 1111_2
- polynomial modulo reduction (get q(x) & r(x)) is
  - (x^3+x^2+x+1) mod $(x^3+x+1) = 1.(x^3+x+1) + (x^2) = x^2$
  - 1111 mod 1011 = 1111 XOR 1011 = 0100_2

Using a Generator

- equivalent definition of a finite field
- a **generator** g is an element whose powers generate all non-zero elements
  - in F have 0, g^0, g^1, …, g^{q-2}
- can create generator from **root** of the irreducible polynomial
- then implement multiplication by adding exponents of generator

Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179 181 191
193 197 199
```

Prime Factorisation

- to **factor** a number `n` is to write it as a product of other numbers: `n=a x b x c`
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number `n` is when its written as a product of primes
  - eg. $91=7\times13$; $3600=2^4\times3^2\times5^2$

$$a = \prod_{p \in \mathrm{P}} p^{a_p}$$

Relatively Prime Numbers & GCD

- two numbers `a, b` are **relatively prime** if have **no common divisors** apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - eg. $300=2^1 \times 3^1 \times 5^2$ $18=2^1 \times 3^2$ hence $GCD(18,300)=2^1 \times 3^1 \times 5^0=6$

Fermat's Theorem

- $a^{p-1} = 1 \pmod{p}$
  - where p is prime and $\gcd(a,p)=1$
- also known as Fermat's Little Theorem
- also $a^{p} = p \pmod{p}$
- useful in public key and primality testing

- If elements of Z_p multiply with {0,1,....,(p-1)} *a* , modulo p residues sequences of $Z_{p\,.}$ In addition, a x 0 = 0mod p ., So array of { *a*mod p, 2*a*modp, ...,(*p*-1)*a*mod p } is (p-1) number {0,1,....,(p-1)}.
- *a*x2*a*x … x ((p-1)*a*) = [(*a* modp) x (2*a*modp) x .....x((p-1)a modp]modp
- = [1 x 2 x ... x (p-1)] modp
- = (p-1)! modp
- But, a x 2a x ......x((p-1)a) = (p-1)!a^{p-1} .
- So, (p-1)!a^{p-1} = (p-1)! Modp. Here we can cancel (p-1)! ‘As a result:
- a^{p-1} = 1 modp

Euler Totient Function ø(n)

- when doing arithmetic modulo n
- **complete set of residues** is: `0..n-1`
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the **Euler Totient Function ø(n)**

Euler Totient Function $\phi(n)$

- to compute ø(n) need to count number of residues to be excluded
- in general need prime factorization, but
  - for p (p prime) $\phi(p) = p-1$
  - for p.q (p,q prime) $\phi(pq) = (p-1) \times (q-1)$
- eg.

  $\phi(37) = 36$

  $\phi(21) = (3-1) \times (7-1) = 2 \times 6 = 12$

Primitive Roots

- from Euler's theorem have $a^{\phi(n)} \bmod n=1$
- consider $a^m=1 \pmod n$, $GCD(a,n)=1$
  - must exist for $m = \phi(n)$ but may be smaller
  - once powers reach m, cycle will repeat
- if smallest is $m = \phi(n)$ then a is called a **primitive root**
- if p is prime, then successive powers of a "generate" the group $\bmod p$
- these are useful but relatively hard to find

$\phi(n)$ values for n=30

n	$\phi(n)$	**n**	$\phi(n)$	**n**	$\phi(n)$
1	1	11	10	21	12
2	1	12	4	22	10
3	2	13	12	23	22
4	2	14	6	24	8
5	4	15	8	25	20
6	2	16	8	26	12
7	6	17	16	27	18
8	4	18	6	28	12
9	6	19	18	29	28
10	4	20	8	30	8

Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{\varnothing(n)} = 1 \pmod{n}$
  - for any `a,n` where `gcd(a,n)=1`
- eg.

  $a=3; n=10;\ \varnothing(10)=4;$

  hence $3^4 = 81 = 1 \bmod 10$

  $a=2; n=11;\ \varnothing(11)=10;$

  hence $2^{10} = 1024 = 1 \bmod 11$

Primality Testing

- often need to find large prime numbers
- traditionally **sieve** using **trial division**
  - ie. divide by all numbers (primes) in turn less than the square root of the number
  - only works for small numbers
- alternatively can use statistical primality tests based on properties of primes
  - for which all primes numbers satisfy property
  - but some composite numbers, called pseudo-primes, also satisfy the property
- can use a slower deterministic primality test

Miller Rabin Algorithm

- a test based on Fermat's Theorem
- algorithm is:

TEST (*n*) is:

1. Find integers *k*, *q*, *k* > 0, *q* odd, so that $(n-1)=2^k q$
2. Select a random integer a, $1<a<n-1$
3. **if** $a^q \bmod n = 1$ **then** return ("maybe prime");
4. **for** *j* = 0 **to** *k* – 1 **do**
   5. **if** ($a^{2^j q} \bmod n = n-1$)
      then return(" maybe prime ")
6. return ("composite")

Probabilistic Considerations

- if Miller-Rabin returns "composite" the number is definitely not prime
- otherwise is a prime or a pseudo-prime
- chance it detects a pseudo-prime is < $^1/_4$
- hence if repeat test with different random a then chance n is prime after t tests is:
  - Pr(n prime after t tests) = $1-4^{-t}$
  - eg. for t=10 this probability is > 0.99999

Prime Distribution

- prime number theorem states that primes occur roughly every (`ln n`) integers
- but can immediately ignore evens
- so in practice need only test `0.5 ln(n)` numbers of size `n` to locate a prime
  - note this is only the "average"
  - sometimes primes are close together
  - other times are quite far apart

Discrete Logarithms

- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find x such that $y = g^x \pmod{p}$
- this is written as $x = \log_g y \pmod{p}$
- if g is a primitive root then it always exists, otherwise it may not, eg.

  $x = \log_3 4 \bmod 13$ has no answer

  $x = \log_2 3 \bmod 13 = 4$ by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem

Chinese Remainder Theorem

- used to speed up modulo computations
- if working modulo a product of numbers
  - eg. $\text{mod } M = m_1 m_2 .. m_k$
- Chinese Remainder theorem lets us work in each moduli m_i separately
- since computational cost is proportional to size, this is faster than working in the full modulus M

Chinese Remainder Theorem

- can implement CRT in several ways
- to compute $\mathtt{A(mod\ M)}$
  - first compute all $\mathtt{a_i = A\ mod\ m_i}$ separately
  - determine constants $\mathtt{c_i}$ below, where $\mathtt{M_i = M/m_i}$
  - then combine results to get answer using:

$$A \equiv \left(\sum_{i=1}^{k} a_i c_i \right) \pmod{M}$$

$$c_i = M_i \times \left(M_i^{-1} \bmod m_i\right) \quad \text{for } 1 \le i \le k$$

- **Chinese Remainder Theorem:** For relatively prime moduli *m* and *n*, the congruences
  x ≡ a (mod m)
  x ≡ b (mod n)

have a unique solution *x* modulo *mn*. Our example problem would have a unique solution modulo .

- x ≡ 13 (mod 27)
  x ≡ 7 (mod 16)

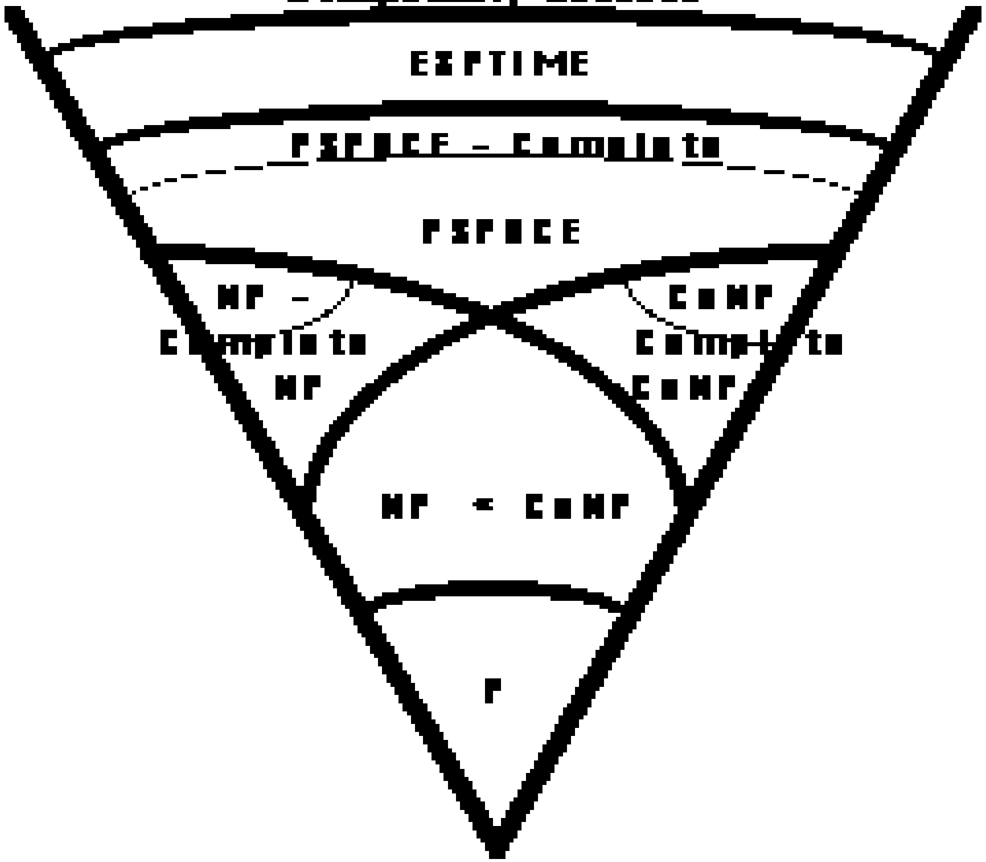
Our example problem would have a unique solution modulo 27.16

Computational Complexity Theory

- study of how hard a problem is to solve in general
- allows classification of types of problems
- some problems intrinsically harder than others, eg
  - multiplying numbers $O(n^2)$
  - multiplying matrices $O(n^{(2)(2n-1)})$ (n^(2)(2n-1))
  - solving crossword O(26^(n))
  - recognizing primes O(n^(log log n))
- deal with worst case complexity
  - may on average be easier

Computational Complexity Theory

Classification of Problems - Complexity Classes



Some Unknowns in Complexity Theory :

i) NP = P

ii) NP = CoNP

iii) P = CoNP = NP

Computational Complexity Theory

- an **instance** of a problem is a particular case of a general problem
- the **input length** of a problem is the number **n** of symbols used to characterize a particular instance of it
- the **order** of a function **f(n)** - is some **O(g(n))** of some function **g(n)** s.t.
  - $f(n) <= c.|g(n)|$, for all $n >= 0$, for some c

Computational Complexity Theory

- a **polynomial time** algorithm (**P**) is one which solves any instance of a particular problem in a length of time O(p(n)), where p is some polynomial on input length
- an **exponential time** algorithm (**E**) - is one whose solution time is not so bounded
- a **non-deterministic polynomial time** algorithm (**NP**) - is one for which any guess at the solution of an instance of the problem may be checked for validity in polynomial time

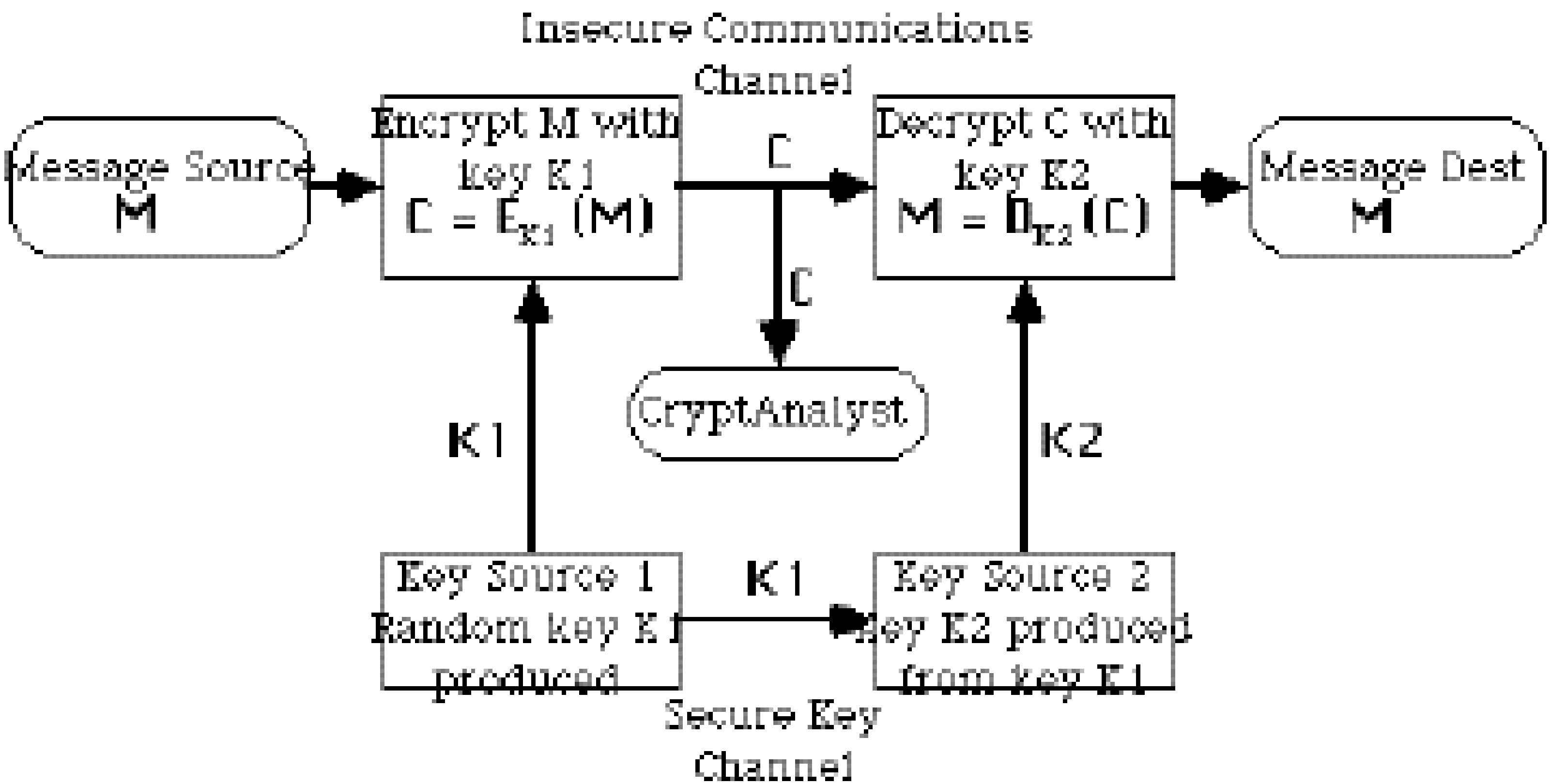
Computational Complexity Theory

- **NP-complete** problems - are a subclass of NP problems for which it is known that if any such problem has a polynomial time solution, then **all NP** problems have polynomial solutions. These are thus the **hardest NP** problems
- **Co-NP** problems - are the complements of **NP** problems, to prove a guess at a solution of a Co-NP problem may well require an exhaustive search of the solution space

Modern Block Ciphers

- now look at modern block ciphers
- one of the most widely used types of cryptographic algorithms
- provide secrecy /authentication services
- focus on DES (Data Encryption Standard)
- to illustrate block cipher design principles

Symmetric Cryptosystems

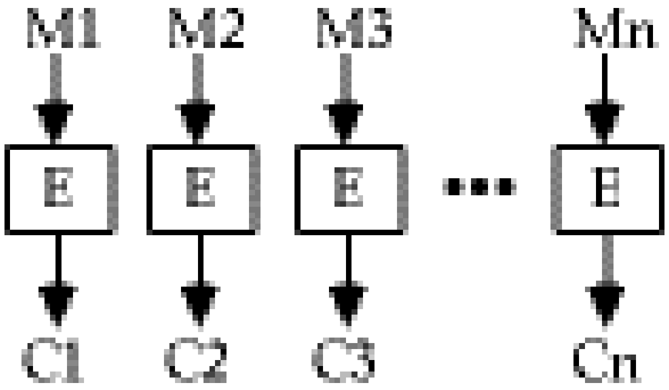


Symmetric (Private-Key) Encryption System

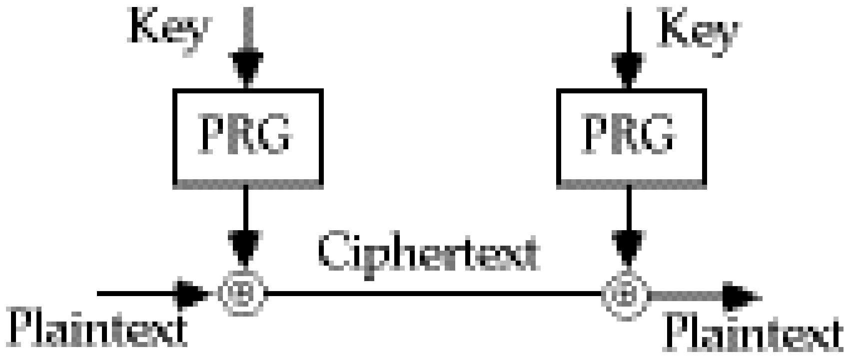
Block vs Stream Ciphers

- block ciphers process messages in blocks, each of which is then en/decrypted
- like a substitution on very big characters
  - 64-bits or more
- stream ciphers process messages a bit or byte at a time when en/decrypting
- many current ciphers are block ciphers
- broader range of applications

Block and Stream ciphers



Block Cipher

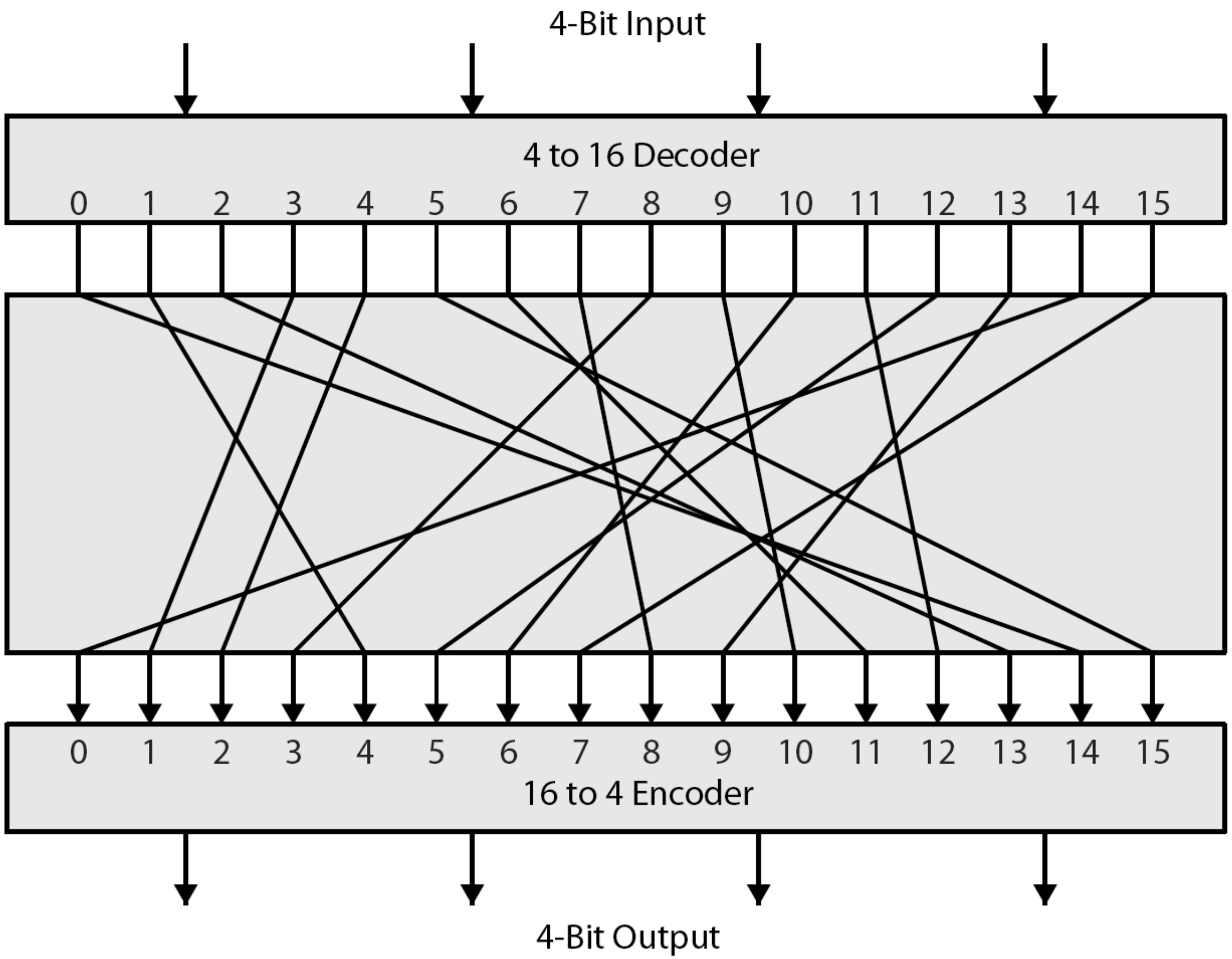


Stream Cipher

Block Cipher Principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently
- block ciphers look like an extremely large substitution
- would need table of 2^{64} entries for a 64-bit block
- instead create from smaller building blocks
- using idea of a product cipher

Ideal Block Cipher



Claude Shannon and Substitution-Permutation Ciphers

- Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper
- form basis of modern block ciphers
- S-P nets are based on the two primitive cryptographic operations seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
- provide *confusion* & *diffusion* of message & key

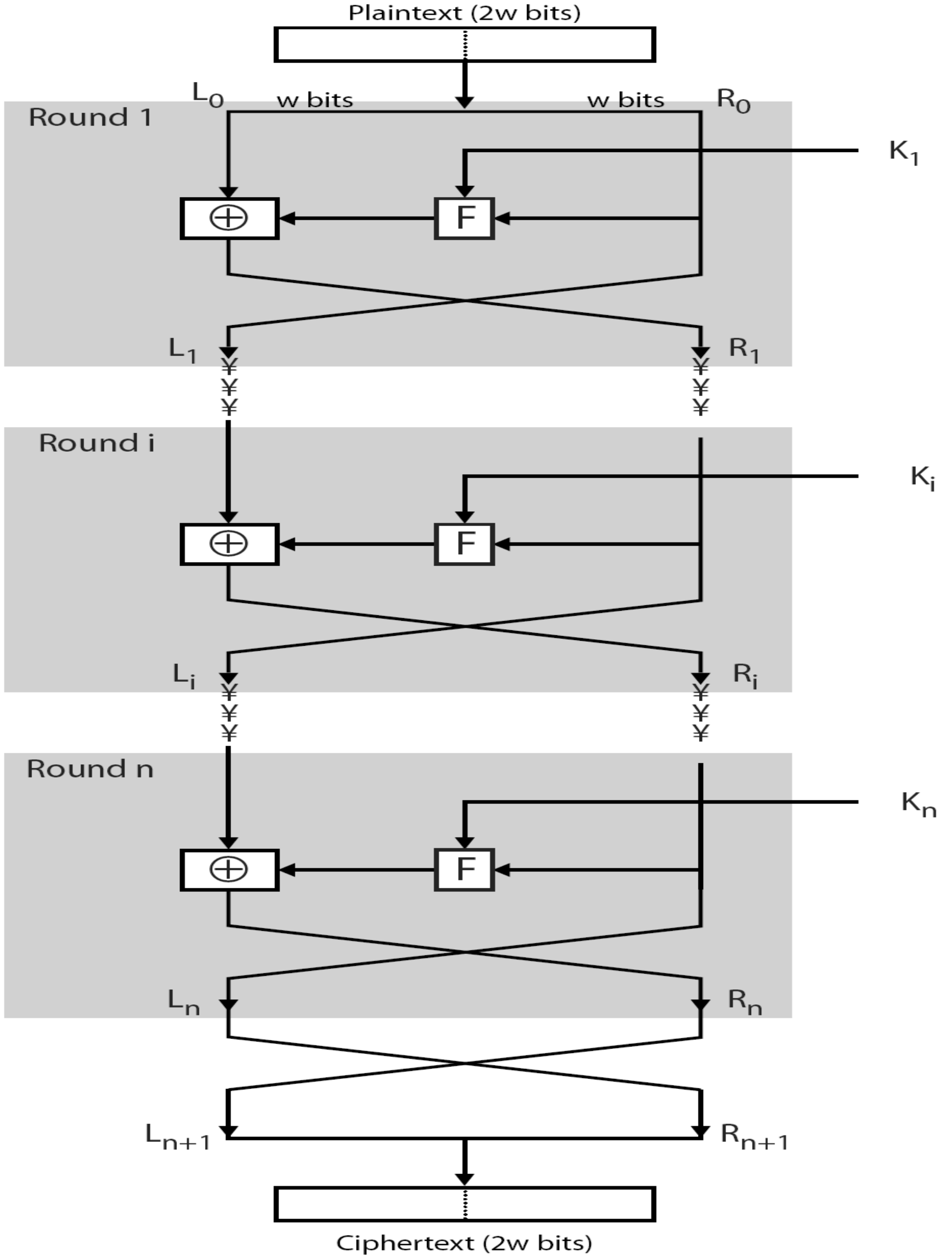
Confusion and Diffusion

- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this
- more practically Shannon suggested combining S & P elements to obtain:
- **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

Feistel Cipher Structure

- 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

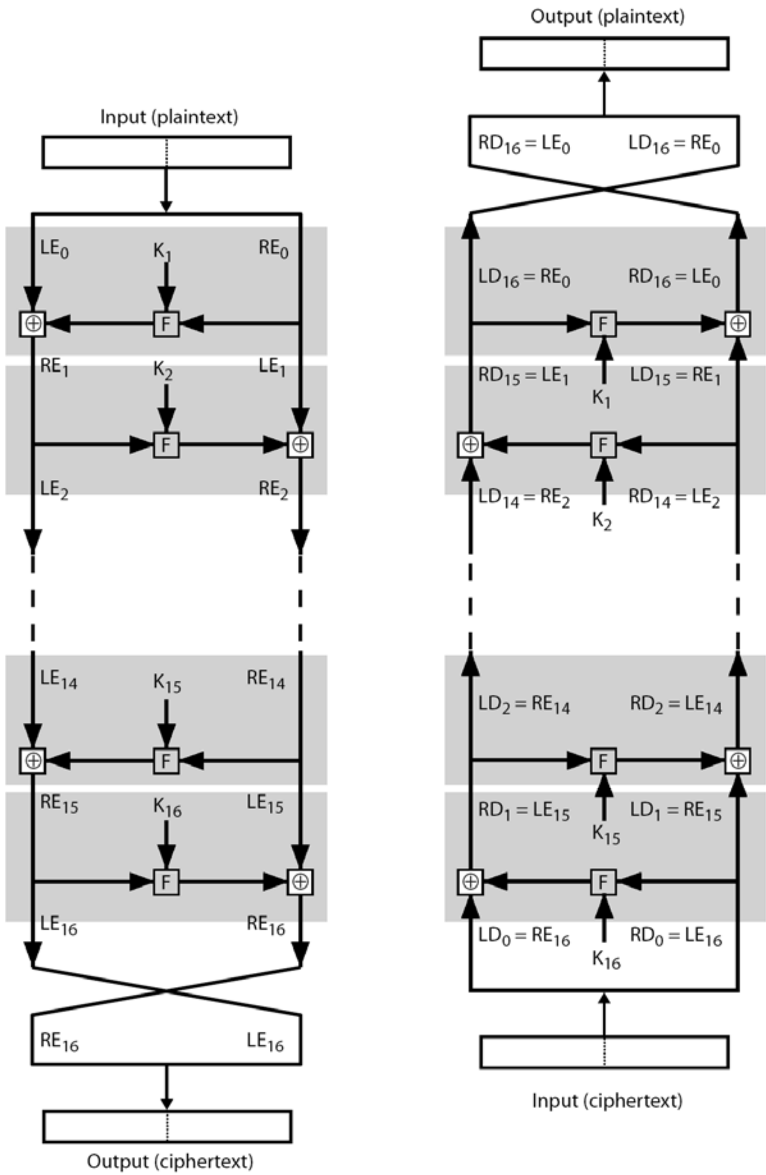
Feistel Cipher Structure



Feistel Cipher Design Elements

- block size
- key size
- number of rounds
- subkey generation algorithm
- round function
- fast software en/decryption
- ease of analysis

Feistel Cipher Decryption



Data Encryption Standard (DES)

- most widely used block cipher in world
- adopted in 1977 by NBS (now NIST)
  - as FIPS PUB 46
- encrypts 64-bit data using 56-bit key
- has widespread use
- has been considerable controversy over its security

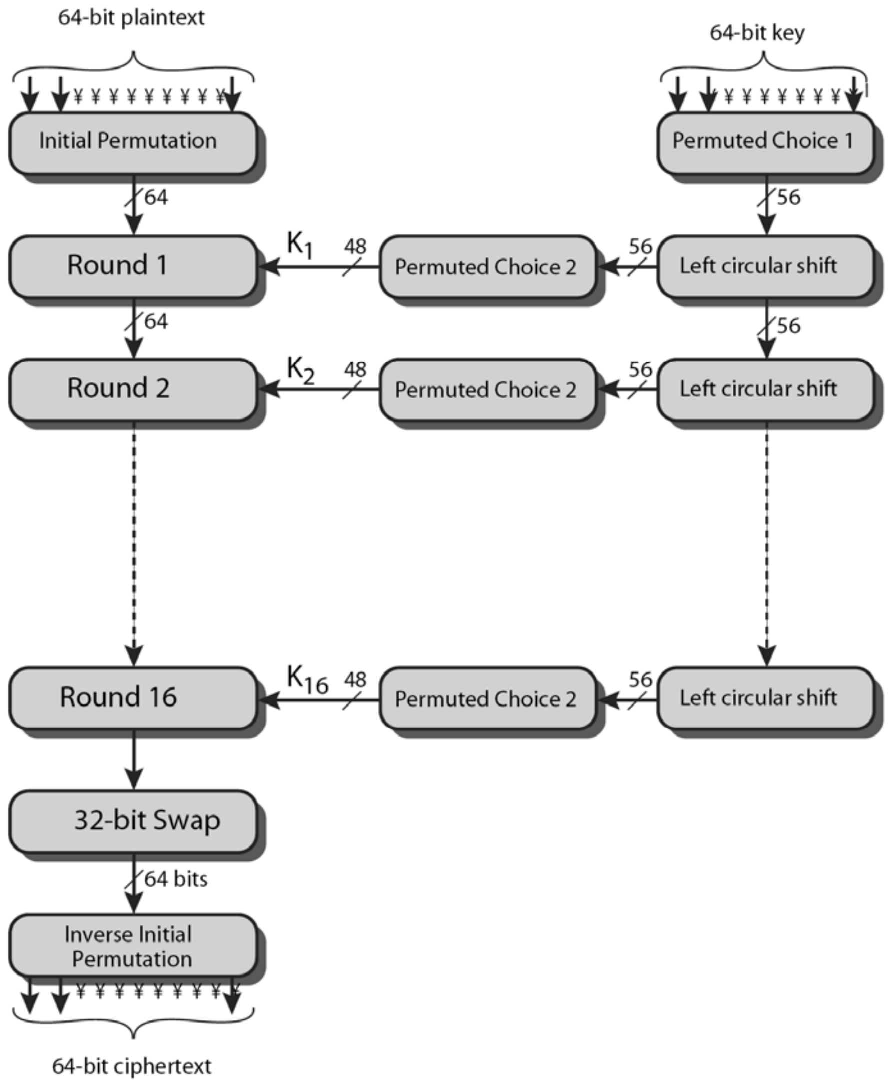
DES History

- IBM developed Lucifer cipher
  - by team led by Feistel in late 60's
  - used 64-bit data blocks with 128-bit key
- then redeveloped as a commercial cipher with input from NSA and others
- in 1973 NBS issued request for proposals for a national cipher standard
- IBM submitted their revised Lucifer which was eventually accepted as the DES

DES Design Controversy

- although DES standard is public
- was considerable controversy over design
  - in choice of 56-bit key (vs Lucifer 128-bit)
  - and because design criteria were classified
- subsequent events and public analysis show in fact design was appropriate
- use of DES has flourished
  - especially in financial applications
  - still standardised for legacy application use

DES Encryption Overview



Initial Permutation IP

- first step of the data computation
- IP reorders the input data bits
- even bits to LH half, odd bits to RH half
- quite regular in structure (easy in h/w)
- example:

```
IP(675a6967 5e5a6b5a) = (ffb2194d
004df6fb)
```

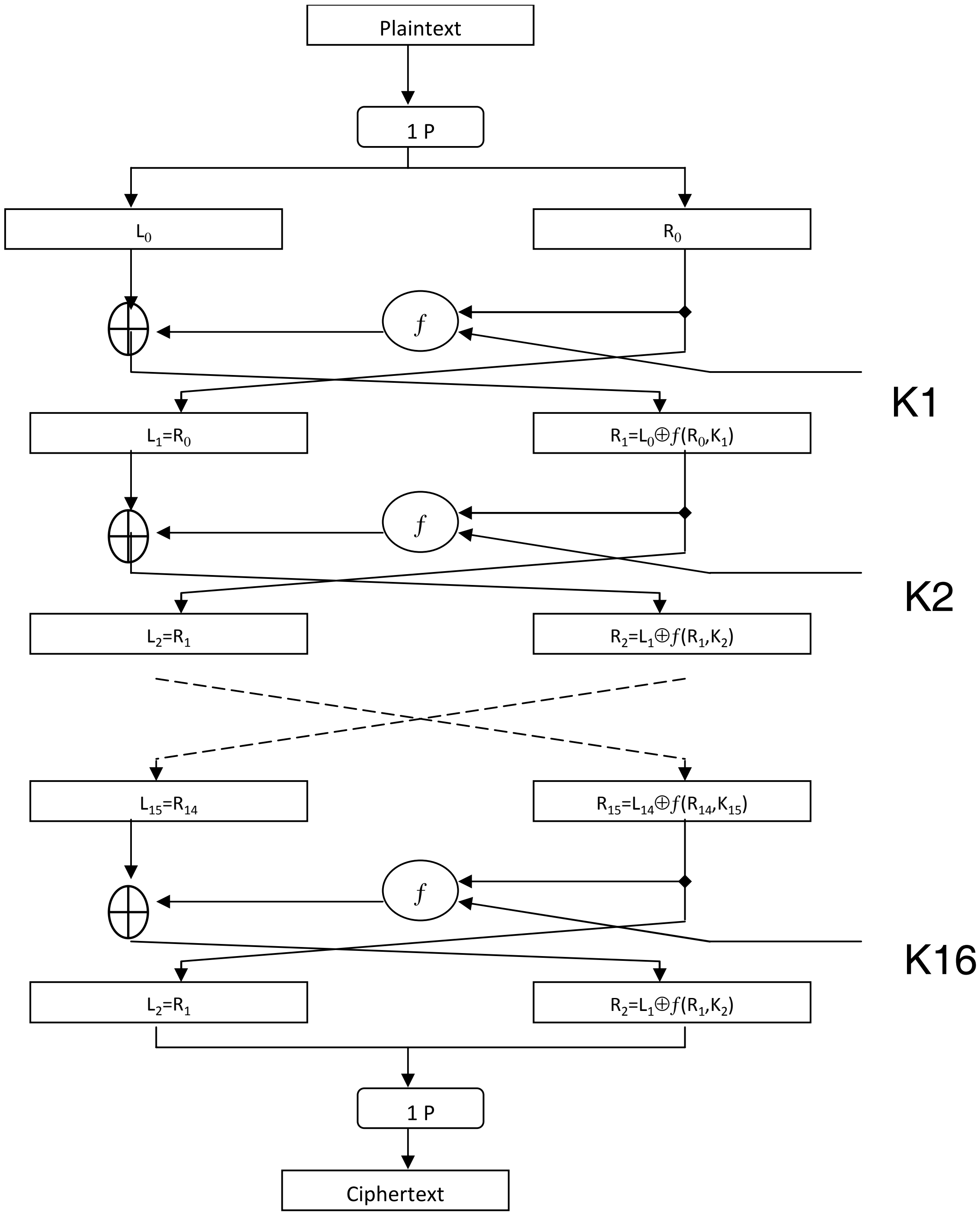
DES Round Structure

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:

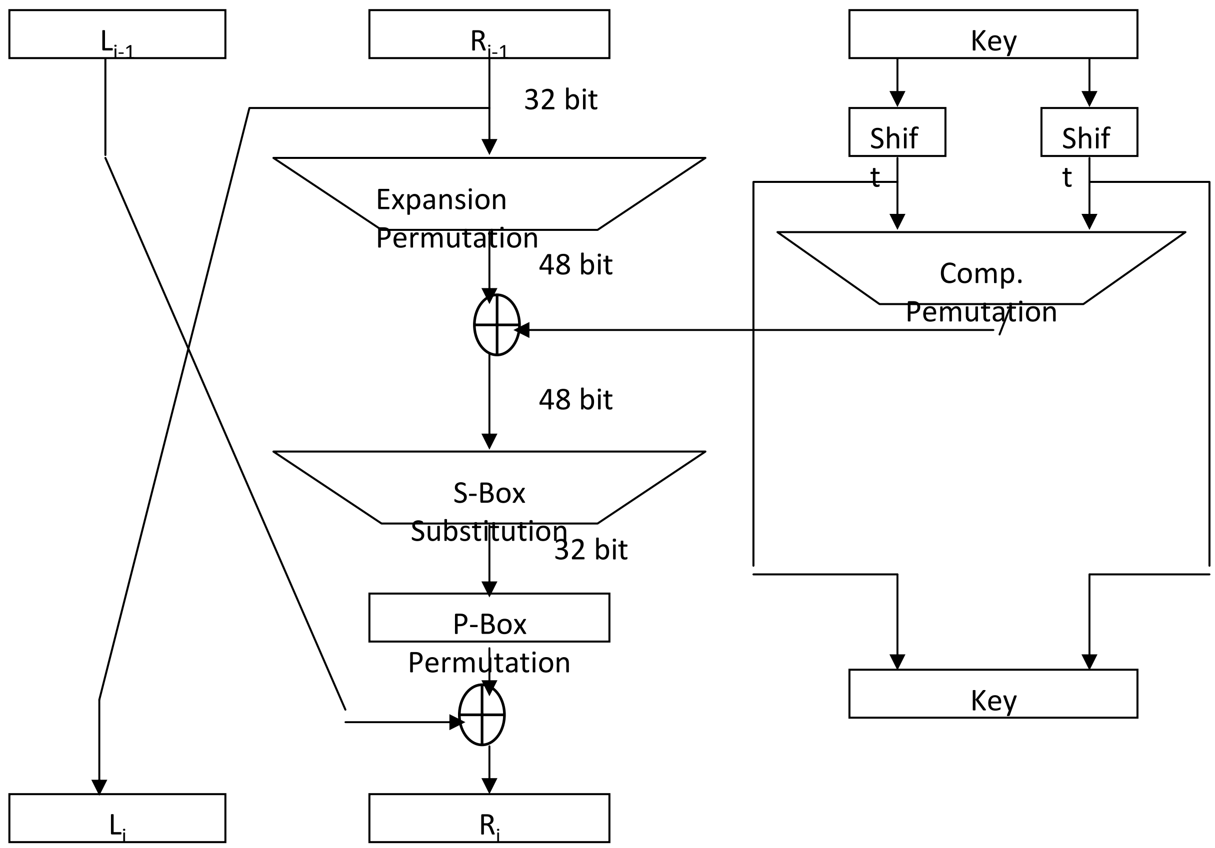
  $L_i = R_{i-1}$

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

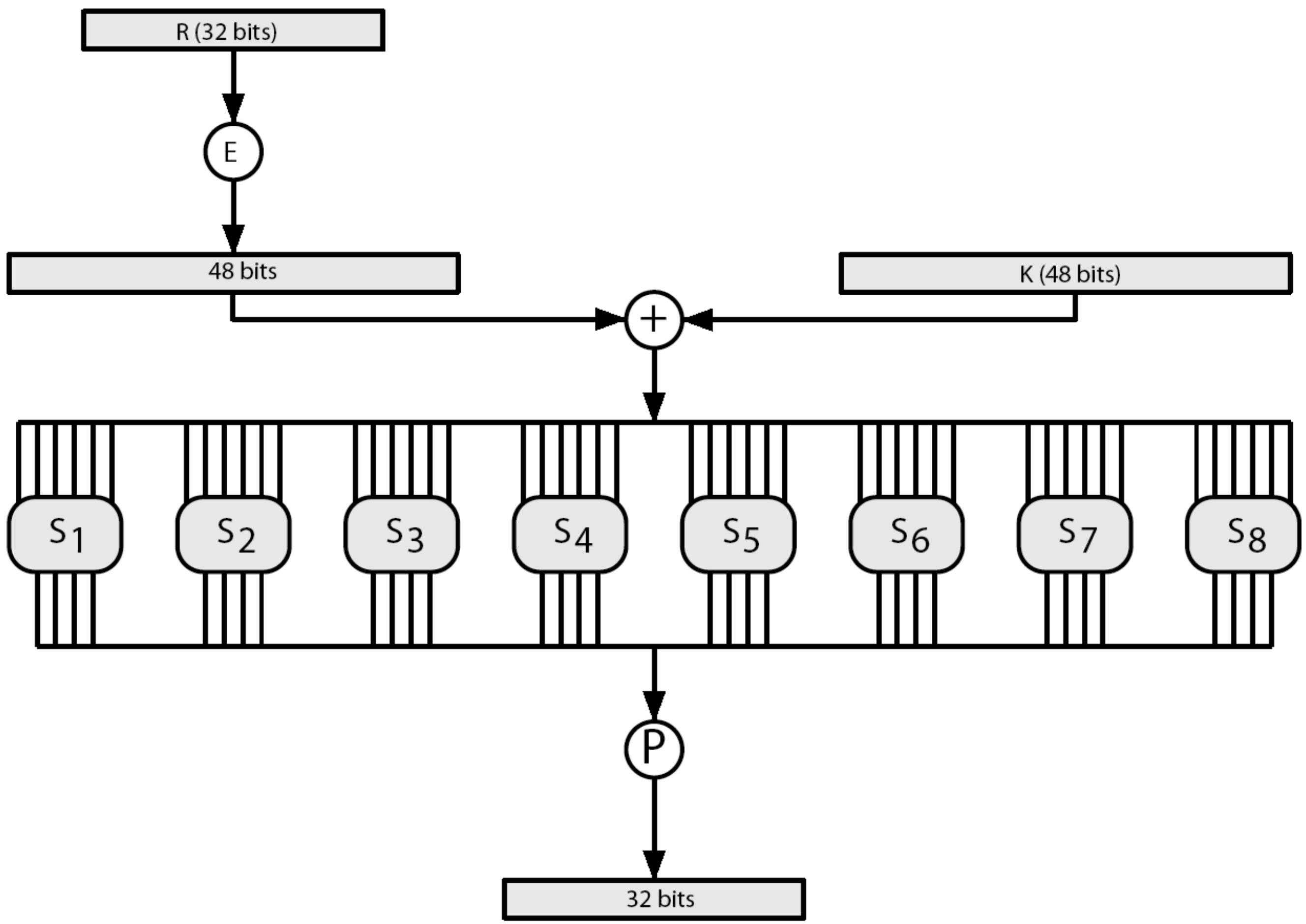
- F takes 32-bit R half and 48-bit subkey:
  - expands R to 48-bits using perm E
  - adds to subkey using XOR
  - passes through 8 S-boxes to get 32-bit result
  - finally permutes using 32-bit perm P


Plaintext
1 P
L0
R0
f
K1
L1=R0
R1=L0⊕f(R0,K1)
f
K2
L2=R1
R2=L1⊕f(R1,K2)
L15=R14
R15=L14⊕f(R14,K15)
f
K16
L2=R1
R2=L1⊕f(R1,K2)
1 P
Ciphertext

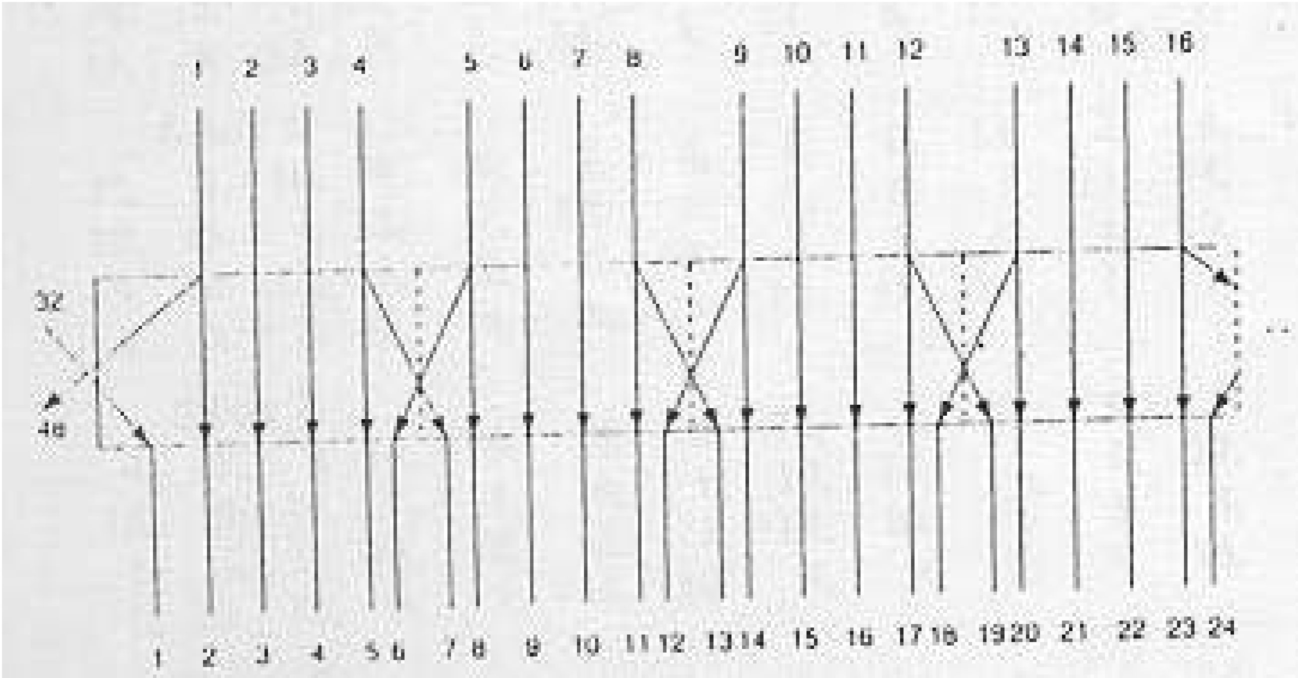
One round of DES



DES Round Structure



Expanded permutation



Substitution Boxes S

- have eight S-boxes which map 6 to 4 bits
- each S-box is actually 4 little 4 bit boxes
  - outer bits 1 & 6 (**row** bits) select one row of 4
  - inner bits 2-5 (**col** bits) are substituted
  - result is 8 lots of 4 bits, or 32 bits
- row selection depends on both data & key
  - feature known as autoclaving (autokeying)
- example:
  - `S(18 09 12 3d 11 17 38 39) = 5fd25e03`

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S1 0:	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1:	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2:	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3:	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D
S2 0:	F	1	8	E	6	B	3	4	9	7	2	D	C	0	5	A
1:	3	D	4	7	F	2	8	E	C	0	1	A	6	9	B	5
2:	0	E	7	B	A	4	D	1	5	8	C	6	9	3	2	F
3:	D	8	A	1	3	F	4	2	B	6	7	C	0	5	E	9
S3 0:	A	0	9	E	6	3	F	5	1	D	C	7	B	4	2	8
1:	D	7	0	9	3	4	6	A	2	8	5	E	C	B	F	1
2:	D	6	4	9	8	F	3	0	B	1	2	C	5	A	E	7
3:	1	A	D	0	6	9	8	7	4	F	E	3	B	5	2	C
S4 0:	7	D	E	3	0	6	9	A	1	2	8	5	B	C	4	F
1:	D	8	B	5	6	F	0	3	4	7	2	C	1	A	E	9
2:	A	6	9	0	C	B	7	D	F	1	3	E	5	2	8	4
3:	3	F	0	6	A	1	D	8	9	4	5	B	C	7	2	E
S5 0:	2	C	4	1	7	A	B	6	8	5	3	F	D	0	E	9
1:	E	B	2	C	4	7	D	1	5	0	F	A	3	9	8	6
2:	4	2	1	B	A	D	7	8	F	9	C	5	6	3	0	E
3:	B	8	C	7	1	E	2	D	6	F	0	9	A	4	5	3
S6 0:	C	1	A	F	9	2	6	8	0	D	3	4	E	7	5	B
1:	A	F	4	2	7	C	9	5	6	1	D	E	0	B	3	8
2:	9	E	F	5	2	8	C	3	7	0	4	A	1	D	B	6
3:	4	3	2	C	9	5	F	A	B	E	1	7	6	0	8	D
S7 0:	4	B	2	E	F	0	8	D	3	C	9	7	5	A	6	1
1:	D	0	B	7	4	9	1	A	E	3	5	C	2	F	8	6
2:	1	4	B	D	C	3	7	E	A	F	6	8	0	5	9	2
3:	6	B	D	8	1	4	A	7	9	5	0	F	E	2	3	C
S8 0:	D	2	8	4	6	F	B	1	A	9	3	E	5	0	C	7
1:	1	F	D	8	A	3	7	4	C	5	6	B	0	E	9	2
2:	7	B	4	1	9	C	E	2	0	6	A	D	F	3	5	8
3:	2	1	E	7	4	A	8	D	F	C	9	0	3	5	6	B

P-box 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

P-Box Permutasyonu

DES Key Schedule

- forms subkeys used in each round
  - initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
  - 16 stages consisting of:
    - rotating **each half** separately either 1 or 2 places depending on the **key rotation schedule** K
    - selecting 24-bits from each half & permuting them by PC2 for use in round function F
- note practical use issues in h/w vs s/w

DES Decryption

- decrypt must unwind steps of data computation
- with Feistel design, do encryption steps again using subkeys in reverse order (SK16 … SK1)
  - IP undoes final FP step of encryption
  - 1st round with SK16 undoes 16th encrypt round
  - ….
  - 16th round with SK1 undoes 1st encrypt round
  - then final FP undoes initial encryption IP
  - thus recovering original data value

Avalanche Effect

- key desirable property of encryption alg
- where a change of **one** input or key bit results in changing approx **half** output bits
- making attempts to “home-in” by guessing keys impossible
- DES exhibits strong avalanche

Strength of DES – Key Size

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- brute force search looks hard
- recent advances have shown is possible
  - in 1997 on Internet in a few months
  - in 1998 on dedicated h/w (EFF) in a few days
  - in 1999 above combined in 22hrs!
- still must be able to recognize plaintext
- must now consider alternatives to DES

Strength of DES – Analytic Attacks

- now have several analytic attacks on DES
- these utilise some deep structure of the cipher
  - by gathering information about encryptions
  - can eventually recover some/all of the sub-key bits
  - if necessary then exhaustively search for the rest
- generally these are statistical attacks
- include
  - differential cryptanalysis
  - linear cryptanalysis
  - related key attacks

Strength of DES – Timing Attacks

- attacks actual implementation of cipher
- use knowledge of consequences of implementation to derive information about some/all subkey bits
- specifically use fact that calculations can take varying times depending on the value of the inputs to it
- particularly problematic on smartcards

Differential Cryptanalysis

- one of the most significant recent (public) advances in cryptanalysis
- known by NSA in 70's cf DES design
- Murphy, Biham & Shamir published in 90’s
- powerful method to analyse block ciphers
- used to analyse most current block ciphers with varying degrees of success
- DES reasonably resistant to it, cf Lucifer

Differential Cryptanalysis

- a statistical attack against Feistel ciphers
- uses cipher structure not previously used
- design of S-P networks has output of function *f* influenced by both input & key
- hence cannot trace values back through cipher without knowing value of the key
- differential cryptanalysis compares two related pairs of encryptions

Differential Cryptanalysis Compares Pairs of Encryptions

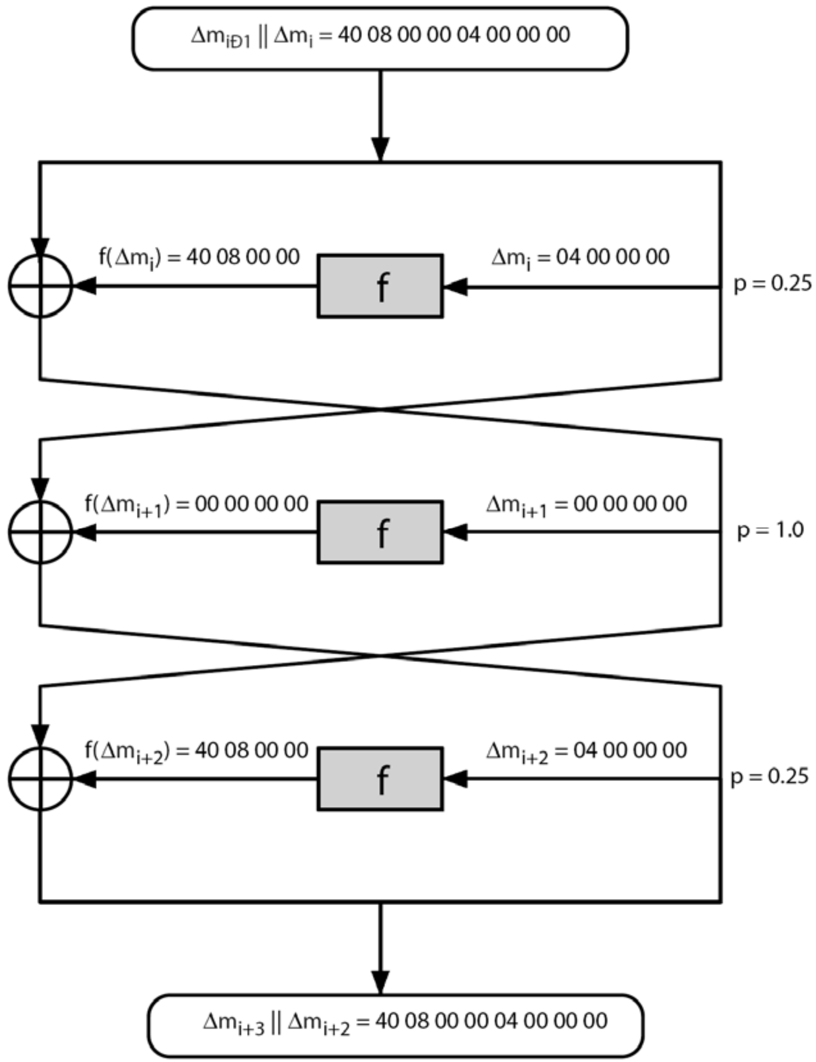
- with a known difference in the input
- searching for a known difference in output
- when same subkeys are used

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)]\end{aligned}$$

Differential Cryptanalysis

- have some input difference giving some output difference with probability p
- if find instances of some higher probability input / output difference pairs occurring
- can infer subkey that was used in round
- then must iterate process over many rounds (with decreasing probabilities)

Differential Cryptanalysis



Differential Cryptanalysis

- perform attack by repeatedly encrypting plaintext pairs with known input XOR until obtain desired output XOR
- when found
  - if intermediate rounds match required XOR have a **right pair**
  - if not then have a **wrong pair**, relative ratio is S/N for attack
- can then deduce keys values for the rounds
  - right pairs suggest same key bits
  - wrong pairs give random values
- for large numbers of rounds, probability is so low that more pairs are required than exist with 64-bit inputs
- Biham and Shamir have shown how a 13-round iterated characteristic can break the full 16-round DES

Linear Cryptanalysis

- another recent development
- also a statistical method
- must be iterated over rounds, with decreasing probabilities
- developed by Matsui et al in early 90's
- based on finding linear approximations
- can attack DES with 2^{43} known plaintexts, easier but still in practise infeasible

Linear Cryptanalysis

- find linear approximations with prob p != ½

  $P[i_1,i_2,...,i_a] \oplus C[j_1,j_2,...,j_b] = K[k_1,k_2,...,k_c]$

  where i_a,j_b,k_c are bit locations in P,C,K

- gives linear equation for key bits
- get one key bit using max likelihood alg
- using a large number of trial encryptions
- effectiveness given by: $|p - {}^1/_2|$

DES Design Criteria

- as reported by Coppersmith in [COPP94]
- 7 criteria for S-boxes provide for
  - non-linearity
  - resistance to differential cryptanalysis
  - good confusion
- 3 criteria for permutation P provide for
  - increased diffusion

Block Cipher Design

- basic principles still like Feistel’s in 1970’s
- number of rounds
  - more is better, exhaustive search best attack
- function f:
  - provides “confusion”, is nonlinear, avalanche
  - have issues of how S-boxes are selected
- key schedule
  - complex subkey creation, key avalanche

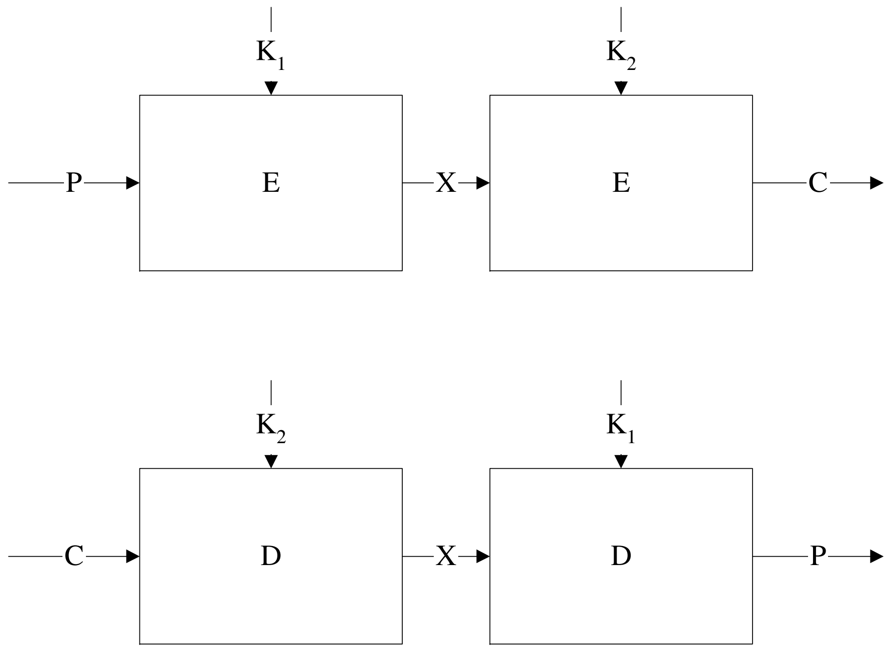
Multiple Encryption & DES

- clear a replacement for DES was needed
  - theoretical attacks that can break it
  - demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

Double-DES?

- could use 2 DES encrypts on each block
  - $C = E_{K2}(E_{K1}(P))$
- issue of reduction to single stage
- and have “meet-in-the-middle” attack
  - works whenever use a cipher twice
  - since $X = E_{K1}(P) = D_{K2}(C)$
  - attack by encrypting P with all keys and store
  - then decrypt C with keys and match X value
  - can show takes $O(2^{56})$ steps

Meet in the middle attack



Triple-DES with Two-Keys

- hence must use 3 encryptions
  - would seem to need 3 distinct keys
- but can use 2 keys with E-D-E sequence
  - $\texttt{C = } E_{K1}(D_{K2}(E_{K1}(P)))$
  - nb encrypt & decrypt equivalent in security
  - if $K1=K2$ then can work with single DES
- standardized in ANSI X9.17 & ISO8732
- no current known practical attacks

Triple-DES with Three-Keys

- although are no practical attacks on two-key Triple-DES have some indications
- can use Triple-DES with Three-Keys to avoid even these
  - $C = E_{K3}(D_{K2}(E_{K1}(P)))$
- has been adopted by some Internet applications, eg PGP, S/MIME

Algorithm	Key length	round	Mathematical operations	Applications
DES	56 Bit	16	XOR, fixed S-boxes	SET,Kerberos
Triple DES	112 or 168 bit	48	XOR, fixed S-boxes	Financial key management, PGP, S/MIME
IDEA	128 Bit	8	XOR, addition, multiplication	PGP
Blowfish	variable, 448 bit	16	XOR, variable S-Boxes, addition	
RC5	variable 2048 Bit	variable 255	addition, subtraction, XOR, round	
CAST-128	40-128 bit	16	addition, subtraction, XOR, round, fixed S-boxes	PGP

Properties of advanced block ciphers

- Variable key length
- Complex mathematical operations
- Data depended rounds
- Key depended S-box
- Multiple length key arrangement algorithms
- Variable plain/cipher text length
- Variable round number
- Operation for each half data at each round
- Variable F Function
- Key depended rounds

Modes of Operation

- block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
- need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- subsequently 5 defined for AES & DES
- have **block** and **stream** modes

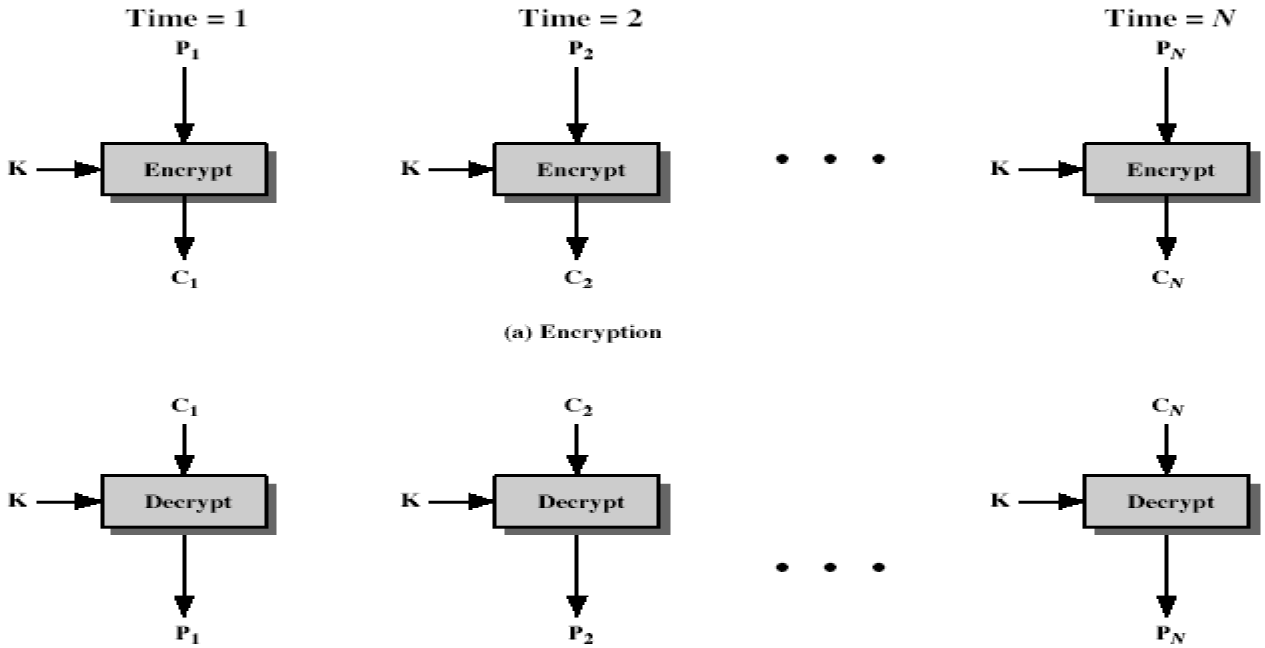
Electronic Codebook Book (ECB)

- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks

$$C_i = DES_{K1}(P_i)$$

- uses: secure transmission of single values

Electronic Codebook Book (ECB)



(a) Encryption

(b) Decryption

Advantages and Limitations of ECB

- message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness is due to the encrypted message blocks being independent
- main use is sending a few blocks of data

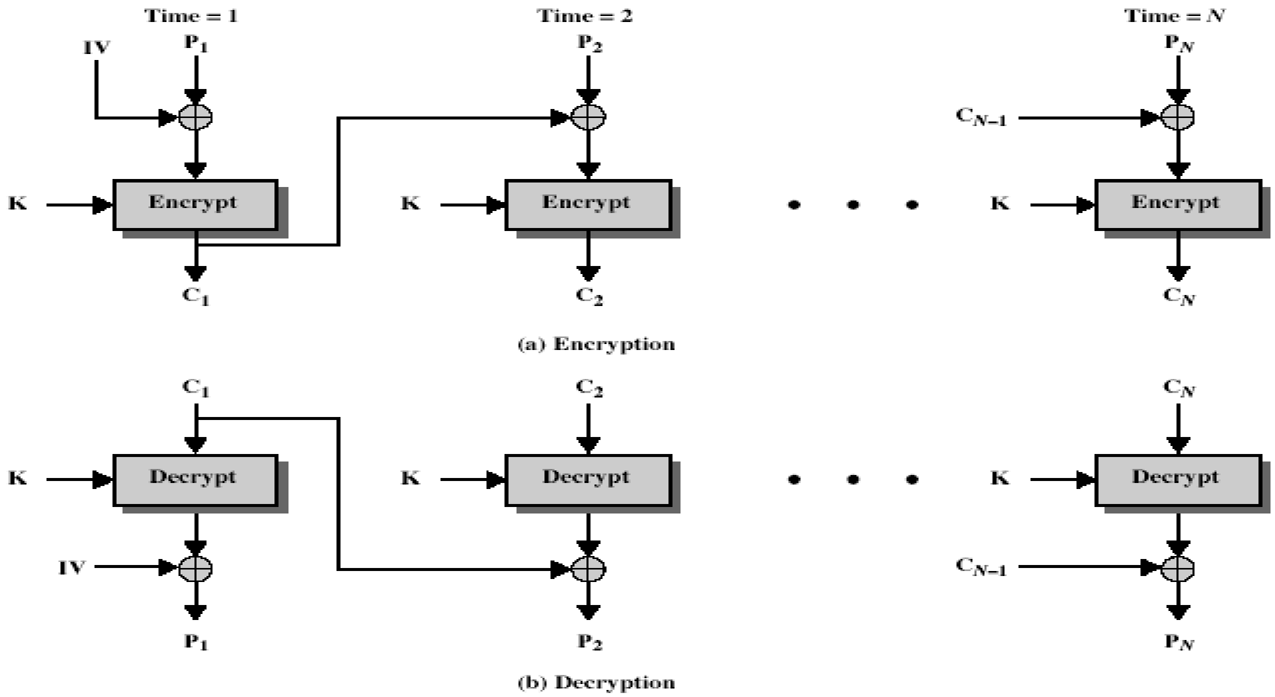
Cipher Block Chaining (CBC)

- message is broken into blocks
- linked together in encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

  $C_i = DES_{K1}(P_i \text{ XOR } C_{i-1})$

  $C_{-1} = IV$
- uses: bulk data encryption, authentication

Cipher Block Chaining (CBC)



Message Padding

- at end of message must handle a possible last short block
  - which is not as large as blocksize of cipher
  - pad either with known non-data value (eg nulls)
  - or pad last block along with count of pad size
    - eg. [b1 b2 b3 0 0 0 0 5]
    - means have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks
- need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value (as in EFTPOS)
  - or must be sent encrypted in ECB mode before rest of message

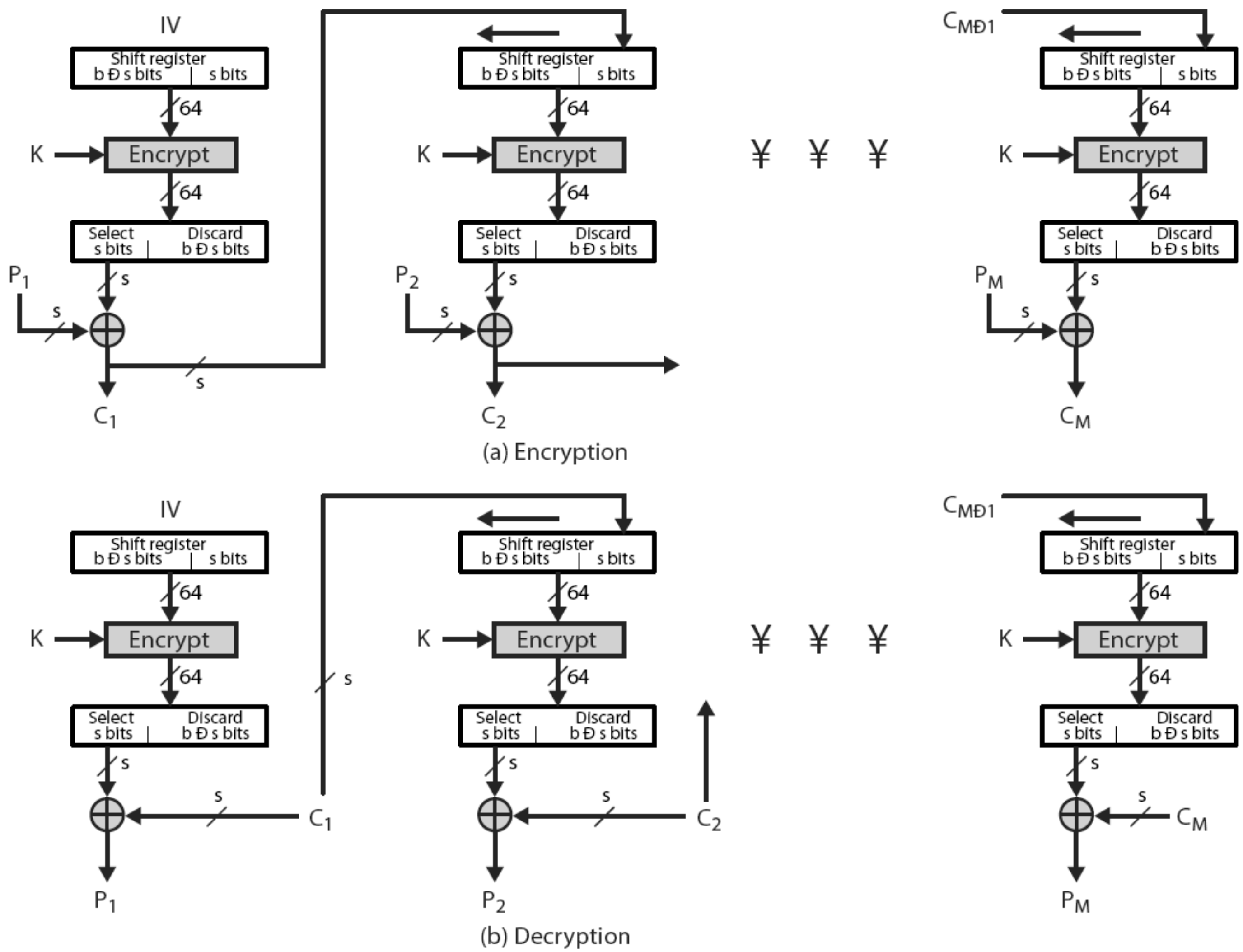
Cipher FeedBack (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)

  $C_i = P_i \text{ XOR } DES_{K1}(C_{i-1})$

  $C_{-1} = IV$
- uses: stream data encryption, authentication

Cipher FeedBack (CFB)



Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- limitation is need to stall while do block encryption after every n-bits
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propogate for several blocks after the error

Output FeedBack (OFB)

- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance

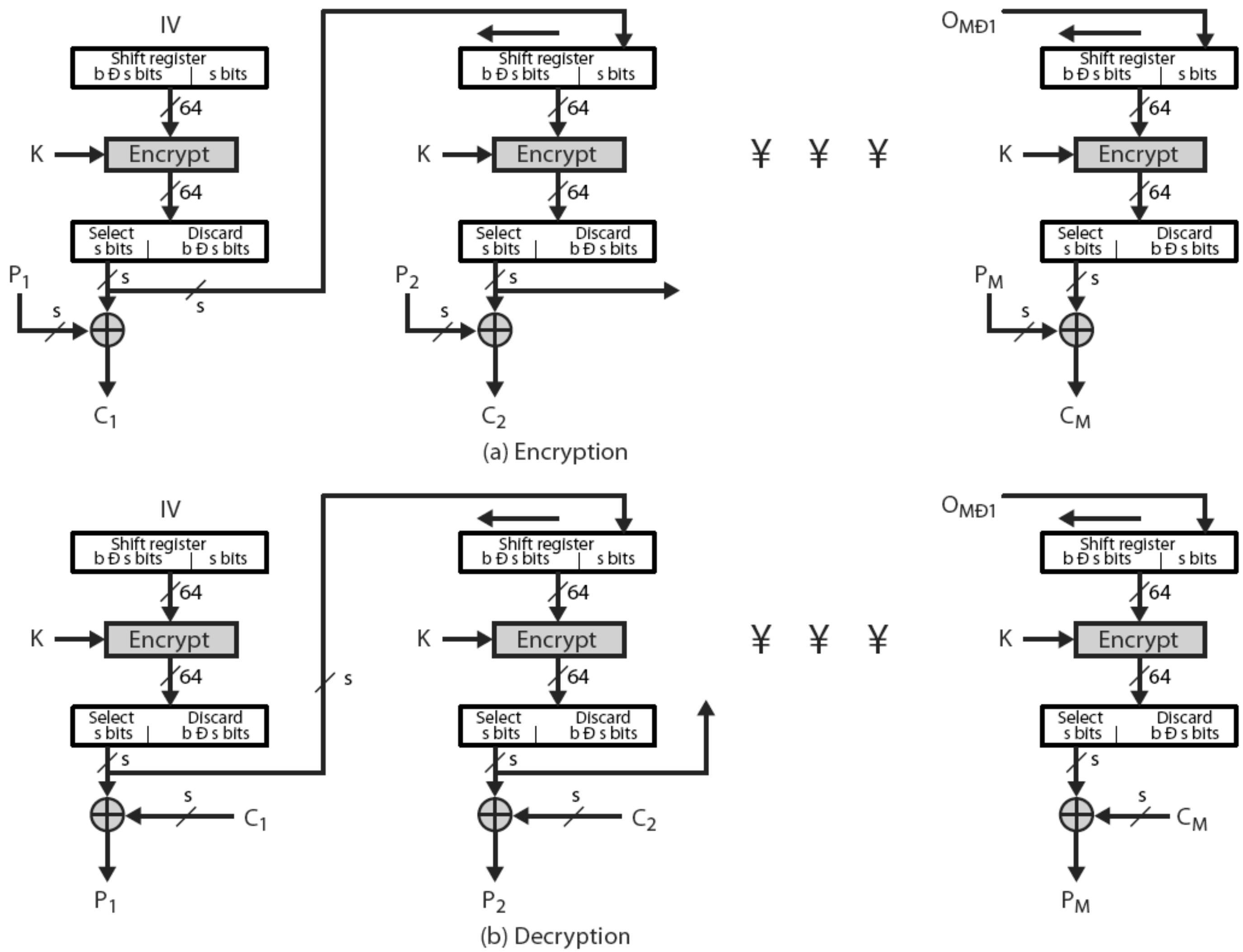
  $C_i = P_i \text{ XOR } O_i$

  $O_i = DES_{K1}(O_{i-1})$

  $O_{-1} = IV$

- uses: stream encryption on noisy channels

Output FeedBack (OFB)



Advantages and Limitations of OFB

- bit errors do not propagate
- more vulnerable to message stream modification
- a variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- originally specified with m-bit feedback
- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

Counter (CTR)

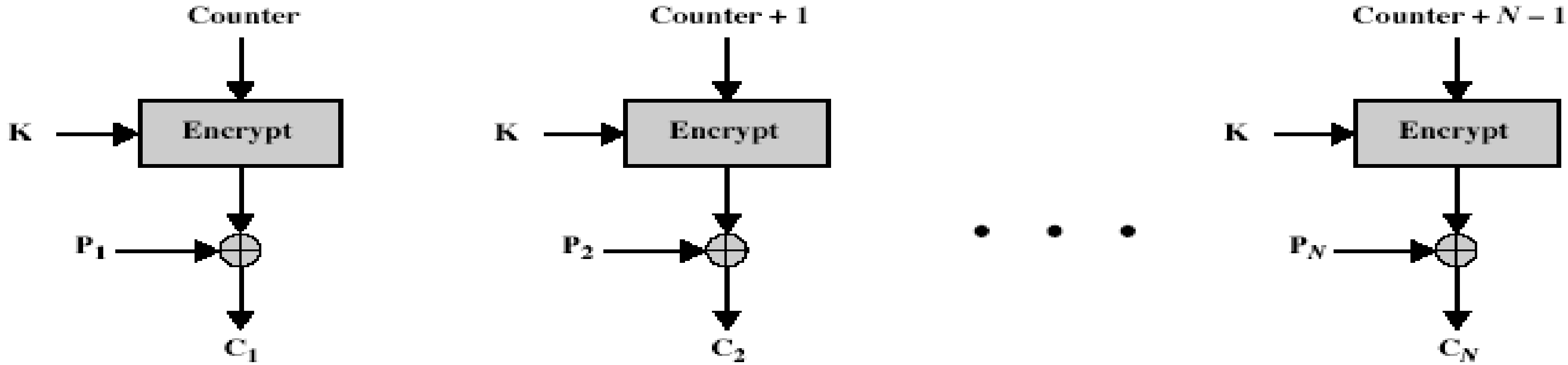
- a "new" mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

$$C_i = P_i \text{ XOR } O_i$$

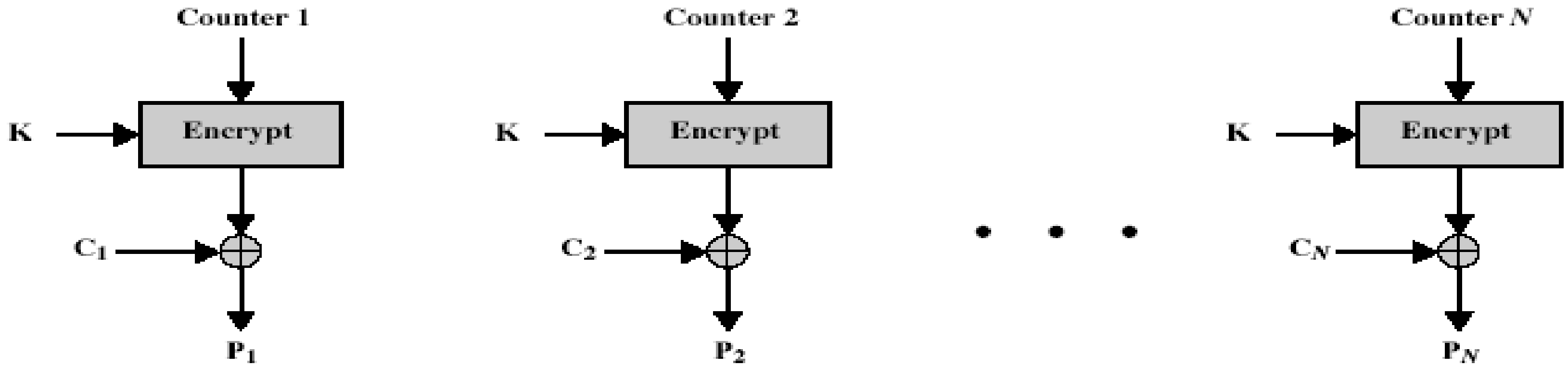
$$O_i = DES_{K1}(i)$$

- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption



(b) Decryption

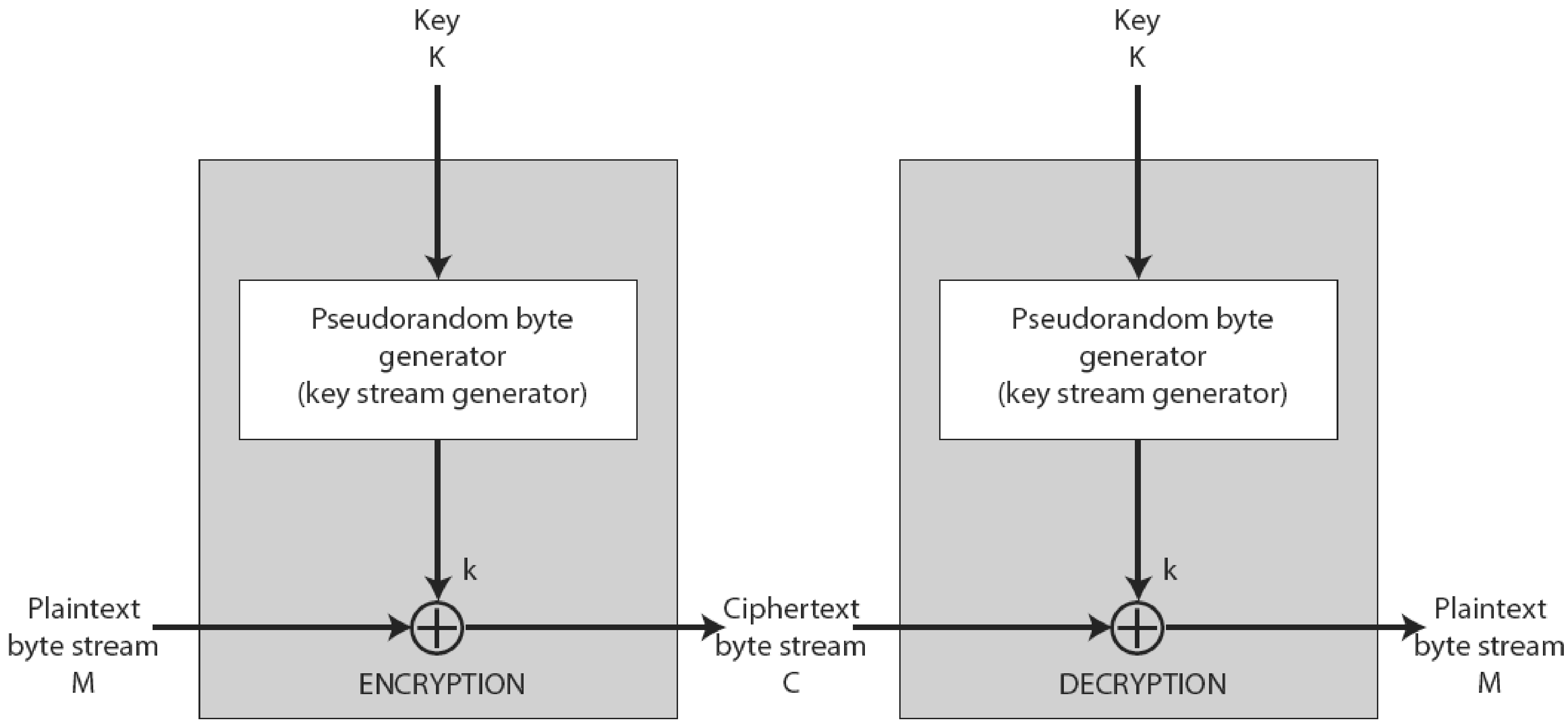
Advantages and Limitations of CTR

- 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)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i \text{ XOR } StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



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
- but usually simpler & faster

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher

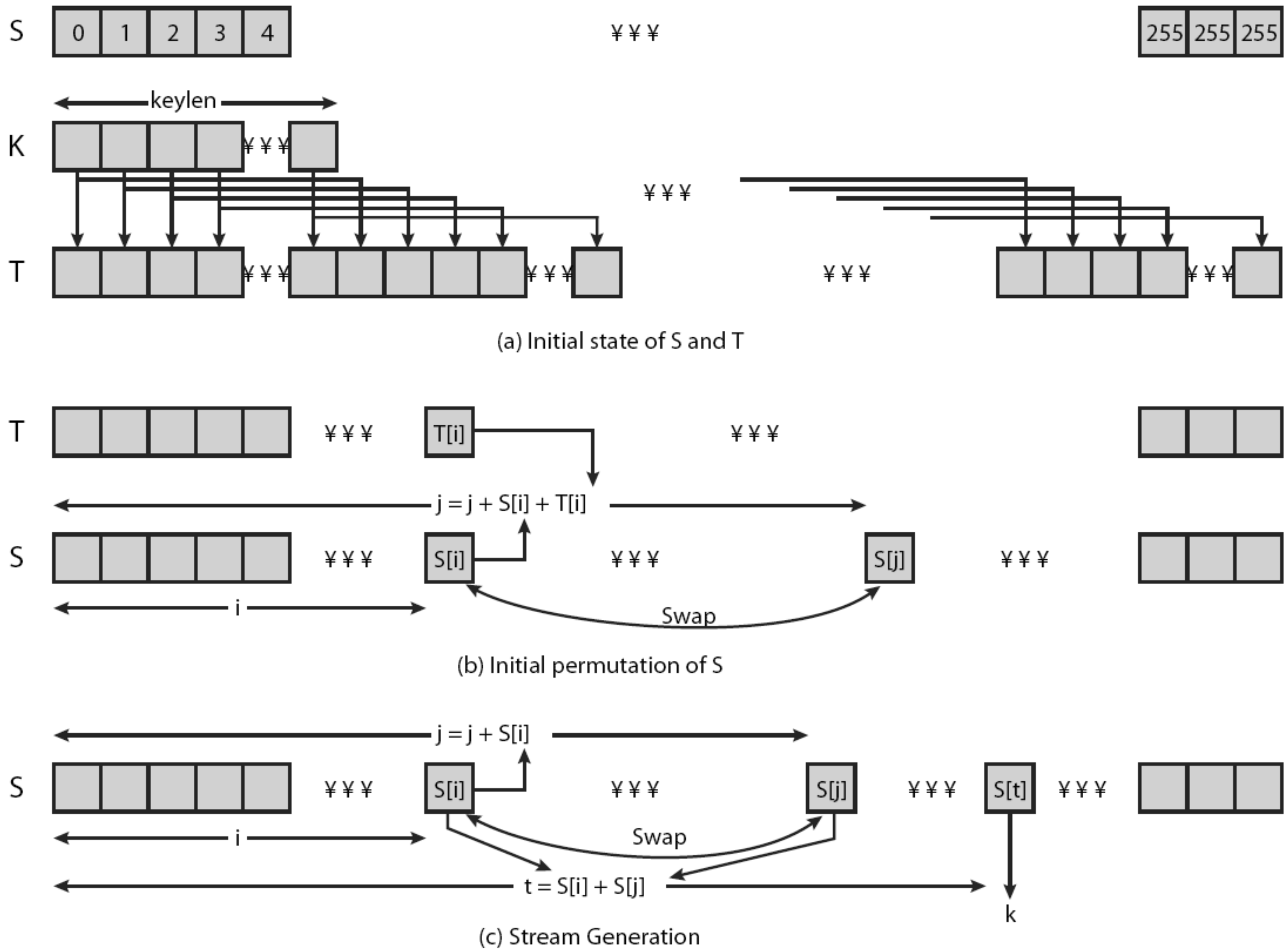
```
for i = 0 to 255 do
   S[i] = i
   T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
   j = (j + S[i] + T[i]) (mod 256)
   swap (S[i], S[j])
```

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
for each message byte Mi
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    Ci = Mi XOR S[t]
```

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**
- have a concern with WEP, but due to key handling rather than RC4 itself

AES ENCRYPTION Origins

- clear a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug-99
- Rijndael was selected as the AES in Oct-2000
- issued as FIPS PUB 197 standard in Nov-2001

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics
- final criteria
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

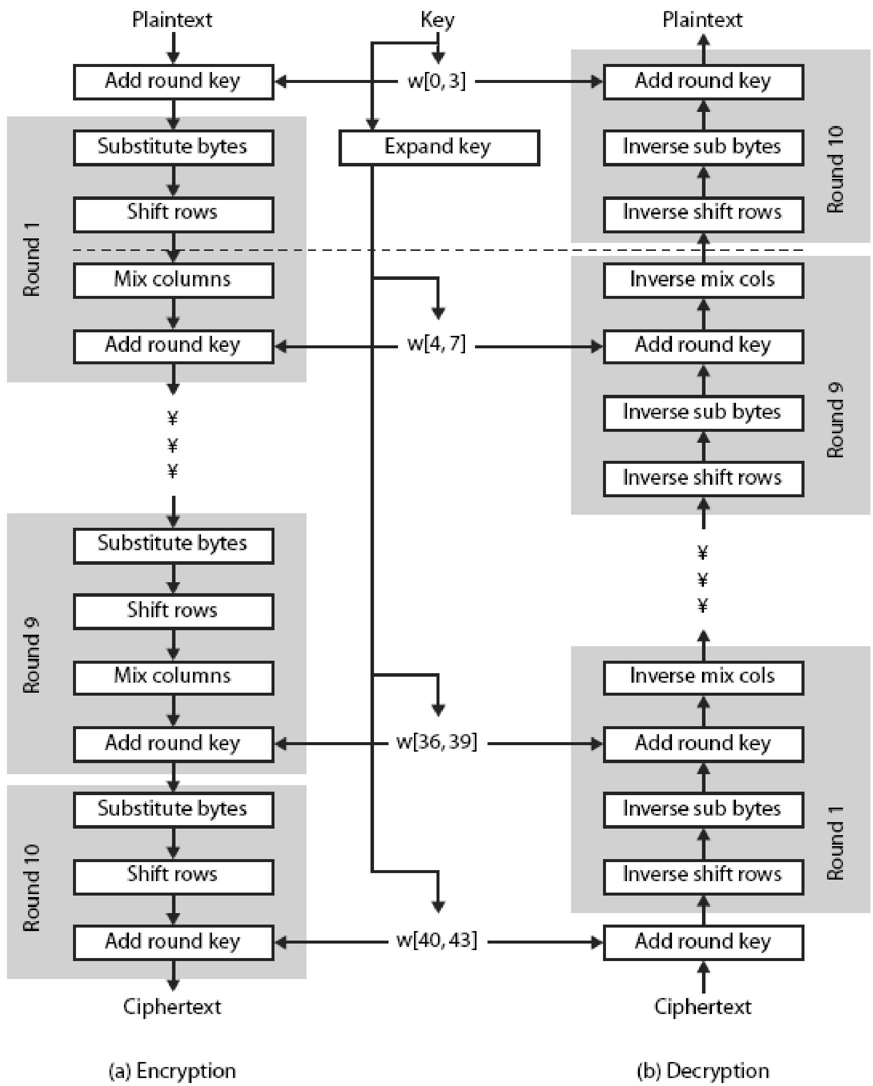
The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Rijndael

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multipy of groups)
  - add round key (XOR state with key material)
  - view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation

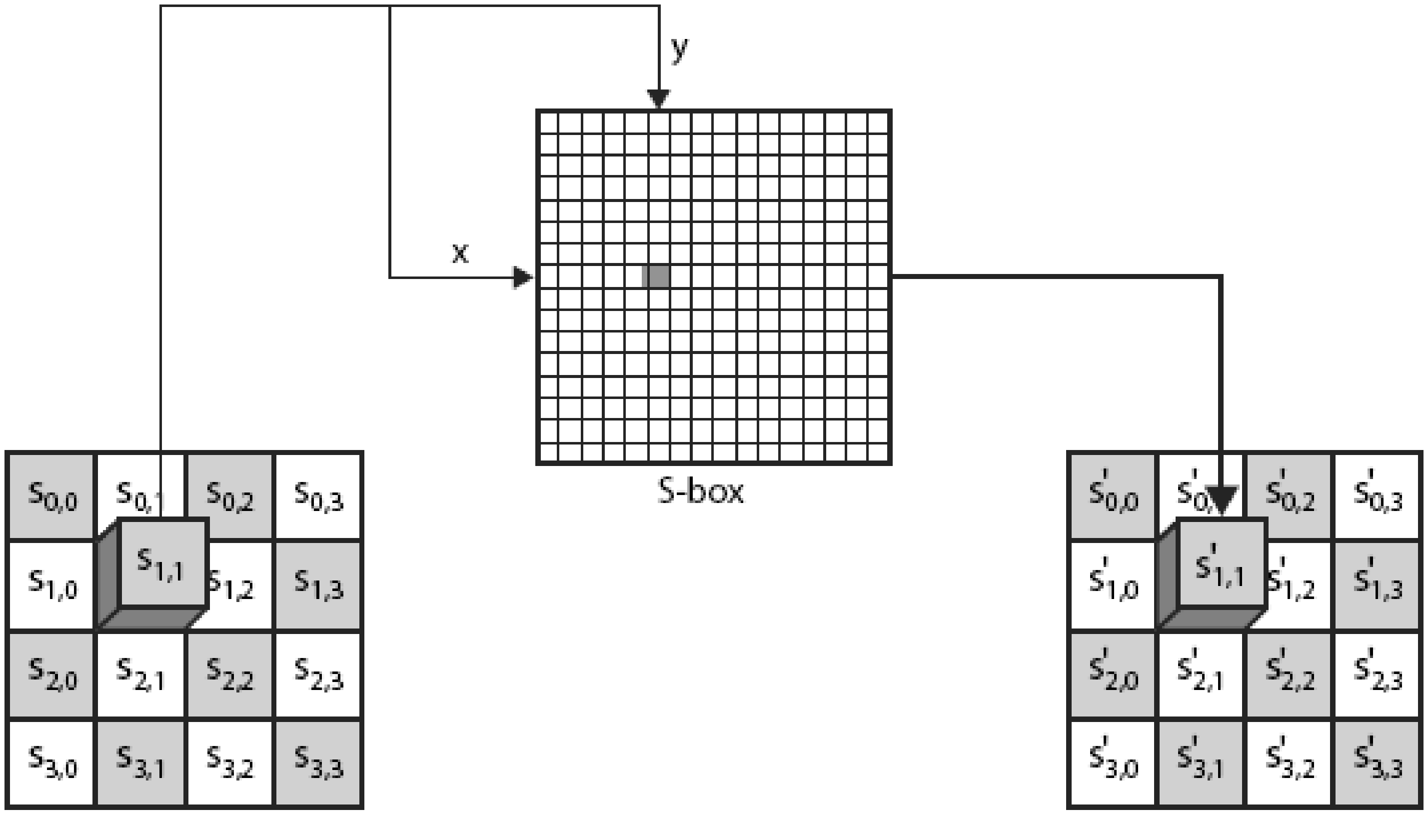
Rijndael


Plaintext
Key
Plaintext
Add round key
w[0, 3]
Add round key
Substitute bytes
Expand key
Inverse sub bytes
Shift rows
Inverse shift rows
Round 10
Round 1
Mix columns
Inverse mix cols
Add round key
w[4, 7]
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
¥
¥
¥
Substitute bytes
Shift rows
Round 9
Mix columns
Inverse mix cols
Add round key
w[36, 39]
Add round key
Substitute bytes
Inverse sub bytes
Shift rows
Inverse shift rows
Round 10
Round 1
Add round key
w[40, 43]
Add round key
Ciphertext
Ciphertext
(a) Encryption
(b) Decryption

Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks

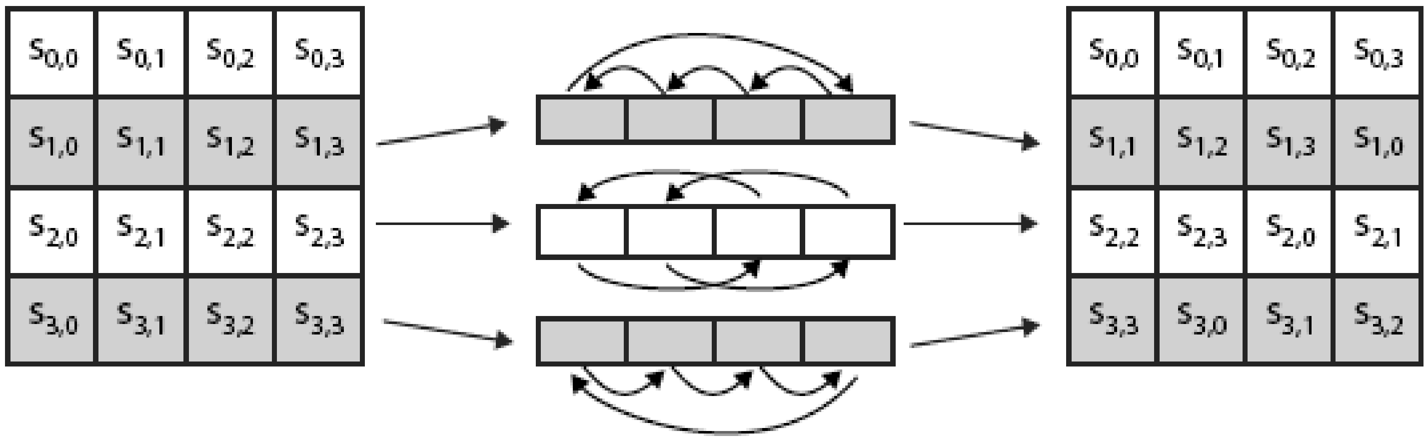
Byte Substitution



Shift Rows

- a circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Shift Rows



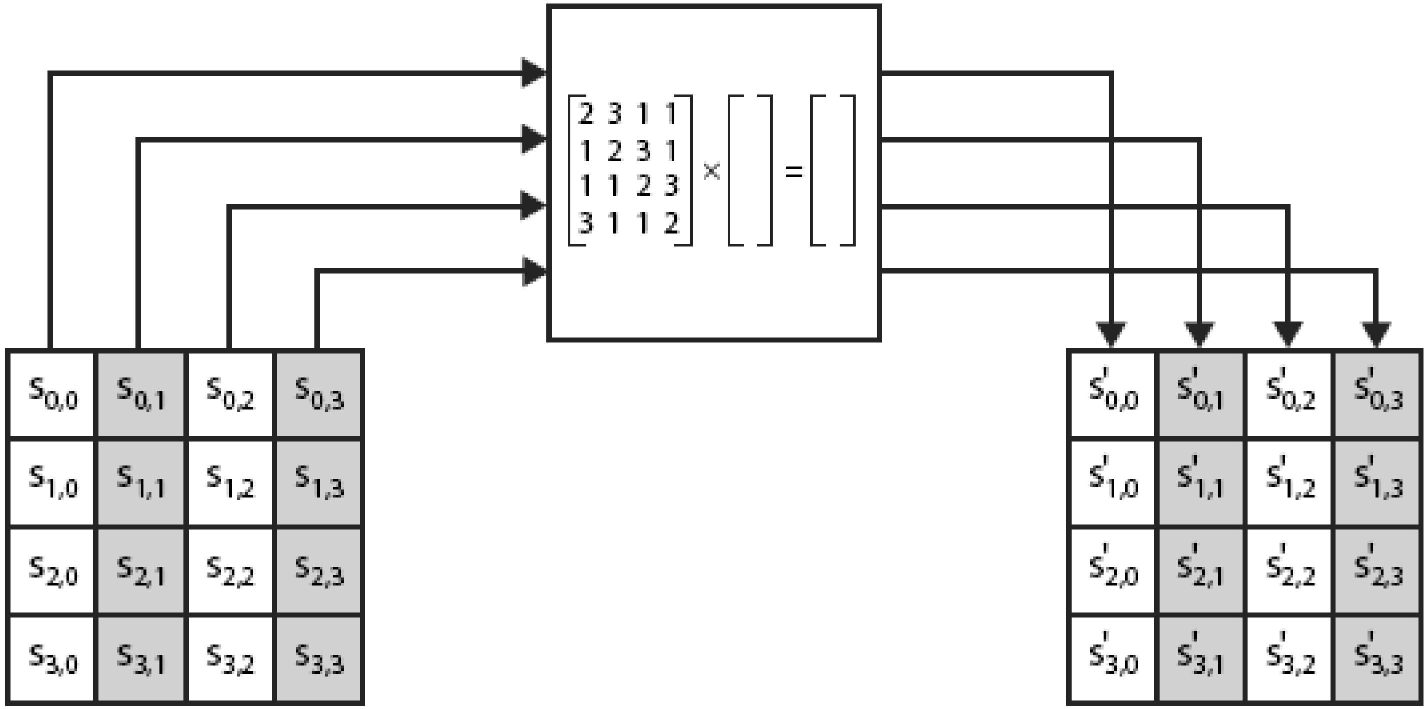
s0,0 s0,1 s0,2 s0,3
s1,0 s1,1 s1,2 s1,3
s2,0 s2,1 s2,2 s2,3
s3,0 s3,1 s3,2 s3,3
s0,0 s0,1 s0,2 s0,3
s1,1 s1,2 s1,3 s1,0
s2,2 s2,3 s2,0 s2,1
s3,3 s3,0 s3,1 s3,2

Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly m(x) $=x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Mix Columns



2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
s0,0 s0,1 s0,2 s0,3
s1,0 s1,1 s1,2 s1,3
s2,0 s2,1 s2,2 s2,3
s3,0 s3,1 s3,2 s3,3
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

Mix Columns

- can express each col as 4 equations
  - to derive each new byte in col
- decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder
- have an alternate characterisation
  - each column a 4-term polynomial
  - with coefficients in $GF(2^8)$
  - and polynomials multiplied modulo (x^4+1)

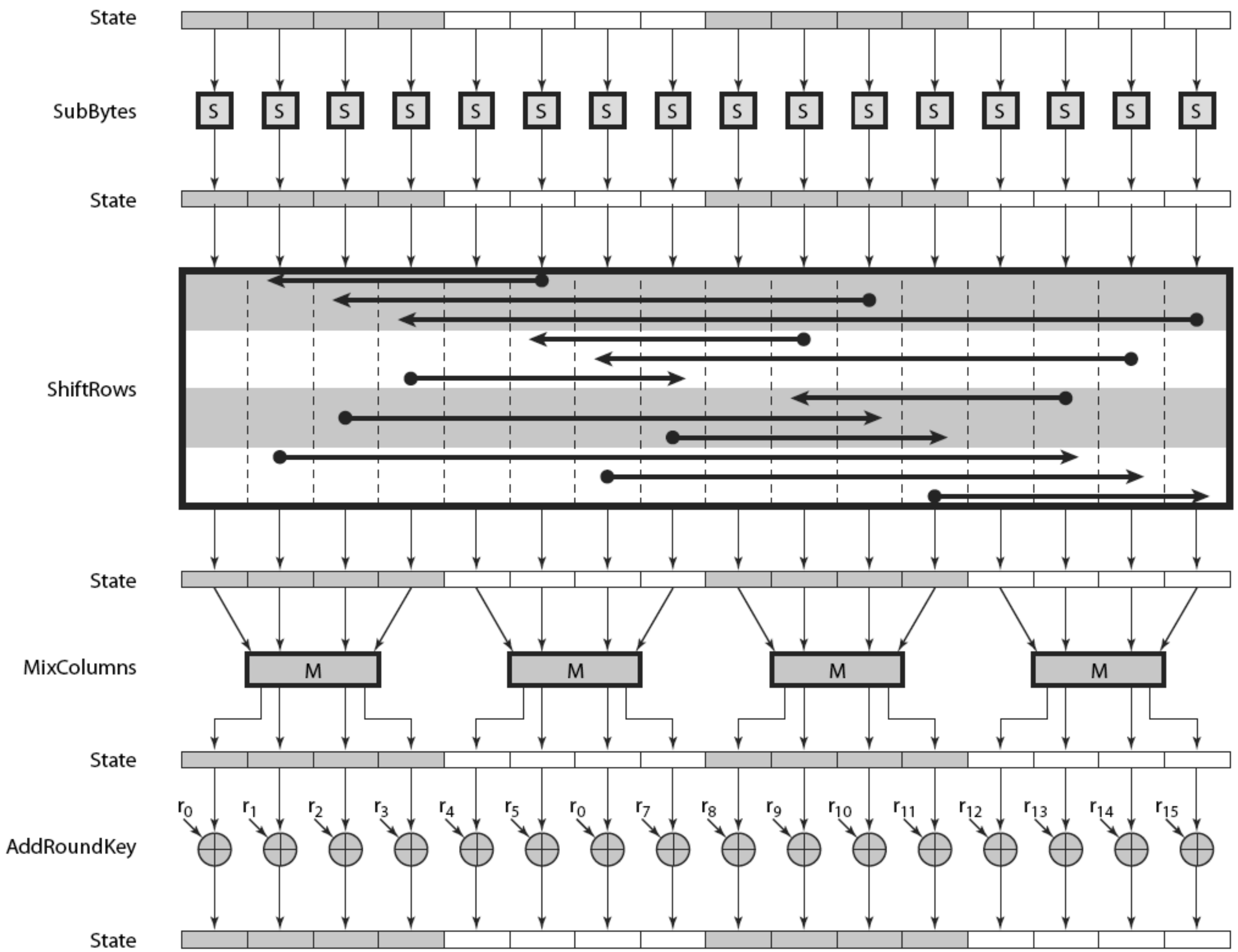
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security

Add Round Key

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

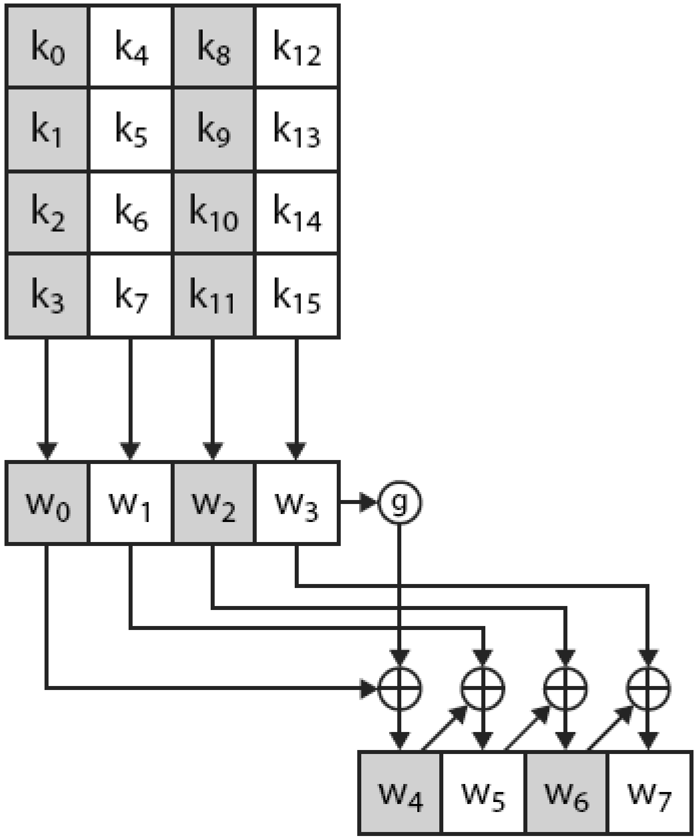
AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r0 r1 r2 r3 r4 r5 r0 r7 r8 r9 r10 r11 r12 r13 r14 r15
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1^{st} word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4^{th} back

AES Key Expansion



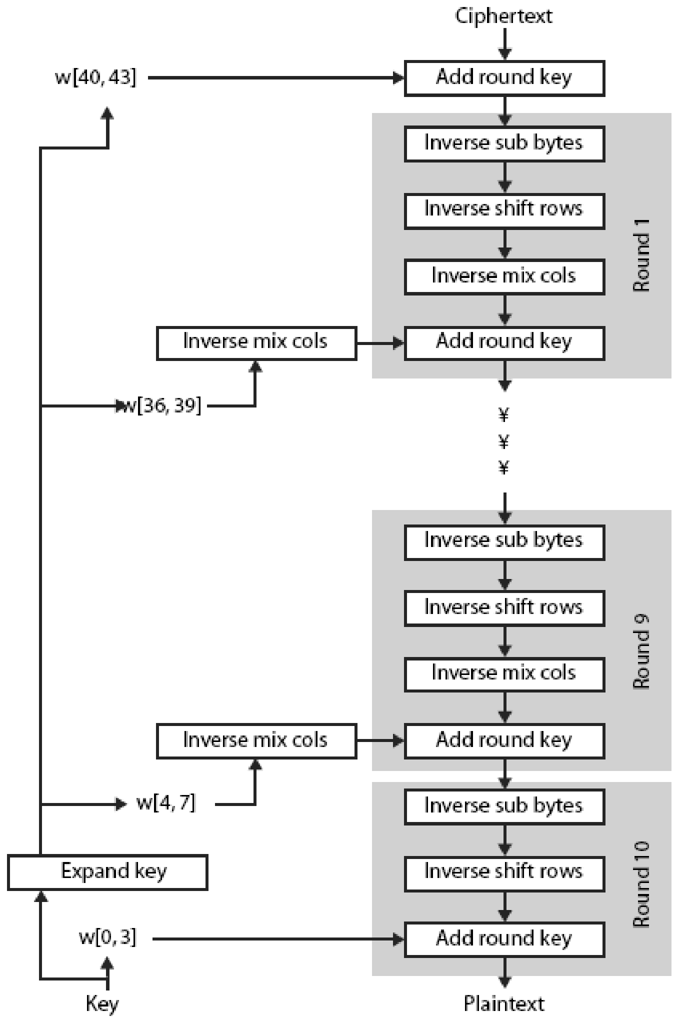
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU’s
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

AES Decryption



Implementation Aspects

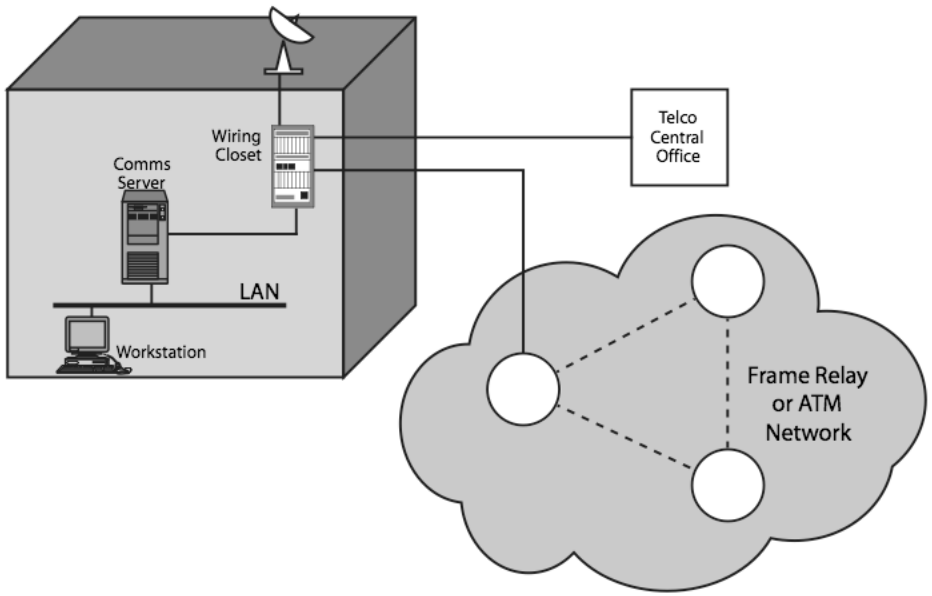
- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR's
  - mix columns requires matrix multiply in $GF(2^8)$ which works on byte values, can be simplified to use table lookups & byte XOR's

Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Confidentiality using Symmetric Encryption

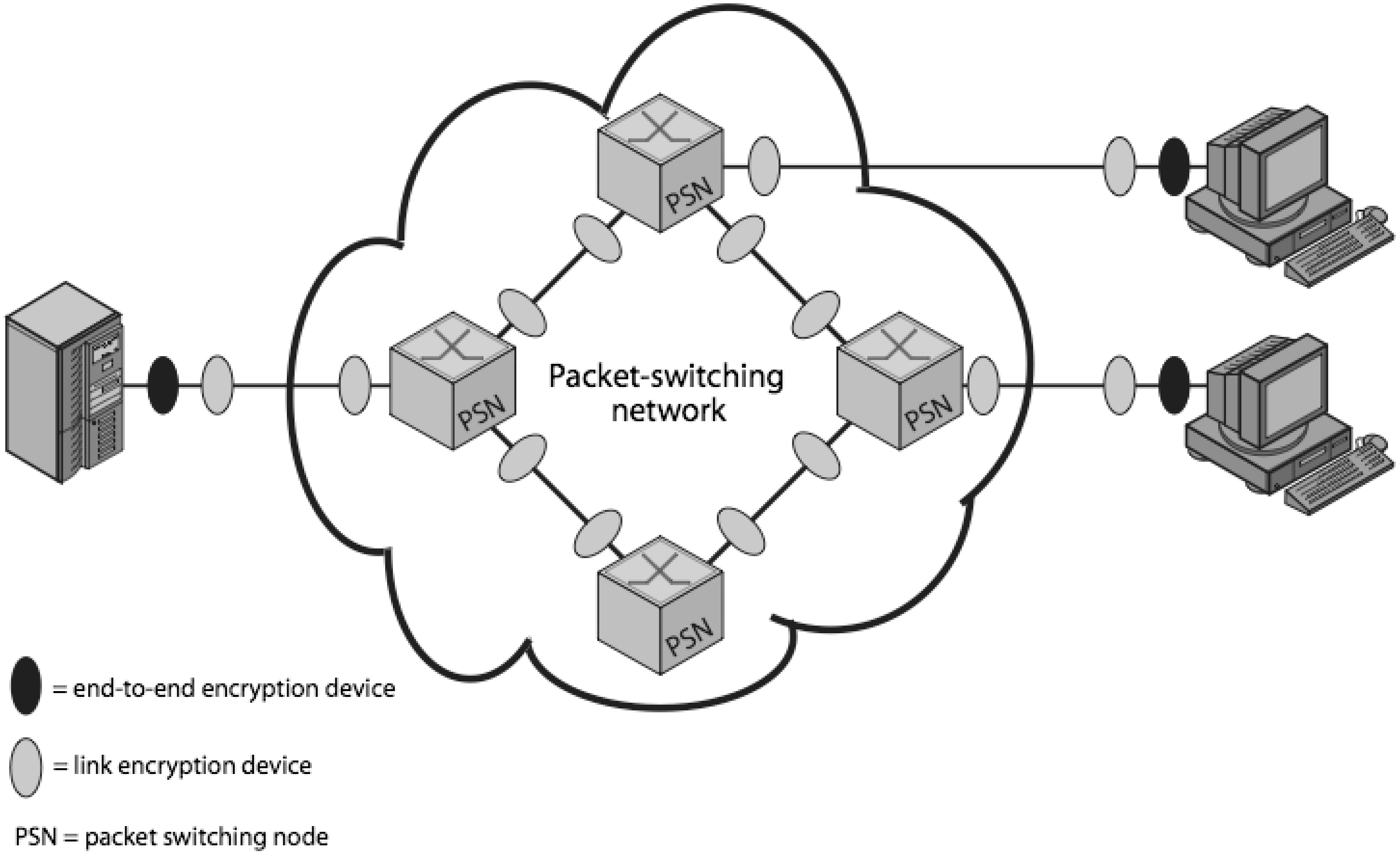
- traditionally symmetric encryption is used to provide message confidentiality



Placement of Encryption

- have two major placement alternatives
- **link encryption**
  - encryption occurs independently on every link
  - implies must decrypt traffic between links
  - requires many devices, but paired keys
- **end-to-end encryption**
  - encryption occurs between original source and final destination
  - need devices at each end with shared keys

Placement of Encryption



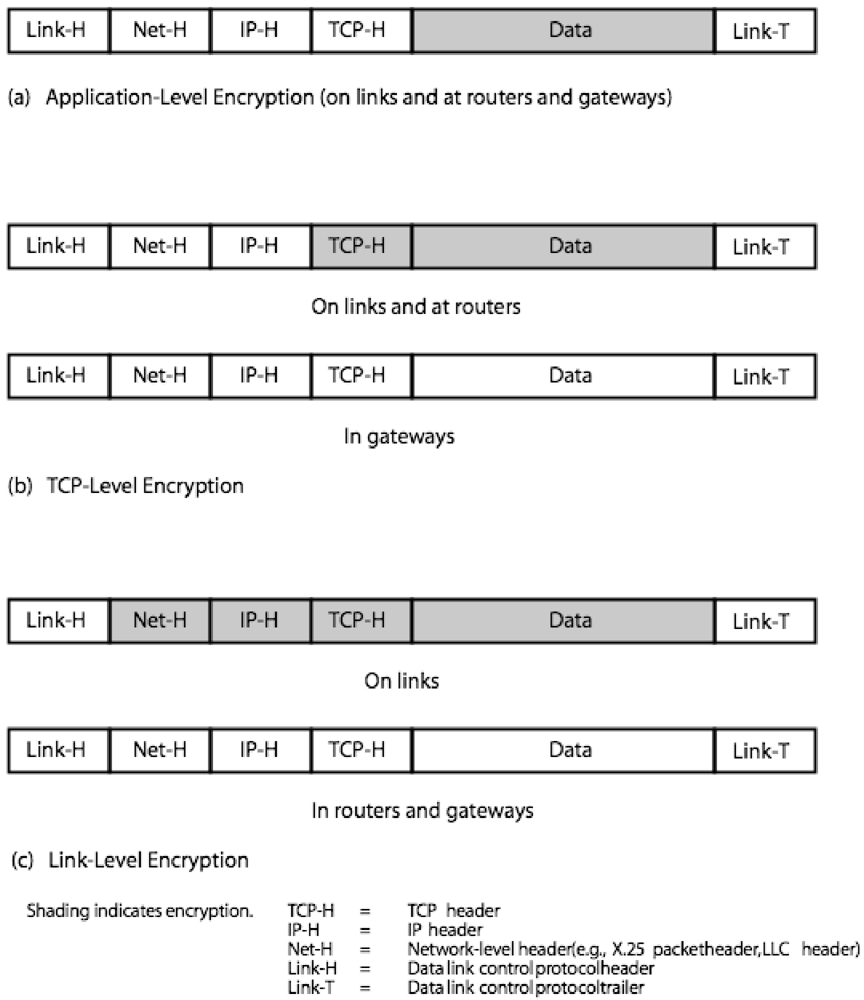
Placement of Encryption

- when using end-to-end encryption must leave headers in clear
  - so network can correctly route information
- hence although contents protected, traffic pattern flows are not
- ideally want both at once
  - end-to-end protects data contents over entire path and provides authentication
  - link protects traffic flows from monitoring

Placement of Encryption

- can place encryption function at various layers in OSI Reference Model
  - link encryption occurs at layers 1 or 2
  - end-to-end can occur at layers 3, 4, 6, 7
  - as move higher less information is encrypted but it is more secure though more complex with more entities and keys

Encryption vs Protocol Level



Traffic Analysis

- is monitoring of communications flows between parties
  - useful both in military & commercial spheres
  - can also be used to create a covert channel
- link encryption obscures header details
  - but overall traffic volumes in networks and at end-points is still visible
- traffic padding can further obscure flows
  - but at cost of continuous traffic

Key Distribution

- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

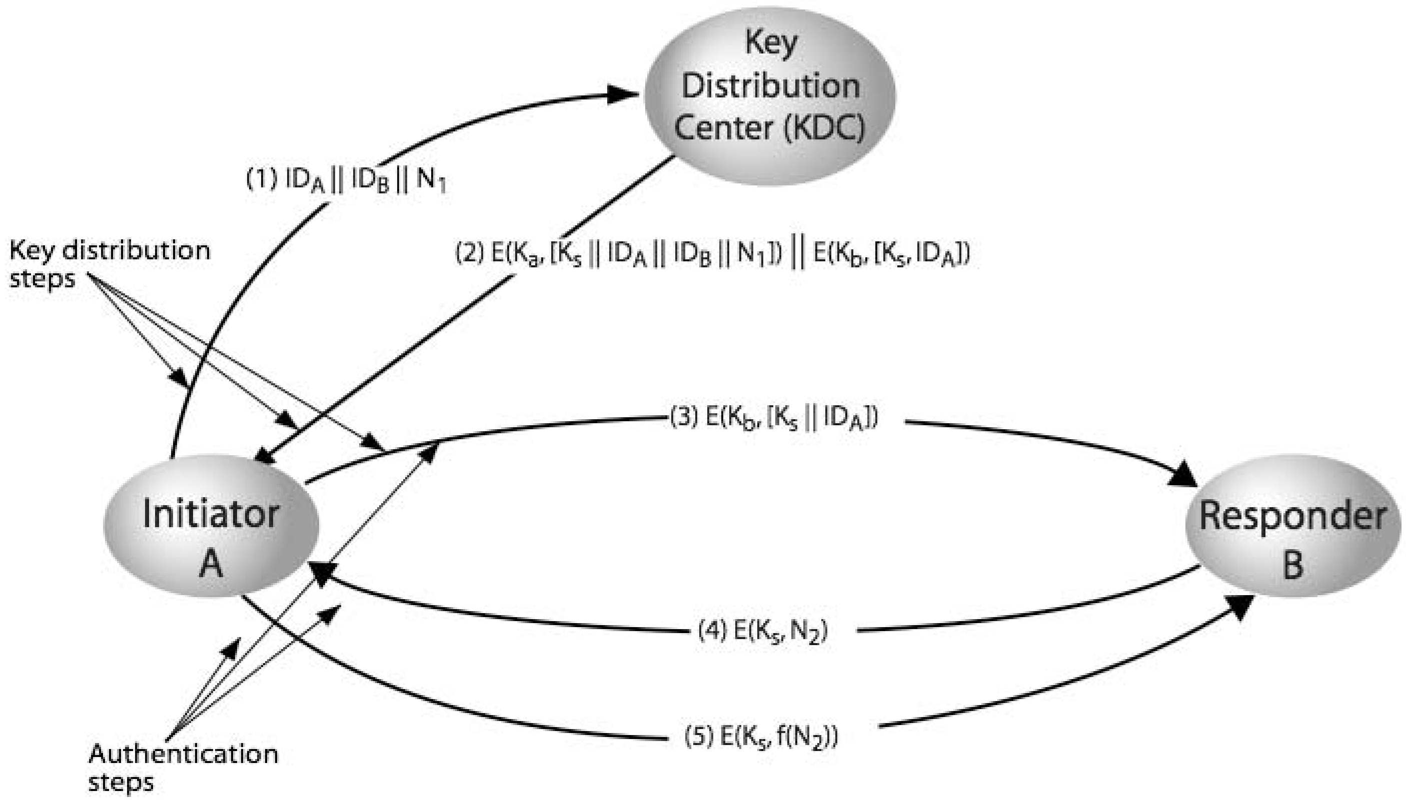
Key Distribution

- given parties A and B have various **key distribution** alternatives:
  1. A can select key and physically deliver to B
  2. third party can select & deliver key to A & B
  3. if A & B have communicated previously can use previous key to encrypt a new key
  4. if A & B have secure communications with a third party C, C can relay key between A & B

Key Hierarchy

- typically have a hierarchy of keys
- session key
  - temporary key
  - used for encryption of data between users
  - for one logical session then discarded
- master key
  - used to encrypt session keys
  - shared by user & key distribution center

Key Distribution Scenario



Key Distribution Issues

- hierarchies of KDC’s required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling key usage