Computer and Network Security

American University in Paris
Prof. Antonio Kung
Spring 2009

Syllabus Detail

- Course objectives:
  - learn the fundamentals of computer security
    - Principles of computer security
    - Basic cryptography
    - Authentication
    - Secure network protocols (Kerberos, SSL)
    - Program security
      - » Bug exploits
      - » Malicious code: viruses, worms, trojan horses, …
    - Attacks and defenses on computer systems
      - » Firewalls
      - » Intrusion detection
      - » Countermeasures
    - Trusted operating systems
    - Smart cards
    - Security evaluation
- Text book:
  - Cryptography and Network Security: Principles and Practice, Prentice Hall, third edition, by William Stallings, 2003
- Slides are a variant of Lecture slides by Lawrie Brown

Note that many slides were derived from lecture slides by Lawrie Brown

Chapter 1

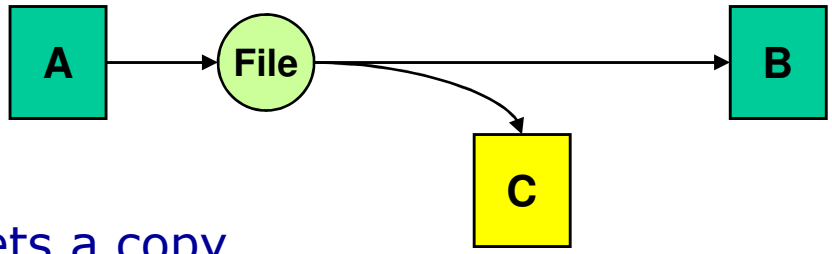
Overview

Some Terms

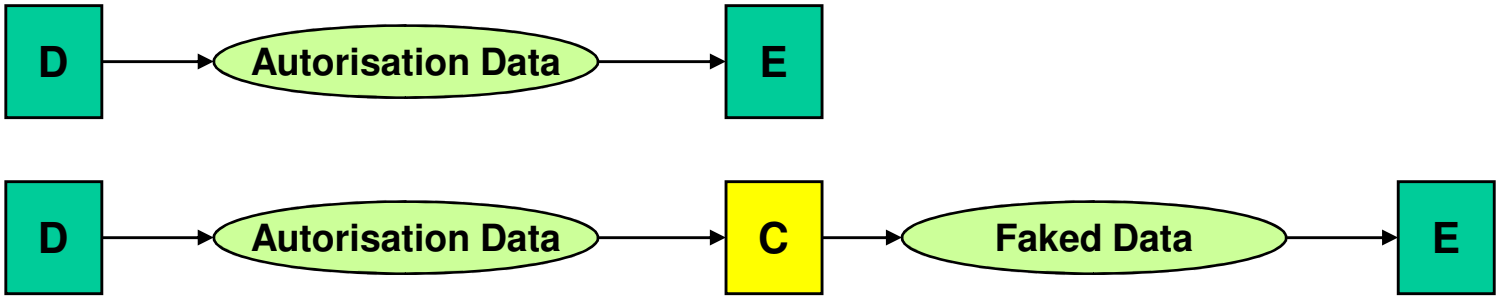
- Information security
- Computer security
- Network security
- Internet security

Examples of Security Breaches

- Getting a file
  - A transmits a file to B
  - File contains payroll data
  - C monitors transmission and gets a copy



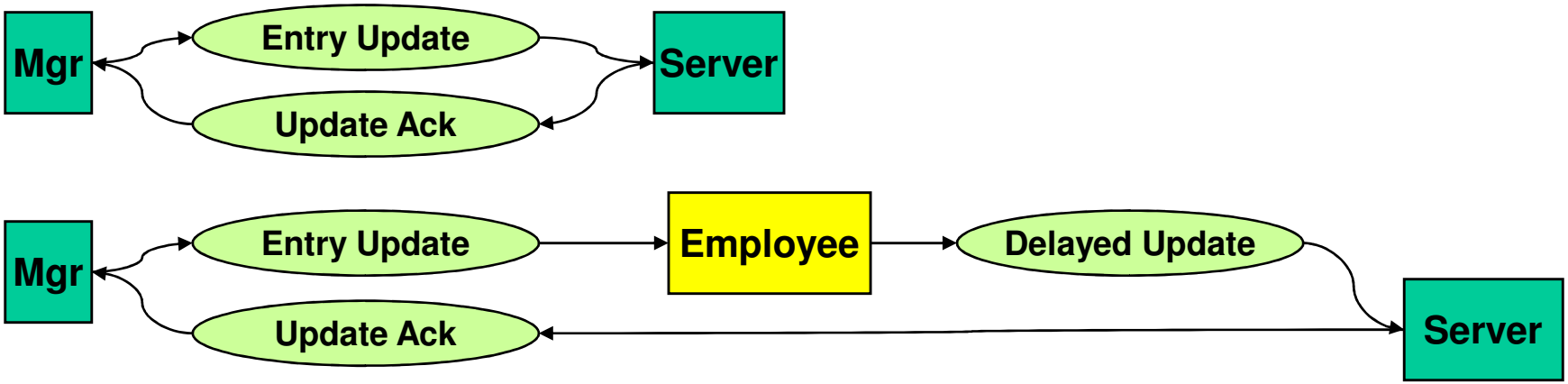
- Changing authorisation
  - Network manager D updates an authorisation file in computer E
  - F intercepts and changes file



Example of Security Breaches

- Faking the sender
  - F sends a message to E and tells him he is D

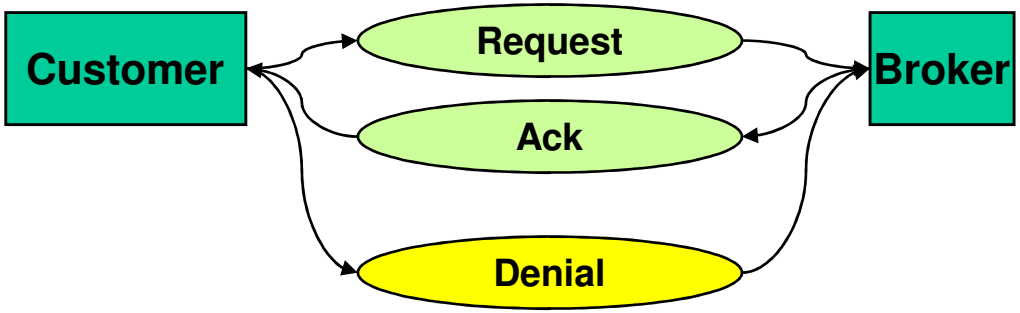
- Firing an employee who takes action
  - Manager sends request to server to delete employee account
  - Employee intercepts and delays the deletion



Note that many slides were derived from lecture slides by Lawrie Brown

Examples of Security Breaches

- Customer denies request
  - customer makes a request to buy shares
  - shares loses value
  - customer denies having made the request



Analysis: Security Covers many Aspects

- There are user requirements
  - confidentiality
  - privacy
  - authentication (is the real person behind?)
  - integrity (content not modified)
- There are business requirements
  - confidentiality
  - authentication
  - non repudiation (cannot deny action)
  - integrity
- Security analysis typically focuses on « attacks » viewpoint
  - what are the possible attacks?
  - what are the assets you want to protect?

Security Analysis

- Objective is to understand/evaluate the system in terms of security
- There are threats and Attacks
  - Threats: potential for violation of security
  - Attacks: assault on system security that derives from a threat
    - gain unauthorised access
    - virus
    - denial of service attack
    - fraudulently authorising transactions

Security Analysis

- There are services and mechanisms
  - Services: set of mechanisms to enhance security
    - mimick security services associated with physical assets
      - e.g. physical documents
        - » signature
        - » protection from disclosure - sealing, tampering, destruction
        - » notarization, witnesses
        - » recording
      - similar services for virtual documents
  - Mechanism: detect, prevent or recover from a security attack
    - e.g. cryptographic techniques

Security Services according to X.800

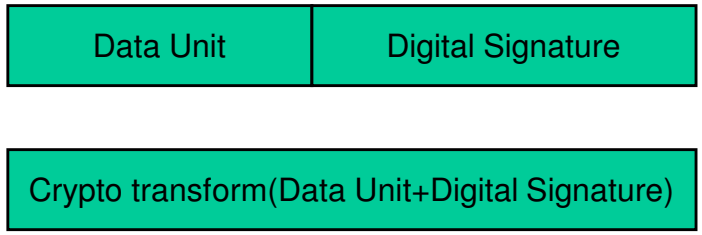
- ITU-T2
  - International Telecommunication Union - Telecommunication Standardisation Sector
  - develops recommandations related to OSI (Open Systems Interconnection)
- Services
  - Authentication
    - Assurance that the communicating entity is the one it claims to be.
      - « I am your bank »
  - Access control
    - Prevention of unauthorised used of a resource
      - « Switch on/off power in electricty station »

Security Services according to X.800

- Services
  - Data confidentiality
    - Protection of data from disclosure
  - Data integrity
    - Assurance that data received is exactly as sent (no modification, no replay)
  - Non repudiation
    - Protection against denial by one entity involved in a communication in having participated in the communication

Security Mechanisms according to X.800

- Mechanisms
  - Encypherment
    - Using math algorithms to transform data in a form that is not « intelligible ». Depends on keys
  - Digital signature
    - Data appended to a data unit of cryptographic transformation to prove source integrity of data



    - Only the sender can create a valid signature
    - Protects against forgery (e.g. by recipient)

Security Mechanisms according to X.800

- Mechanisms
  - Access control
  - Data integrity
  - Traffic padding
    - all message have the same size
    - to protect against traffic analysis
  - Routing control
    - Selects a more secure route (e.g. when a security breach has been identified)
  - Notarisation
    - Use of trusted third party to assure certain property of data exchange
  - Security audit trail
    - Log data for security audit

Passive Attacks

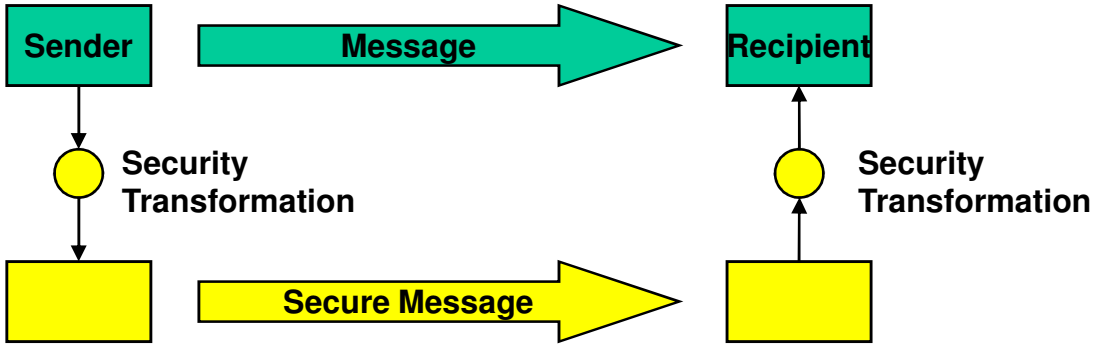
- Based on monitoring of transmissions:
  - obtain message contents
  - monitor traffic flows

Active Attacks

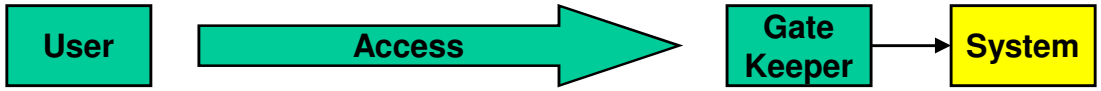
- Masquerade
  - An entity pretends to be a different entity
- Replay
  - Passive capture of data unit
  - Subsequent retransmission
- Modification of messages
  - e.g. message « allow A to do this » is changed to « allow B to do this »
- Denial of service
  - e.g. overloading a web site

Security Models

- Communication-based



- System-based



Note that many slides were derived from lecture slides by Lawrie Brown

Chapter 2

Classical Encryption Techniques

Symmetric Encryption

- Also called
  - conventional encryption
  - private-key encryption
  - single-key encryption
- 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

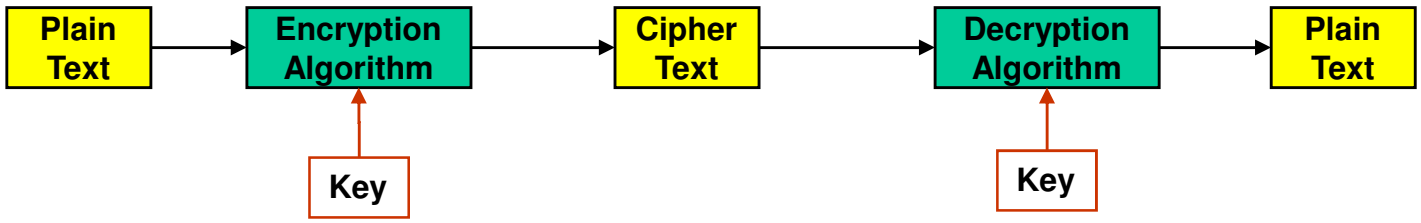
Basic Terminology

- **plaintext** - the original message (texte en clair)
- **ciphertext** - the 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)** - the study of principles/ methods of deciphering ciphertext *without* knowing key
- **cryptology** - the field of both cryptography and cryptanalysis

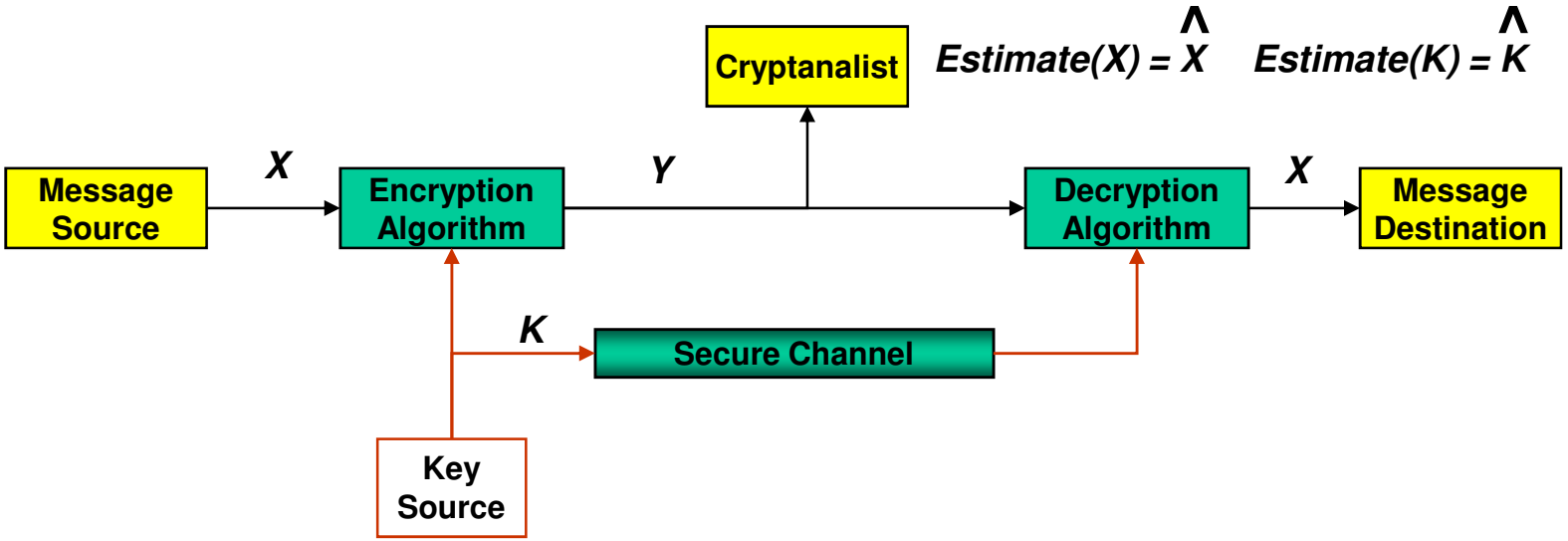
Symmetric Cipher Model

- 5 entities
  - plaintext
  - ciphertext
  - key
  - encryption algorithm
  - decryption algorithm



Requirements

- two requirements for secure use of symmetric encryption:
  - algorithms must be good: "strong encryption algorithm"
  - a secret key known only to sender / receiver
    - implies a secure channel to distribute key
    - not an easy part
    - often the weak part of security
- Model of a conventional crypto system



Cryptography

$$Y = E_K(X)$$

- Y = ciphertext
- E = encryption
- k = key
- X = plaintext

- is characterized 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

Types of Cryptanalytic Attacks

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

Brute Force Search

- Enumerate all keys
  - most basic attack, proportional to key size
  - assume either that plain text is known or can be recognised

Key size	Number	Time 1 encryption/micros	Time 10^6 encryption/micros
32	$4.3\ 10^9$	36mn	215ms
56	$7.2\ 10^{16}$	1142 y	10h
128	$3.4\ 10^{38}$	$5.4\ 10^{24}$	$5.4\ 10^{18}$
168	$3.7\ 10^{50}$	$5.9\ 10^{36}$	$5.9\ 10^{30}$

More Definitions

- unconditional security
  - no matter how much computer power 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

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
- example 1
  - replaces each letter by next letter

```
hello zoo

IFMMP APP
```

- example 2
  - replaces each letter by 3rd letter on

```
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
0  1  2  3  4  5  6  7  8  9  10 11 12
n  o  p  q  r  s  t  u  v  w  x  y  Z
13 14 15 16 17 18 19 20 21 22 23 24 25
```

- then have Caesar cipher as:

  $C = E(p) = (p + k) \bmod (26)$

  $p = D(C) = (C - k) \bmod (26)$

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
```

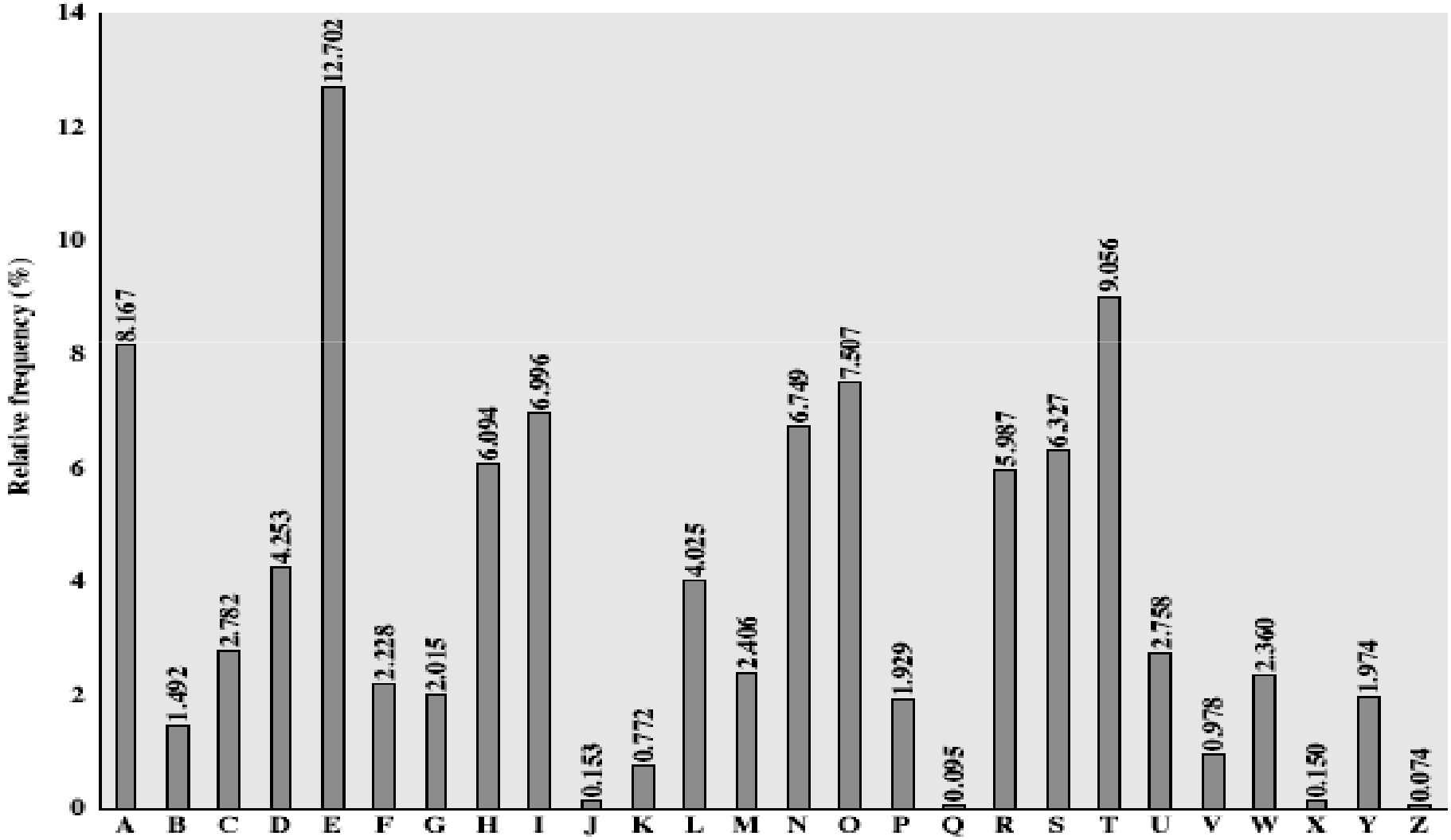
Monoalphabetic Cipher Security

- 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
- then T,R,N,I,O,A,S
- other letters are fairly rare
- cf. Z,J,K,Q,X
- have tables of single, double & triple letter frequencies

English Letter Frequencies



Note that many slides were derived from lecture slides by Lawrie Brown

Use in Cryptanalysis

- key concept - monoalphabetic substitution ciphers do not change relative letter frequencies
- discovered by Arabian scientists in 9th 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

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

```
MONAR
CHYBD
EFGIK
LPQST
UVWXZ
```

Encrypting and Decrypting

- plaintext encrypted two letters at a time:
  - if a pair is a repeated letter, insert a filler like 'X',
    - "balloon" -> "ba lx lo on"
- encryption
  - both letters in the same row
    - replace each with letter to right (with wrapping),
    - “ar" -> "RM"
  - both letters in the same column
    - replace each with the letter below it (with wrapping),
    - “mu" -> "CM"
  - otherwise
    - each letter is replaced by the one in its row in the column of the other letter of the pair,
    - “hs" -> "BP", “ea" -> "IM" or "JM" (as desired)

Security of the Playfair Cipher

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

Polyalphabetic Ciphers

- another approach to improving security is to use multiple cipher alphabets
- called **polyalphabetic substitution ciphers**
- makes 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

- Invented by Blaise de Vigenère (1523-1596)
- the **Vigenère Cipher is** the simplest polyalphabetic substitution cipher
  - effectively multiple caesar ciphers
  - key is multiple letters long K = k1 k2 ... kd
  - 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

- write the plaintext out
- write the keyword repeated above it
- use each key letter as a caesar cipher key
- encrypt the corresponding plaintext letter
- example 1 using keyword *abc*

```
key:        abcabc
plaintext:  foobar
ciphertext:GQRCCU
```

- example 2 using keyword *deceptive*

```
key:        deceptivedeceptivedeceptive
plaintext:  wearediscoveredsaveyourself
ciphertext:ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

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

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
- have problem of 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:

```
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 scheme
- 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:        4 3 1 2 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: TTNA APTM TSUO AODW COIX KNLY PETZ
```

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 product cipher
- were widely used in WW2
  - German Enigma, Allied Hagelin, Japanese Purple
- implemented a very complex, varying substitution cipher
  - used a series of cylinders
  - each cylinder gives one substitution (e.g. 26 possibilities)
  - after substitution (e.g. key pressed), cylinders are rotated, and therefore use a different substitution
- with 3 cylinders we have
  - 26^3=17576 alphabets

Steganography

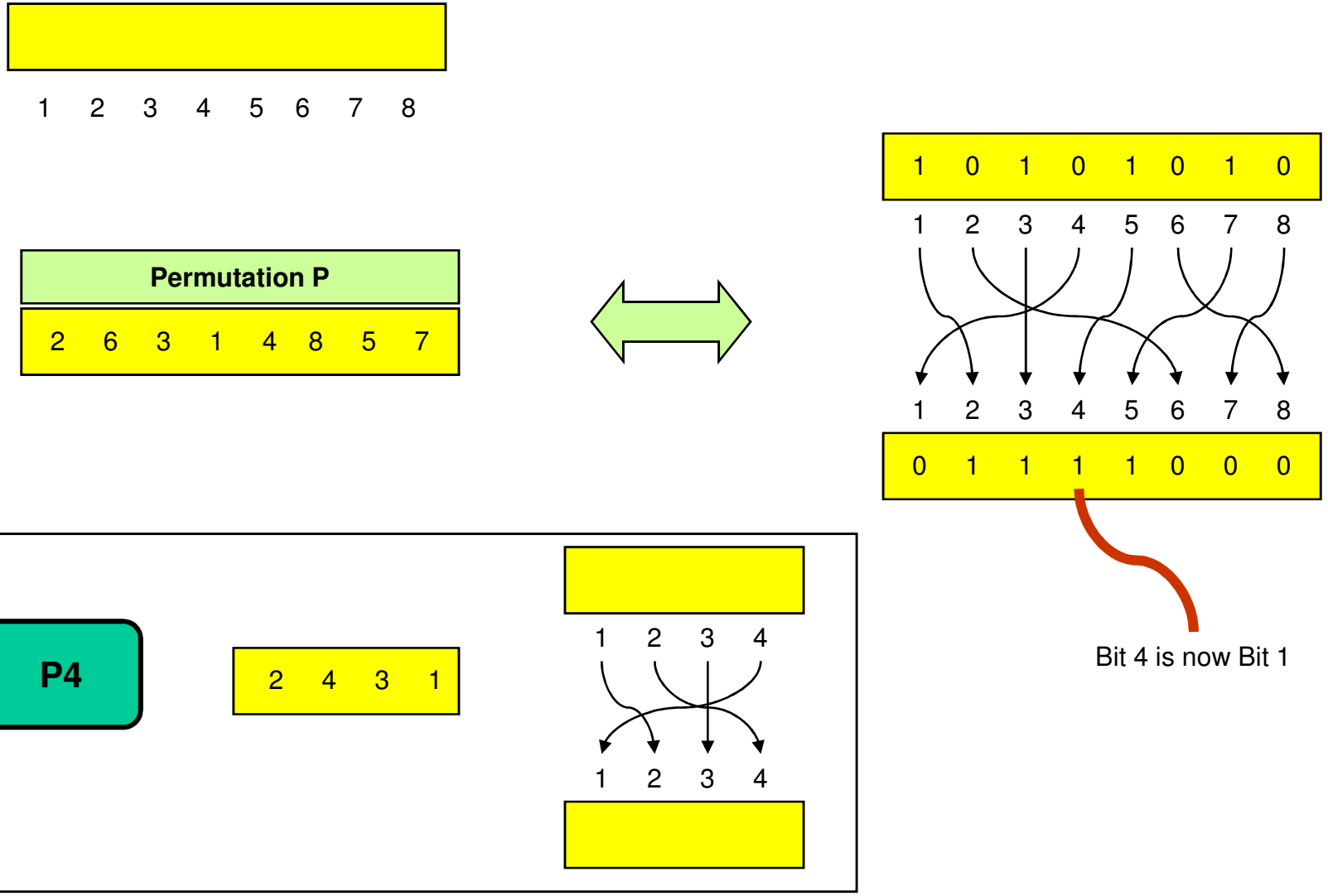
- an alternative to encryption
- hides existence of message
  - using only a subset of letters/words in a longer message marked in some way
  - using invisible ink
  - hiding in LSB in graphic image or sound file
- has drawbacks
  - high overhead to hide relatively few info bits

Chapter 3 – Block Ciphers and the Data Encryption Standard

Modern Block Ciphers

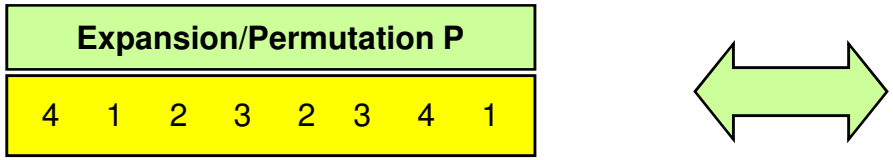
- will now look at modern block ciphers
- one of the most widely used types of cryptographic algorithms
- provide secrecy and/or authentication services
- in particular will introduce DES (Data Encryption Standard)
- will start with S-DES
  - Simple DES (introduced by Ed. Schaeffer - U. Santa Cruz)

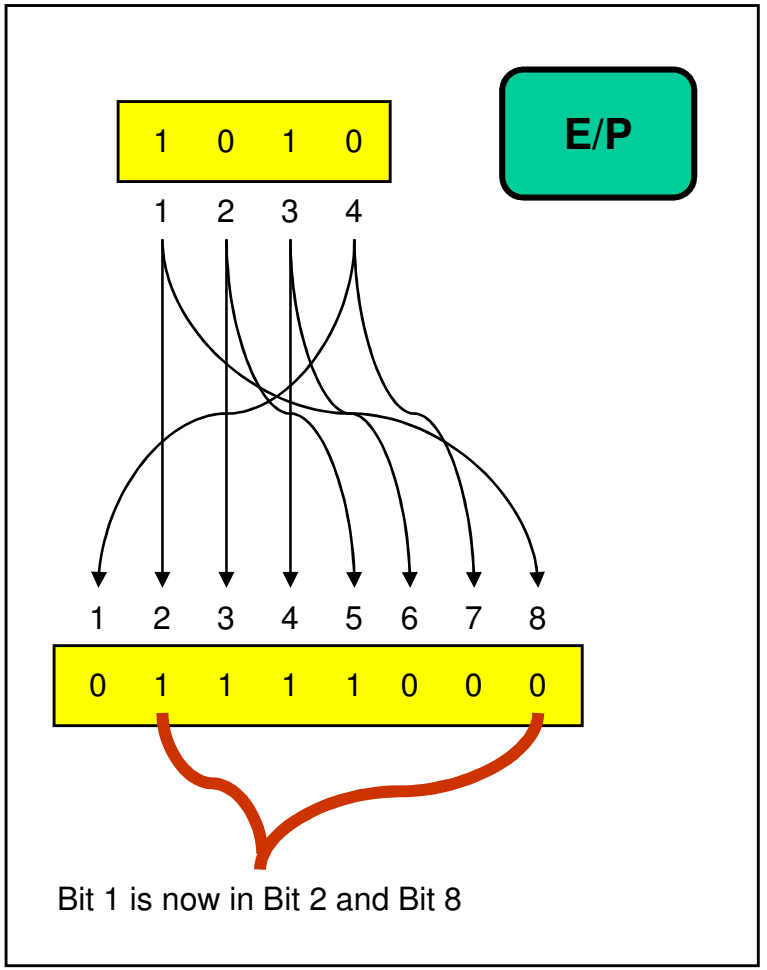
Some Basic Mechanisms: Permutation



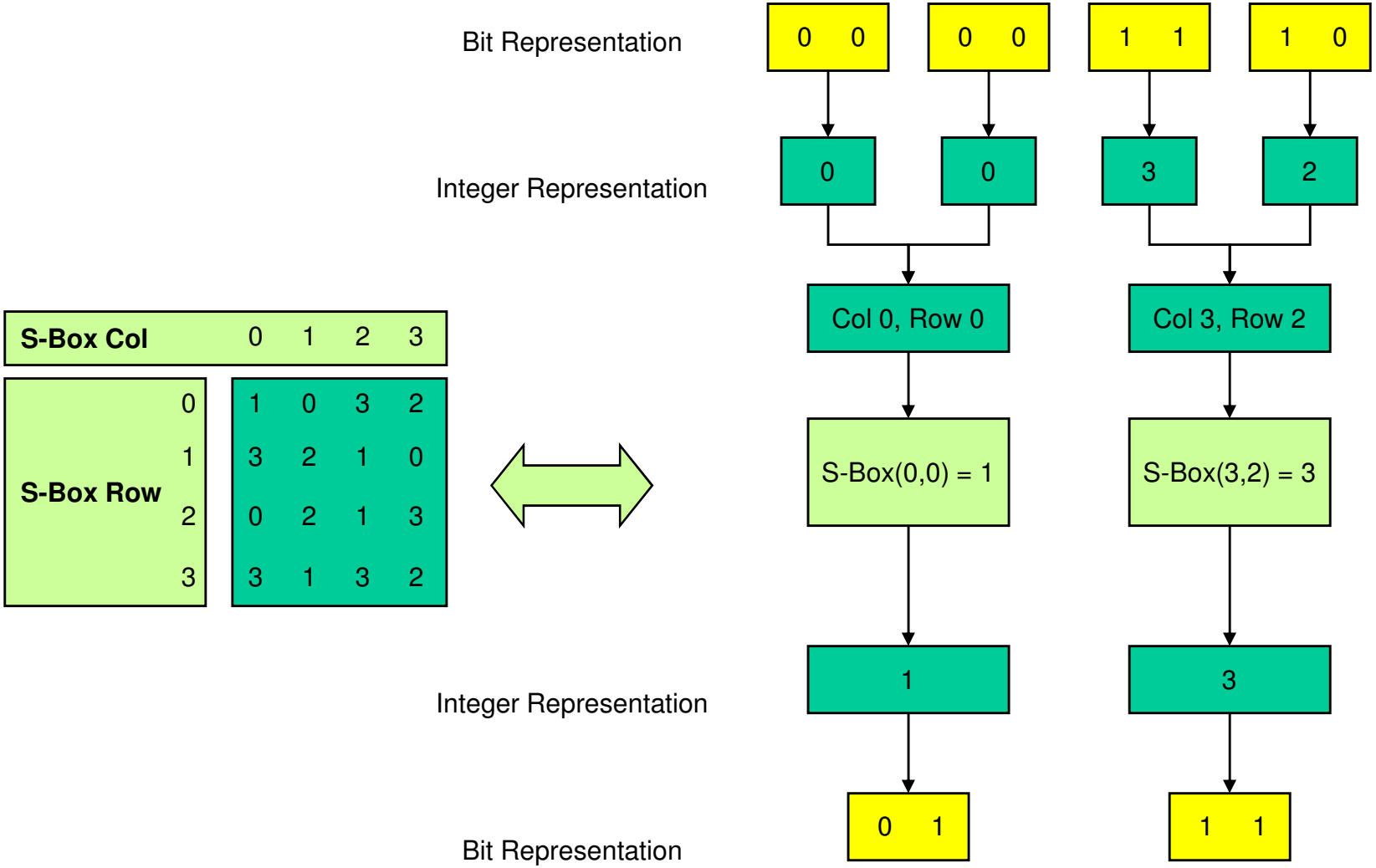
Note that many slides were derived from lecture slides by Lawrie Brown

Some Basic Mechanisms: Expansion/Permutation





Some Basic Mechanisms: S-Boxes



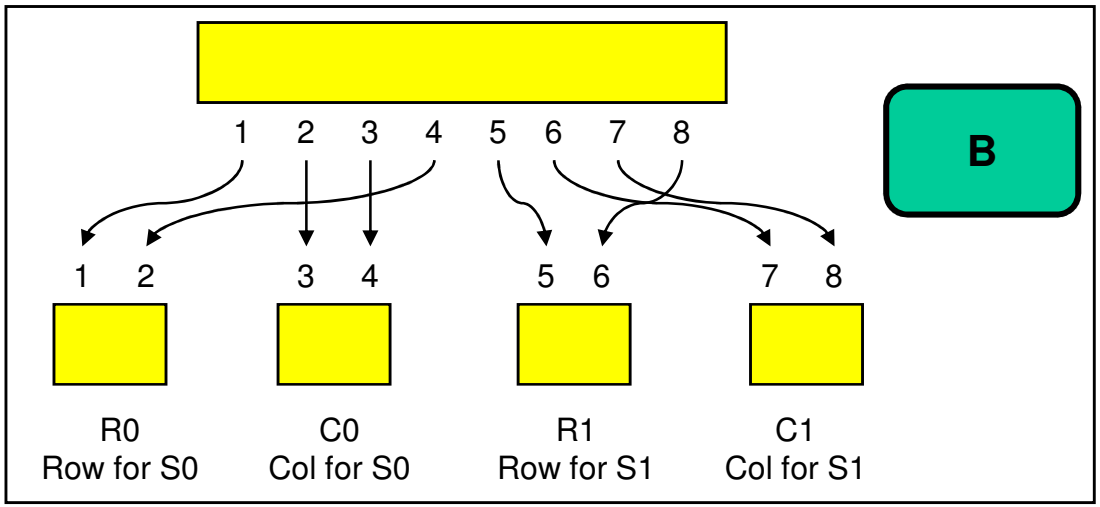
Example of Use

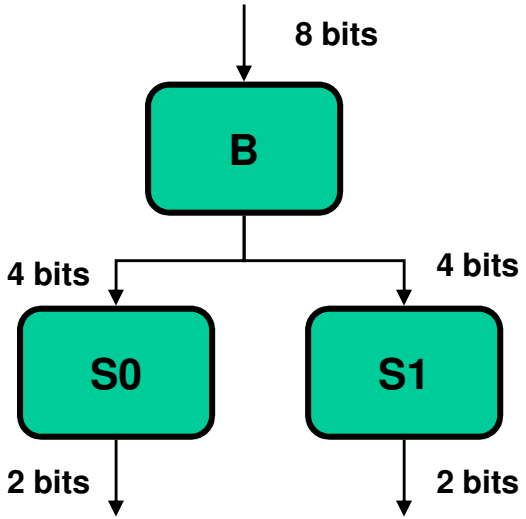
S0	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

S0

S1	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

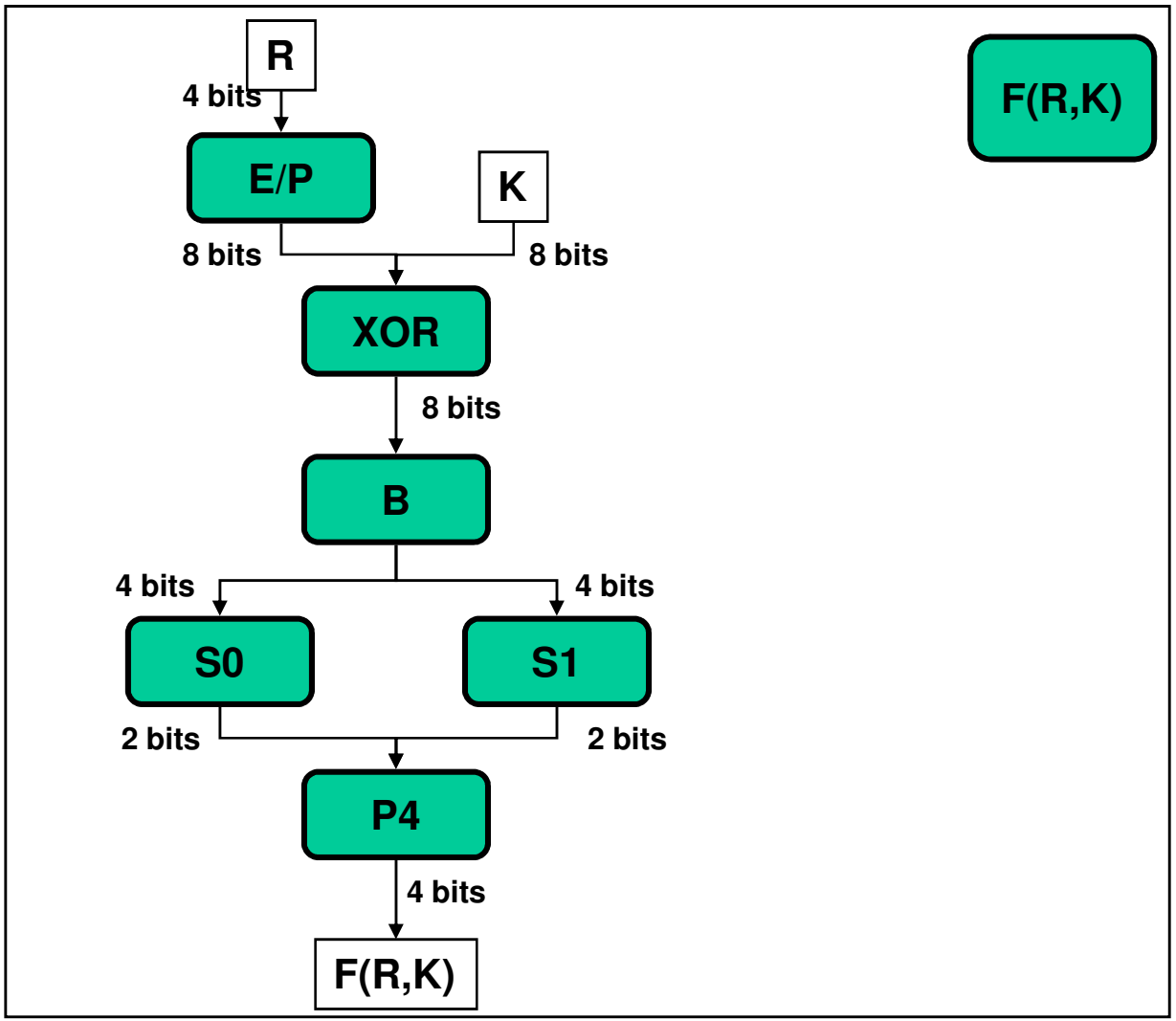
S1





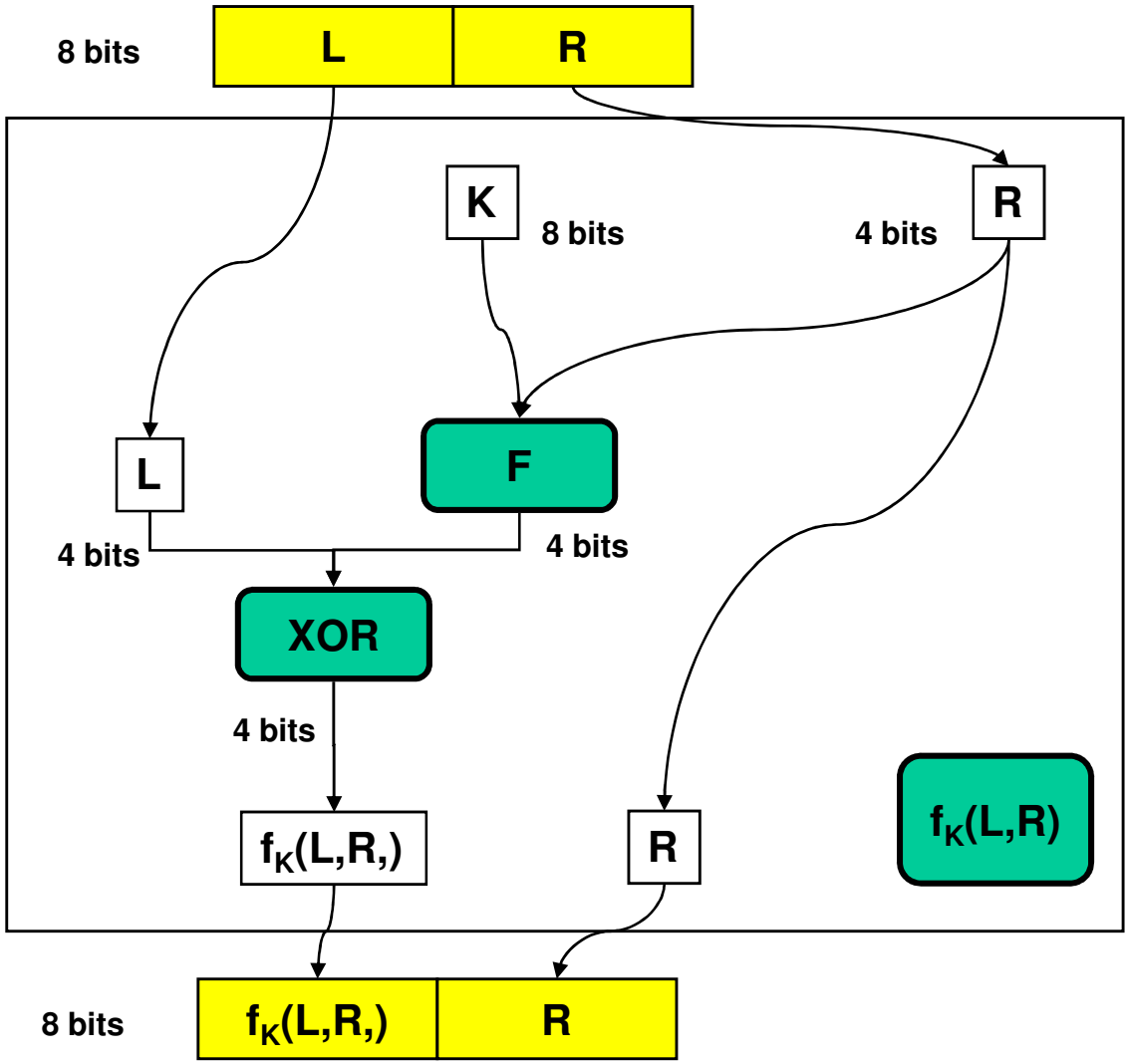
Example of Use : F(R,K)

R : 4 bit value
K : 8 bit value



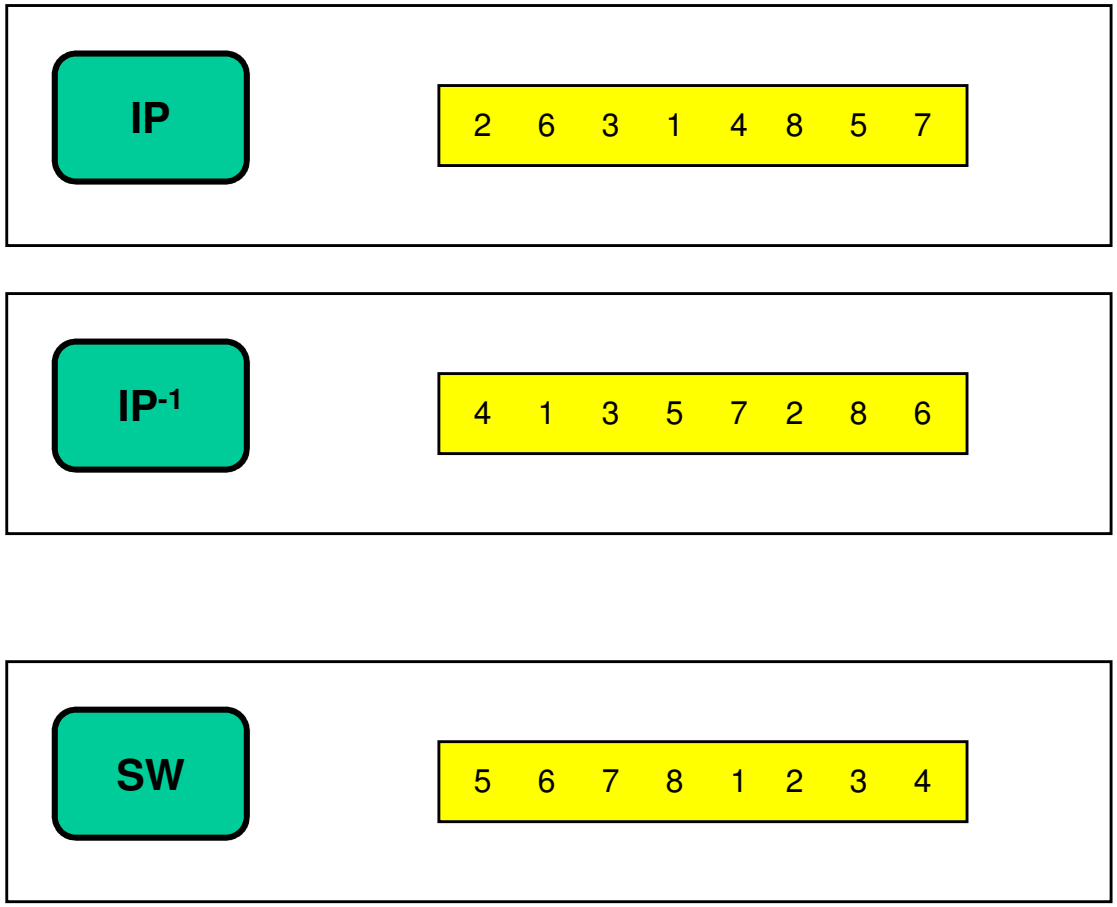
Note that many slides were derived from lecture slides by Lawrie Brown

Example of Use : $f_K(L,R)$



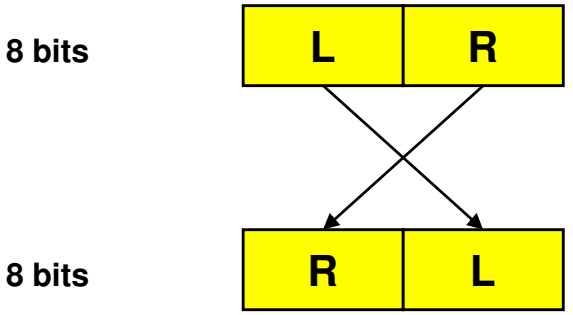
L : 4 bit value
R : 4 bit value

Other Elements



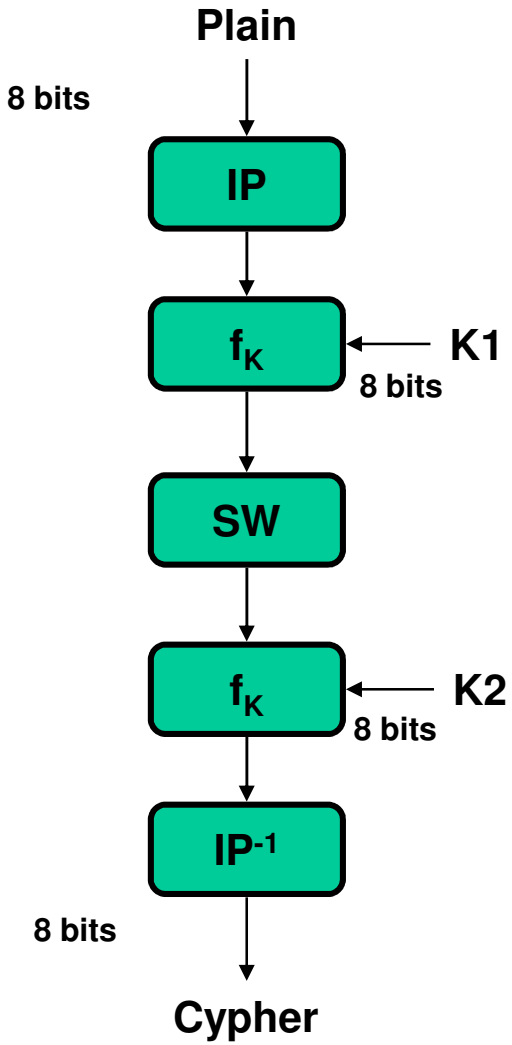
Initial Permutation

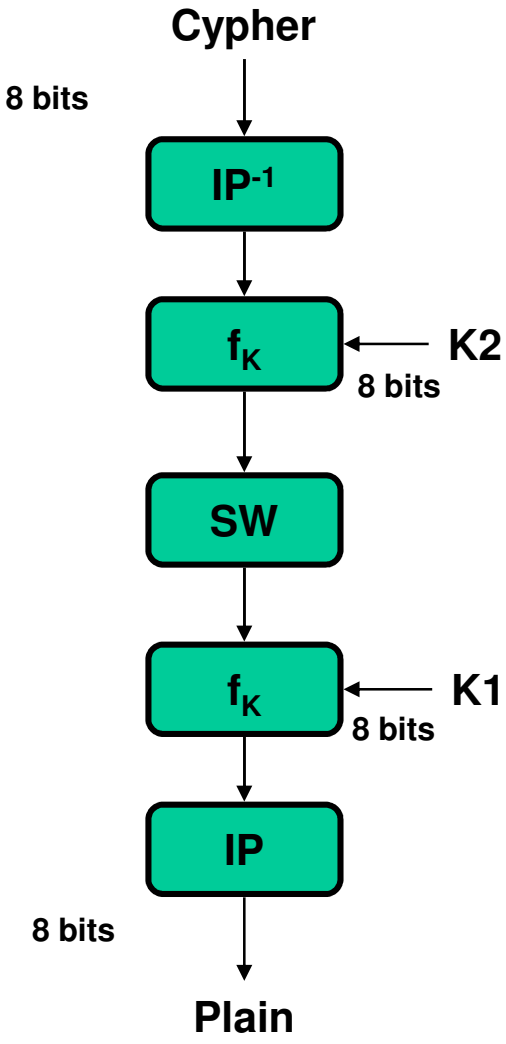
Switch « Word »



Note that many slides were derived from lecture slides by Lawrie Brown

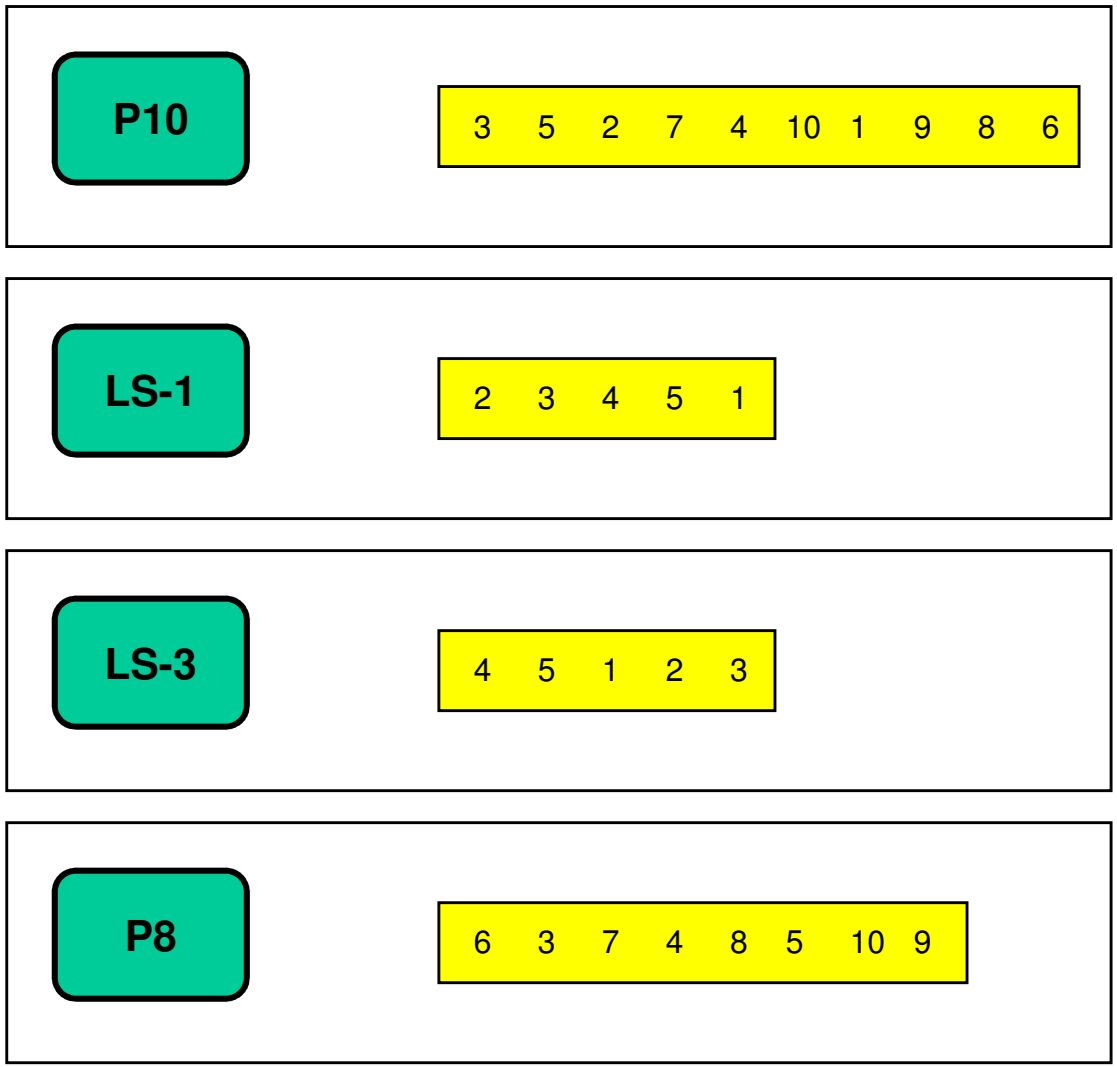
S-DES Encryption and Decryption





Note that many slides were derived from lecture slides by Lawrie Brown

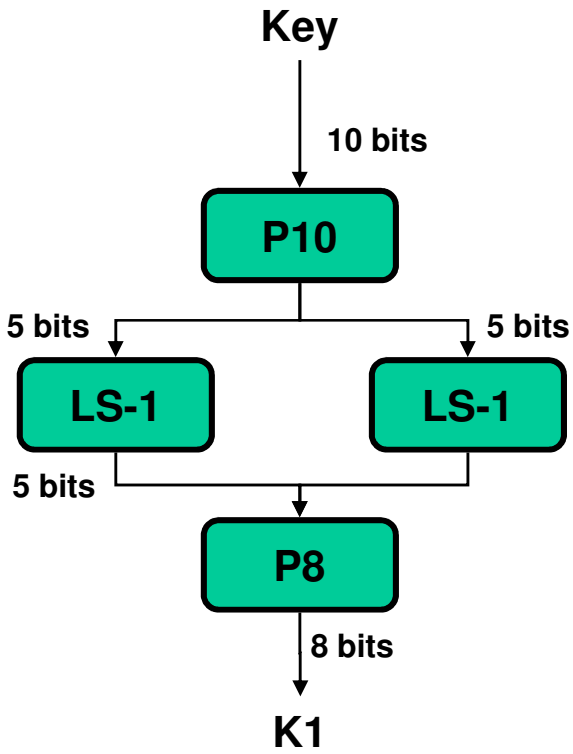
Other Elements

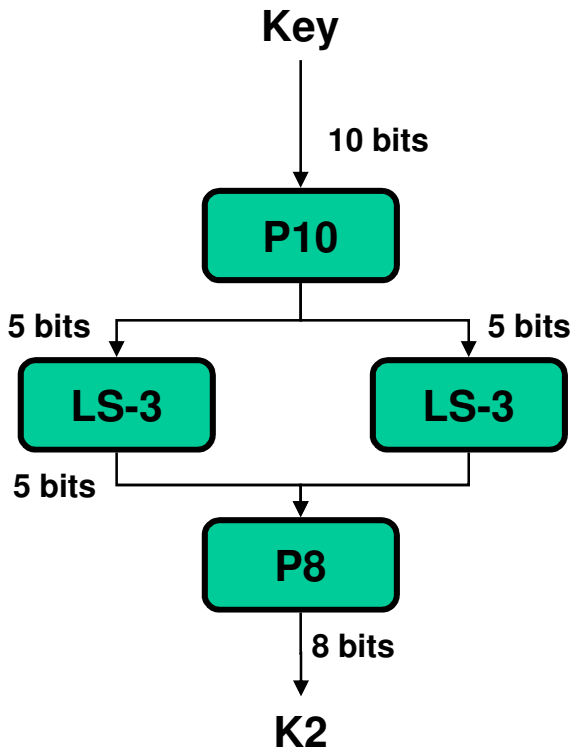


Left Shift One bit

Left Shift Three bits

K1 and K2 generation





DES and SDES calculator

- Initial version
  - Buzzard.ups.edu/sdes/sdes.htm (no longer available)
  - Is slightly different from the one used in the book
- Other version
  - www.codeproject.com/KB/recipes/Simple_Cryptographer.aspx

Block vs Stream Ciphers

- block ciphers process messages in into 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
- hence are focus of course

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

Claude Shannon and Substitution-Permutation Ciphers

- in 1949 Claude Shannon introduced idea of substitution-permutation (S-P) networks
  - modern substitution-transposition product cipher
- these form the basis of modern block ciphers
- S-P networks are based on the two primitive cryptographic operations we have seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
- provide *confusion* and *diffusion* of message

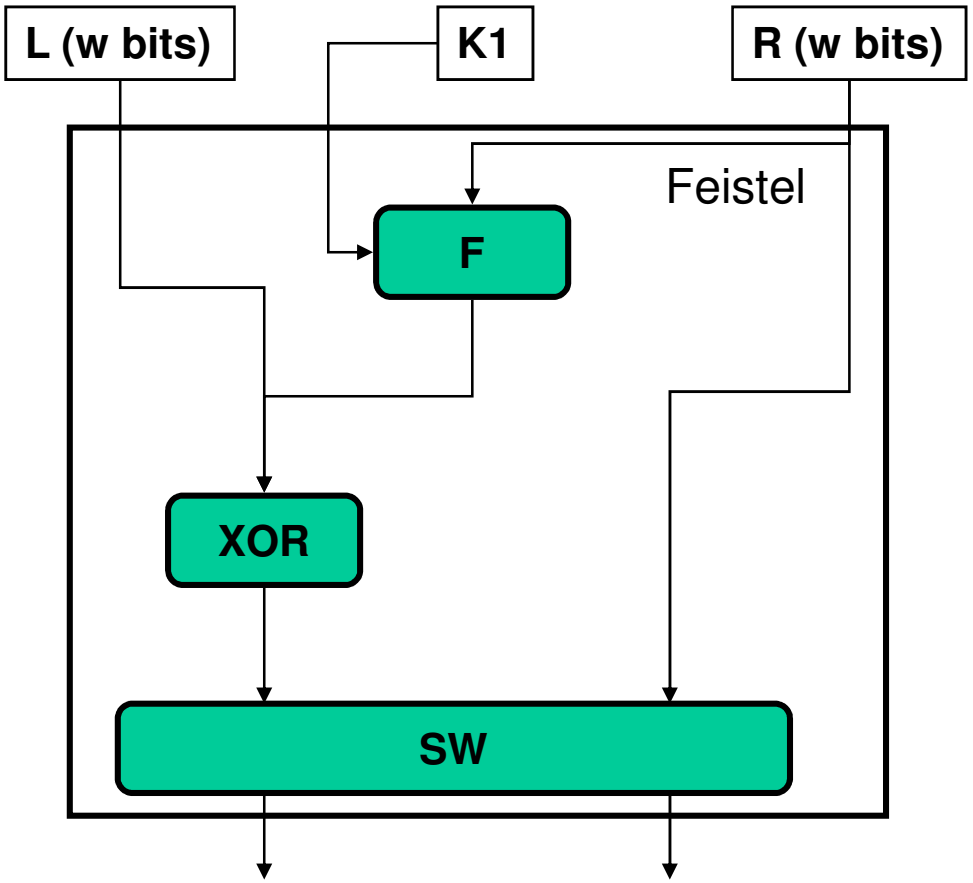
Confusion and Diffusion

- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this
- more practically Shannon suggested combining 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 substitution-permutation network concept

Feistel Cipher Structure



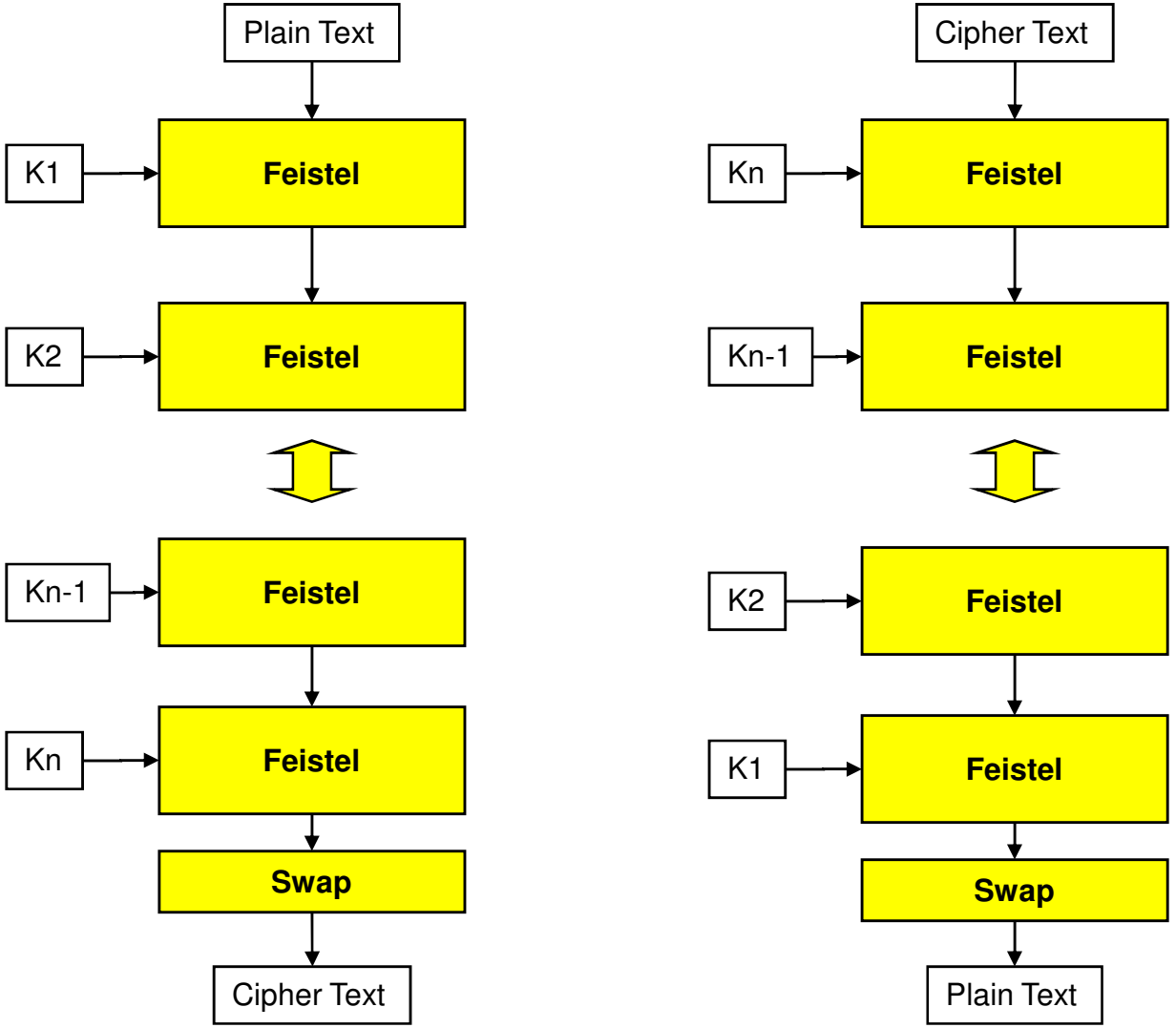
- **Encryption**
  - This block is repeated N times, from K1 to Kn
- **Decryption**
  - This block is repeated N times, from Kn to K1

Note that many slides were derived from lecture slides by Lawrie Brown

Feistel Cipher Design Principles

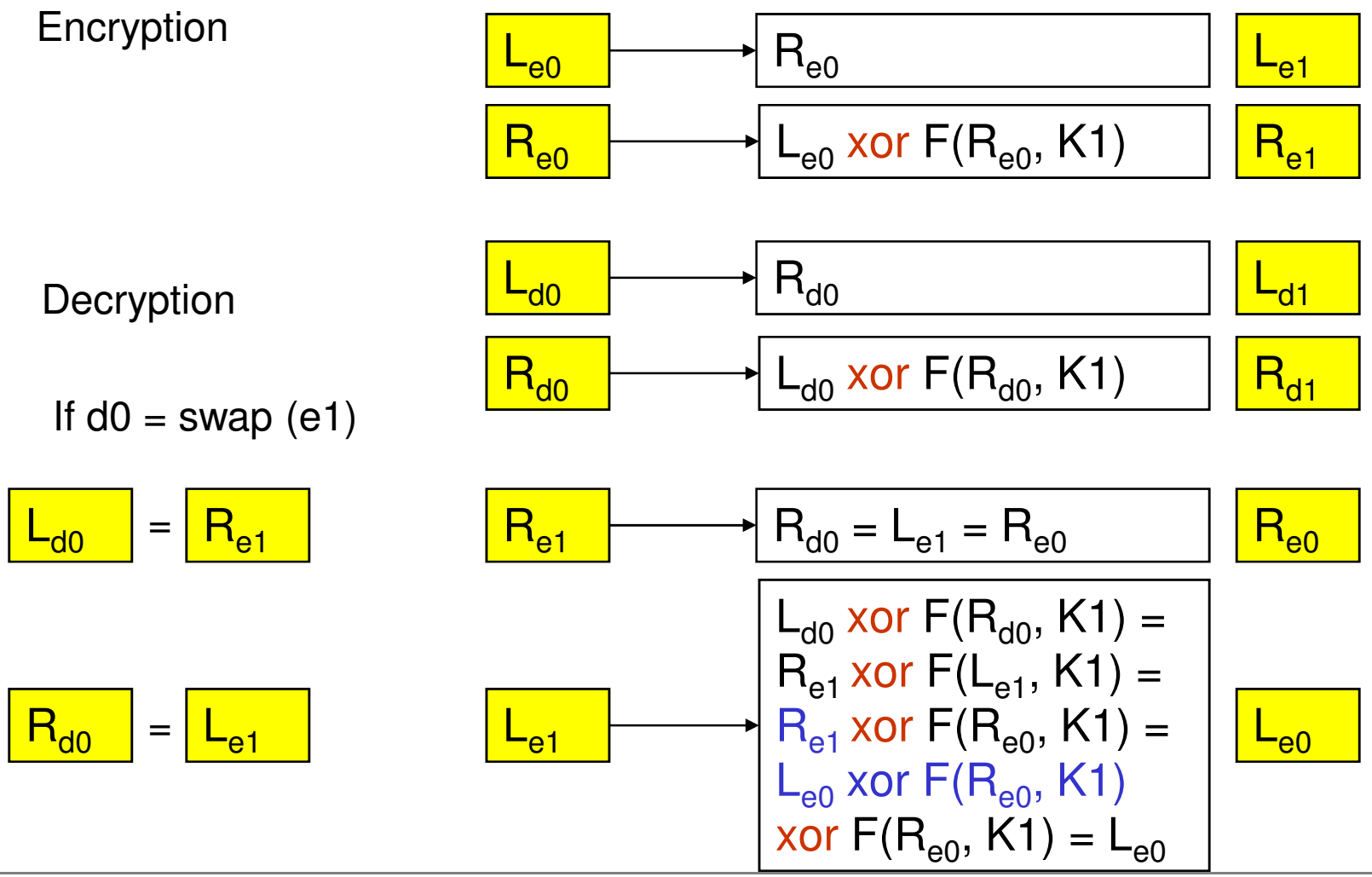
- **block size**
  - increasing size improves security, but slows cipher
  - DES: 64 bits, AES: 128 bits
- **key size**
  - increasing size improves security, makes exhaustive key searching harder, but may slow cipher
  - e.g. 128 bits
- **number of rounds**
  - increasing number improves security, but slows cipher
  - e.g. 16
- **subkey generation**
  - greater complexity can make analysis harder, but slows cipher
- **round function**
  - greater complexity can make analysis harder, but slows cipher
- **fast software en/decryption & ease of analysis**
  - are more recent concerns for practical use and testing

Feistel Cipher Decryption



Note that many slides were derived from lecture slides by Lawrie Brown

Proof that Description Works



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

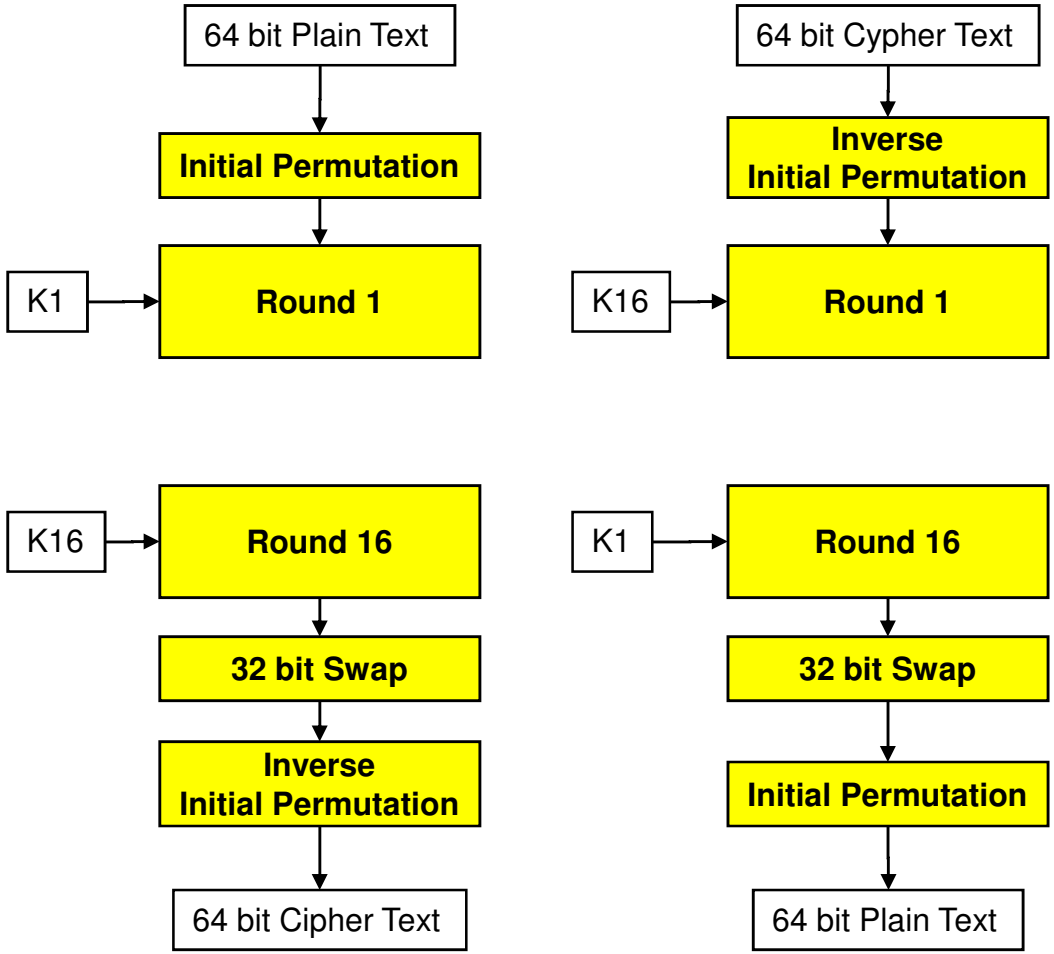
DES History

- IBM developed Lucifer cipher
  - by team led by Feistel
  - 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)
    - $7.2\ 10^{16}$
    - Brute force would take
      - 1000 years with a one encryption per microsecond
      - 10 hours with one million encryptions per microsecond
      - in 1998 a machine was build for less than 250000 dollars which took 3 days to crack a code
  - and because design criteria were classified
- subsequent events and public analysis show in fact design was appropriate
- DES has become widely used, esp in financial applications

DES Encryption and Decryption

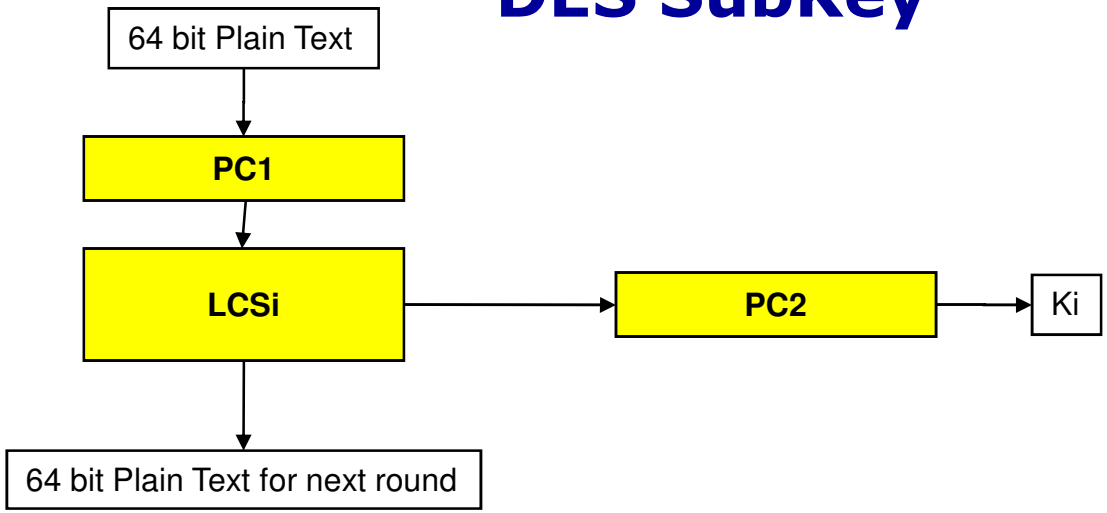


IP (58 50 42...

IP-1 (40 8 48 16...

Note that many slides were derived from lecture slides by Lawrie Brown

DES SubKey

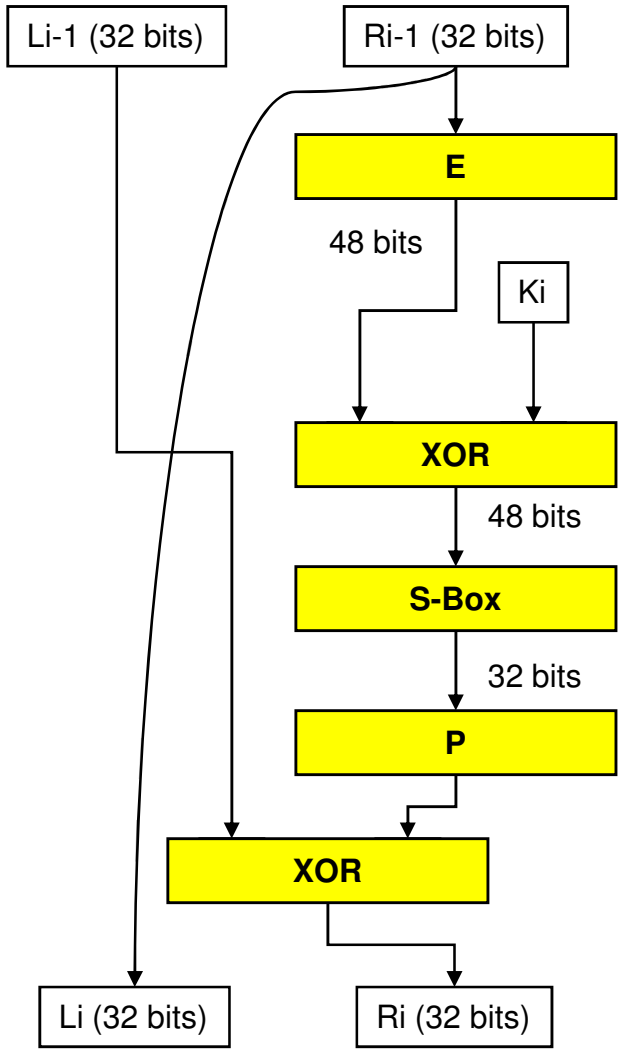


PC1 **Permuted Choice 1 (57 49 41 33...**

LCSi **Left Circular Shift (i+1 i+2 ...**

PC2 **Permuted Choice 2 (14 17 11 24...**

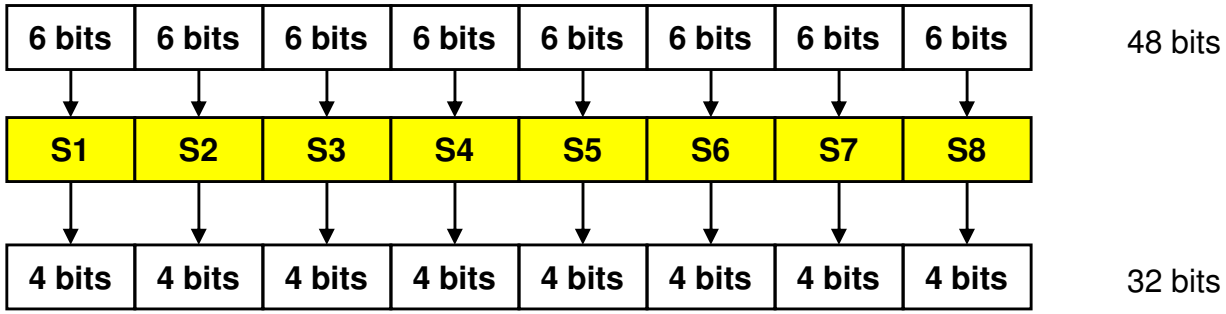
Detail of Round I

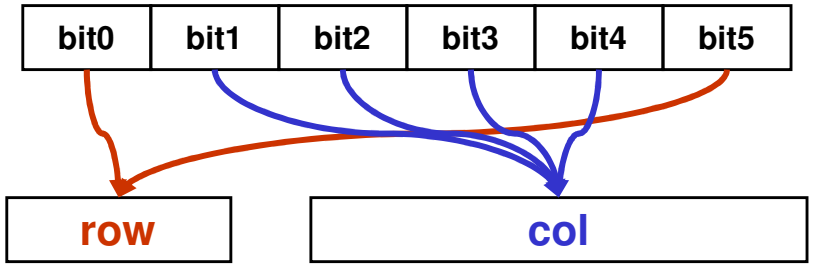


E (32 1 2 3 4 5 ….

P (16 7 20 21 29 ….

Use of S-Box





	0	1	2
0	14	4	13
1	0	15	7
2	4	1	14
3	15	12	8

Example for S1
Input : 0 0010 1
row : 01 or 1
col : 10 or 2
output : 0111 or 7

Avalanche Effect

- key desirable property of encryption alg
- where a change of **one** input or key bit results in changing approx **half** output bits
  - case 1 : same key, inputs with 1 bit difference
    - R0: 1 bit difference
    - R1: 6
    - R2: 21
    - R3: 35
    - R4: 39
  - case 2 : keys with one bit difference, same input
    - R0: 0 bit difference
    - R1: 2
    - R2: 14
    - R3: 28
    - R4: 32

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
- now considering alternatives to DES

Block Cipher Design Principles

- basic principles still like Feistel in 1970's
- criteria on S-boxes and P function
  - for all bit input, output bit average is 0.5
  - …
  - 3 types of s-box generation
    - use pseudorandom number generation
    - random with testing against criteria
    - human selection (difficult for large s-boxes)
- number of rounds
  - more is better, exhaustive search best attack
- function f:
  - must be nonlinear, provides "confusion", avalanche
- key schedule
  - complex subkey creation, key avalanche

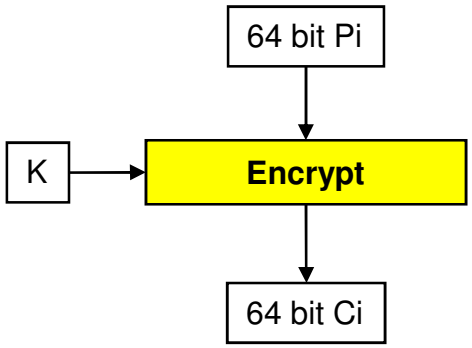
Modes of Operation

- block ciphers encrypt fixed size blocks
- eg. DES encrypts 64-bit blocks, with 56-bit key
- need way to use in practise, given usually have arbitrary amount of information to encrypt
- four were defined for DES in ANSI standard **ANSI X3.106-1983 Modes of Use**
- subsequently now have 5 for DES and AES
- 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

  $C_i = DES_K(P_i)$
- uses: secure transmission of single values



Decryption uses Decrypt

Advantages and Limitations of ECB

- repetitions in message 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 due to encrypted message blocks being independent
- main use is sending a few blocks of data

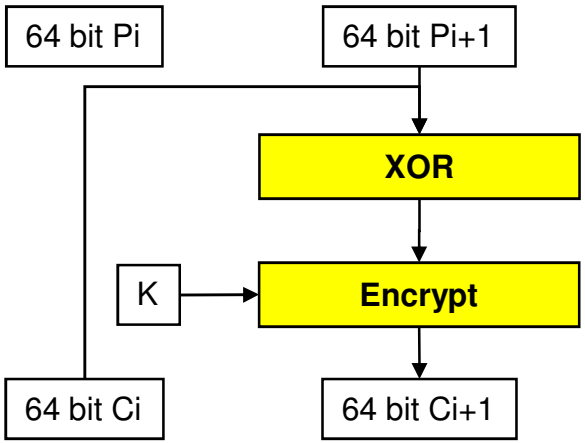
Cipher Block Chaining (CBC)

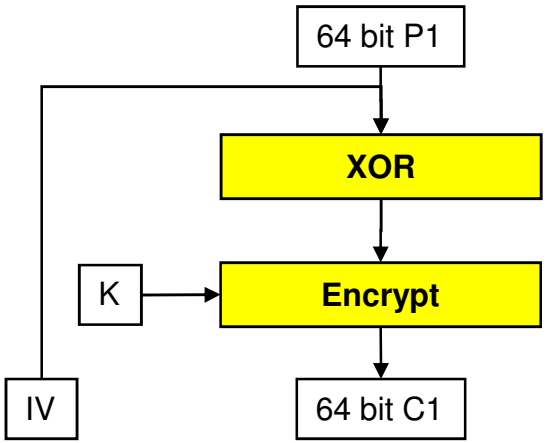
- message is broken into blocks
- but these are linked together in the encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

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

$C_{-1} = IV$

- uses: bulk data encryption, authentication





Decryption uses Decrypt

Advantages and Limitations of CBC

- each ciphertext block depends on **all** previous message blocks
- thus a change in the message affects all ciphertext blocks after the change as well as the original block
- need **Initial Value** (IV) known to sender & receiver
  - however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
  - hence either IV must be a fixed value (as in EFTPOS) or it must be sent encrypted in ECB mode before rest of message
- at end of message, handle possible last short block
  - by padding either with known non-data value (eg nulls)
  - or pad last block with count of pad size
    - eg. [b1 b2 b3 0 0 0 0 5] <- 3 data bytes, then 5 bytes pad+count

Cipher FeedBack (CFB)

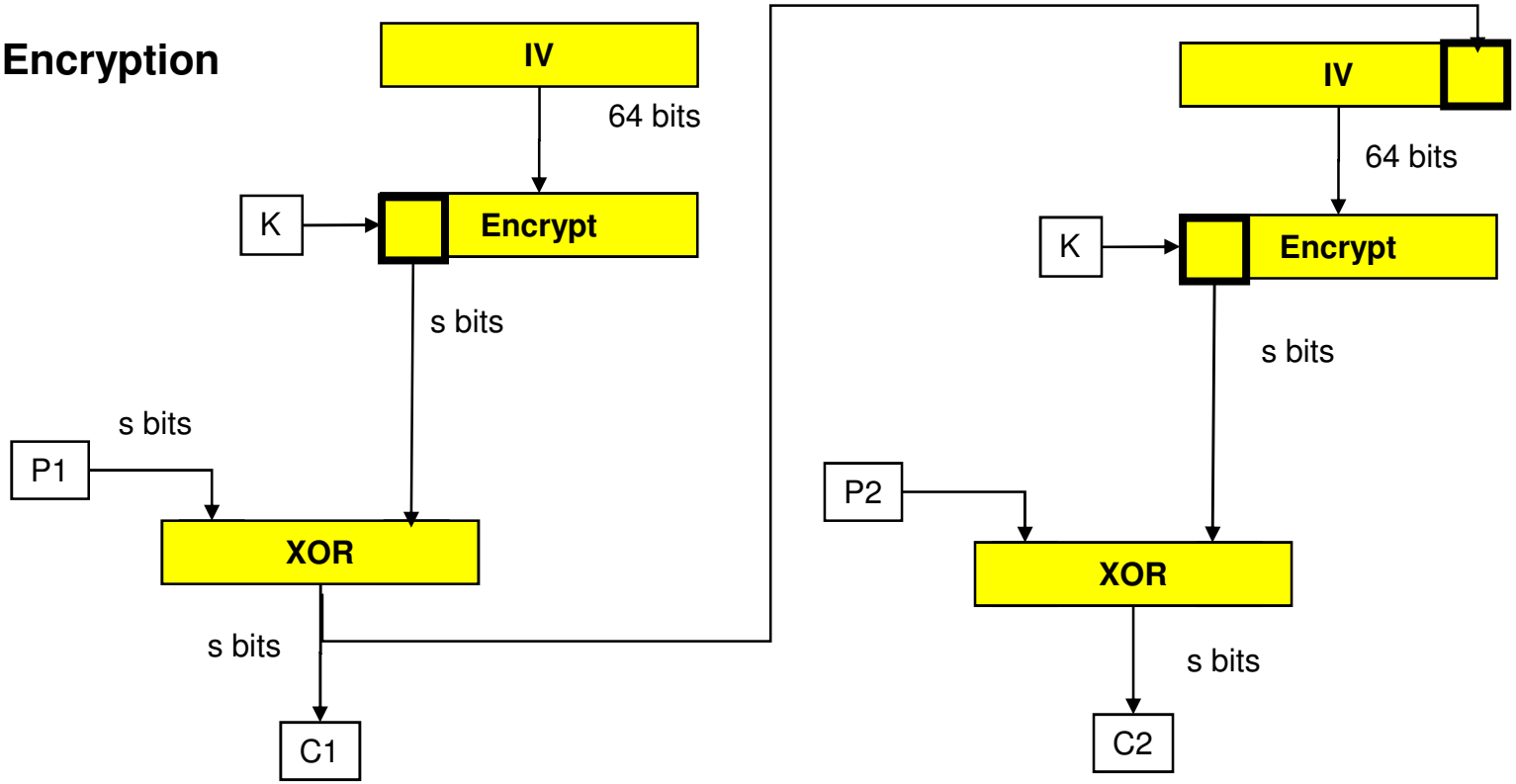
- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8 or 64 or whatever) to be feed back
  - denoted CFB-1, CFB-8, CFB-64 etc
- is most efficient to use all 64 bits (CFB-64)

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

$$C_{-1} = IV$$

- uses: stream data encryption, authentication

Cipher FeedBack (CFB)



Decryption uses same scheme!!

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 propagate for several blocks after the error

Output FeedBack (OFB)

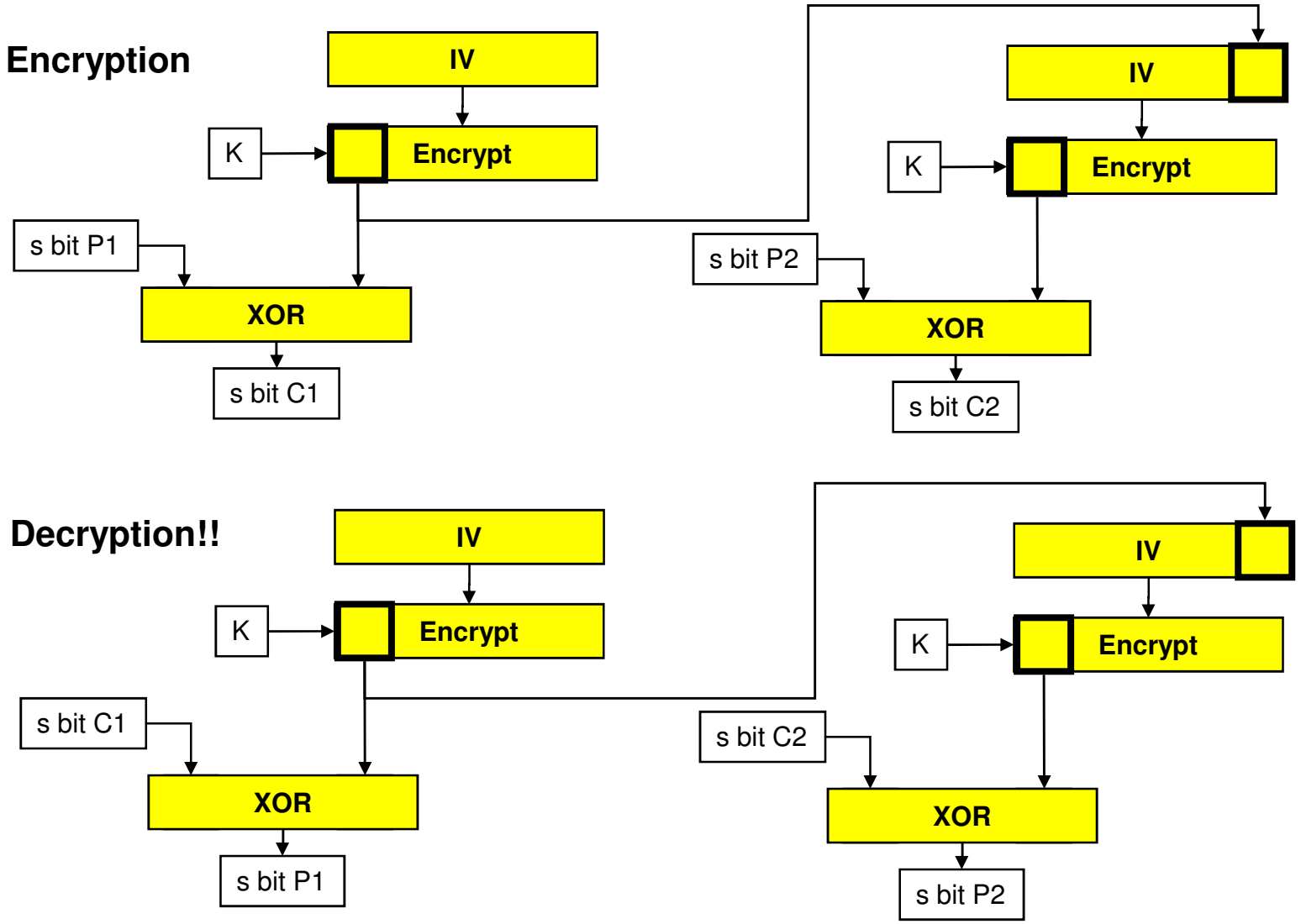
- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance

  $C_i = P_i \text{ XOR } O_i$

  $O_i = DES_K(O_{i-1})$

  $O_{-1} = IV$
- uses: stream encryption over noisy channels

Output FeedBack (OFB)



Advantages and Limitations of OFB

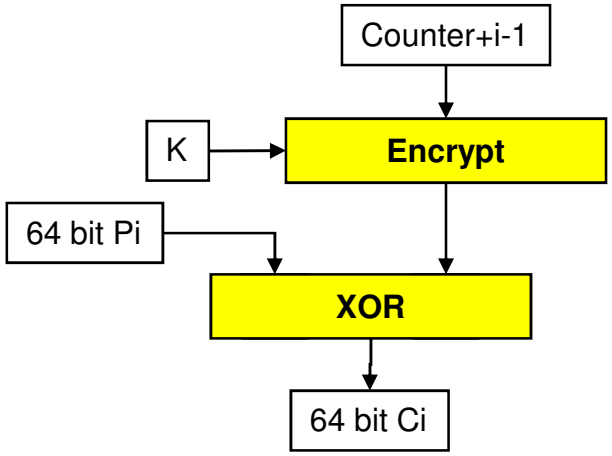
- used when error feedback a problem or where need to encryptions before message is available
- superficially similar to CFB
- but feedback is from the output of cipher and is independent of message
- a variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs
- originally specified with m-bit feedback in the standards
- subsequent research has shown that only **OFB-64** 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)

  $C_i = P_i \text{ XOR } O_i$

  $O_i = DES_K(i)$
- uses: high-speed network encryptions



Decryption uses Decrypt

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions
  - 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)

Chapter 4 – Finite Fields

Introduction

- finite fields are important in cryptography
  - AES, Elliptic Curve, IDEA, Public Key
- concern operations on "numbers"
  - where what constitutes a "number" and the type of operations (i.e. "addition" and "multiplication") 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**

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 are:
- an abelian group with addition operation
- 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 inverses and no zero divisors, it forms an **integral domain**

Field

- a set of numbers with two operations:
  - abelian group for addition
  - abelian group for multiplication (ignoring 0)
  - ring

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. 1 ≡ `12 ≡ 23 ≡` 100 ≡ 34 mod 11
- b is called the **residue** of a mod n
  - since with integers can always write: `a = qn + b`
- usually have `0 <= b <= n-1`

  `(-12 mod 7) ≡ (-5 mod 7) ≡ (2 mod 7) ≡ (9 mod 7)`

Modulo 7 Example

```
...
-21 -20 -19 -18 -17 -16 -15
-14 -13 -12 -11 -10  -9  -8
 -7  -6  -5  -4  -3  -2  -1
  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  32  33  34
...
```

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

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`

Modular Arithmetic

- can do modular arithmetic with any group of integers: $Z_n = \{0, 1, \ldots, n-1\}$
- form a commutative ring for addition
- with a multiplicative identity
- note some peculiarities
  - if $(a+b) \equiv (a+c) \bmod n$ then $b \equiv c \bmod n$
  - but $(ab) \equiv (ac) \bmod n$ then $b \equiv c \bmod n$ only if a is relatively prime to n

Modulo 8 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

Greatest Common Divisor (GCD)

- a common problem in number theory
- GCD (a,b) of a and b is the largest number that divides 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

Euclid's GCD Algorithm

- an efficient way to find the GCD(a,b)
- uses theorem that:
  - `GCD(a,b) = GCD(b, a mod b)`
- **Euclid's Algorithm** to compute GCD(a,b):
  - `A=a, B=b`
  - `while B>0`
    - `R = A mod B`
    - `A = B, B = R`
  - `return A`

Example GCD(1970,1066)

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

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)

Example GF(7)

x	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

Finding Inverses

- can extend Euclid's algorithm:

```
EXTENDED EUCLID(m, b)
1. (A1, A2, A3)=(1, 0, m);
   (B1, B2, B3)=(0, 1, b)
2. if B3 = 0
   return A3 (gcd(m, b)); no inverse
3. if B3 = 1
   return B3 (gcd(m, b)); B2 (b^-1 mod m)
4. Q = A3 div B3
5. (T1, T2, T3)=(A1 - Q B1, A2 - Q B2, A3 - Q B3)
6. (A1, A2, A3)=(B1, B2, B3)
7. (B1, B2, B3)=(T1, T2, T3)
8. goto 2
```

Inverse of 550 in GF(1759)

Q	A1	A2	A3	B1	B2	B3
–	1	0	1759	0	1	550
3	0	1	550	1	-3	109
5	1	-3	109	-5	16	5
21	-5	16	5	106	-339	4
1	106	-339	4	-111	355	1

Polynomial Arithmetic

- can compute using polynomials

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

- several alternatives available
  - ordinary polynomial arithmetic
  - polynomial arithmetic with coefficients mod p
  - polynomial arithmetic with coefficients 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 $g(x) = x^2 - x + 1$

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

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

$f(x) \times g(x) = x^5 + 3x^2 - 2x + 2$

Polynomial Arithmetic with Modulo Coefficients

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

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

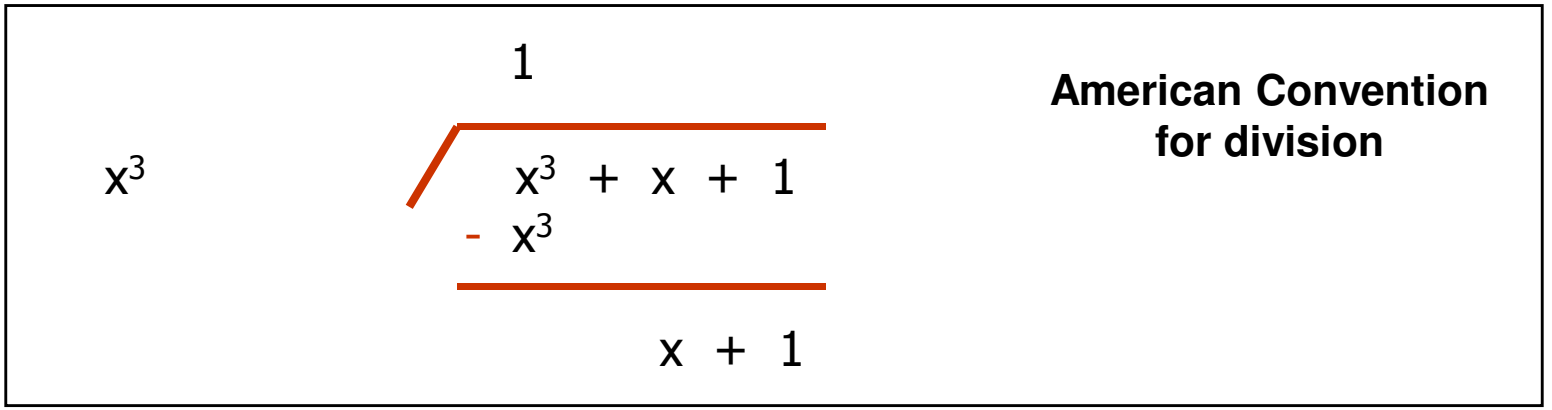
$$\begin{array}{rl} & x^3 + x^2 \\ + & + x^2 + x + 1 \\ \hline & x^3 + x + 1 \end{array}$$

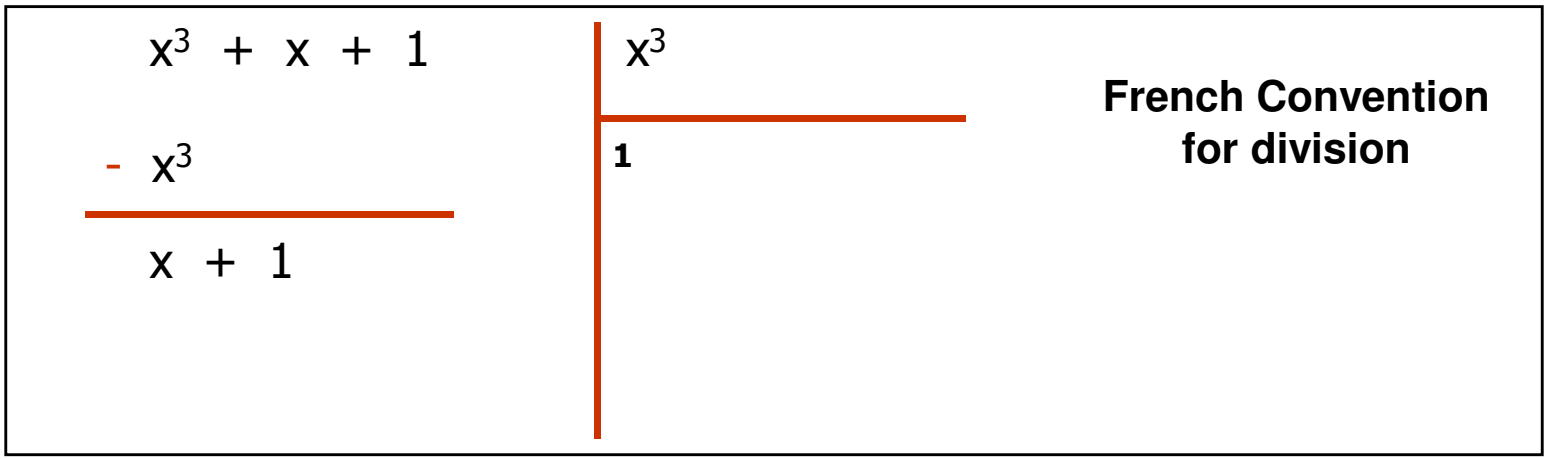
$f(x) \times g(x) = x^5 + x^2$

$$\begin{array}{rl} & x^3 + x^2 \\ \times & x^2 + x + 1 \\ \hline & x^3 + x^2 \\ & x^4 + x^3 \\ x^5 + & x^4 \\ \hline x^5 + & x^2 \end{array}$$

Polynomial Arithmetic with Modulo Coefficients

- $x^3 = x + 1 \mod x^3 + x + 1$





Note that many slides were derived from lecture slides by Lawrie Brown

Modular Polynomial Arithmetic

- 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) mod 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

GF(2^3): Polynomial Arithmetic Modulo x^3+x+1

		000	001	010	011	100	...
+		0	1	x	x+1	x^2	
000	0	0	1	x	x+1	x^2	
001	1	1	0	x+1	x	x^2+1	
010	x	x	x+1	0	1	x^2+x	
011	x+1	x+1	x	1	0	x^2+x+1	
...							

		000	001	010	011	100	...
x		0	1	x	x+1	x^2	
000	0	0	0	0	0	0	
001	1	0	1	x	x+1	x^2	
010	x	0	x	x^2	x^2+x	x+1	
011	x+1	0	x+1	x^2+x	x^2+1	x^2+x+1	

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)

Chapter 5 –Advanced Encryption Standard

Note that many slides were derived from lecture slides by Lawrie Brown

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 with 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 to practically cryptanalyse
  - cost – computational
  - algorithm & implementation characteristics
- final criteria
  - general security
  - software & hardware implementation ease
  - 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

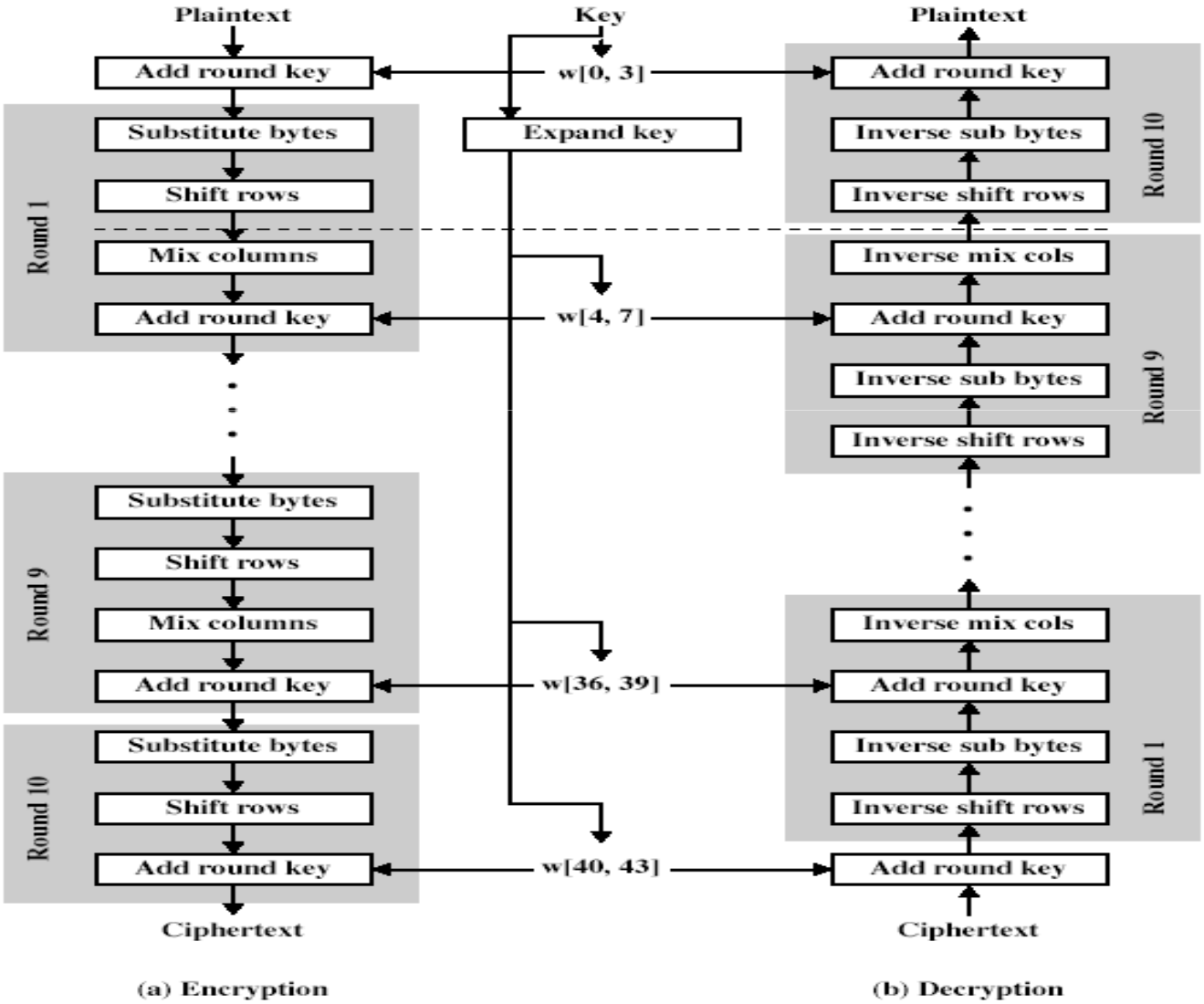
The AES Cipher - Rijndael

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

Rijndael

- processes data as 4 groups of 4 bytes (state)
- 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)
- initial XOR key material & incomplete last round
- all operations can be combined into XOR and table lookups - hence very fast & efficient

Rijndael


Plaintext
Add round key
Substitute bytes
Shift rows
Mix columns
Add round key
Round 1
Substitute bytes
Shift rows
Mix columns
Add round key
Round 9
Substitute bytes
Shift rows
Add round key
Round 10
Ciphertext
(a) Encryption
Key
w[0, 3]
Expand key
w[4, 7]
w[36, 39]
w[40, 43]
Plaintext
Add round key
Inverse sub bytes
Inverse shift rows
Round 10
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 1
Add round key
Ciphertext
(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 in row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by row 9 col 5 byte
  - which is the value {2A}
- **S-box is constructed using a defined transformation of the values in GF(2^8)**
  - **AES explains how to calculate the S-box**
  - **DES does not (heuristic?)**
- designed to be resistant to all known attacks

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 does shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Mix Columns

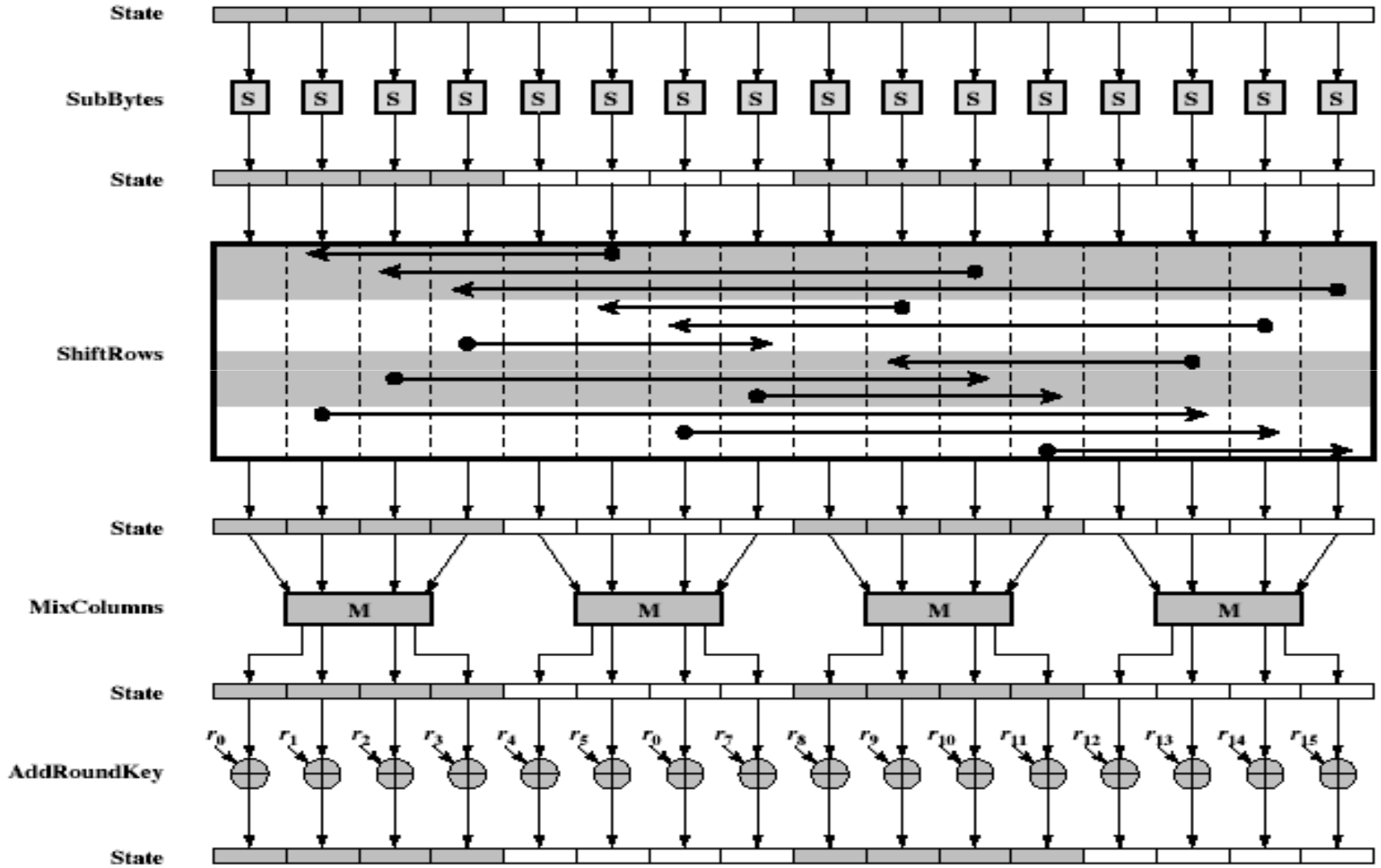
- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in **GF(2^8)** using prime poly **$m(x) = x^8+x^4+x^3+x+1$**

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

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 is identical since XOR is own inverse, just with correct round key
- designed to be as simple as possible

AES Round



Note that many slides were derived from lecture slides by Lawrie Brown

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
  - every 4th has S-box + rotate + XOR constant of previous before XOR together
- designed to resist known attacks

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

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 shifting
  - add round key works on byte XORs
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use a table lookup

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 16Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Summary

- have considered:
  - the AES selection process
  - the details of Rijndael – the AES cipher
  - looked at the steps in each round
  - the key expansion
  - implementation aspects

Chapter 6 – Contemporary Symmetric Ciphers

Triple 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

Why Triple-DES?

- why not Double-DES?
  - NOT same as some other single-DES use, but have
- meet-in-the-middle attack
  - works whenever use a cipher twice
    - $C = E_{K2}[E_{K1}[P]]$
    - $X = E_{K1}[P] = D_{K2}[C]$
  - 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

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

Blowfish

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
  - fast implementation on 32-bit CPUs
  - compact in use of memory
  - simple structure eases analysis/implemention
  - variable security by varying key size
- has been implemented in various products

Blowfish Key Schedule

- uses a **32 to 448** bit key
- used to generate
  - **18** 32-bit subkeys stored in K-array K_j
  - four **8x32** S-boxes stored in $S_{i,j}$
- key schedule consists of:
  - initialize **P-array** and then **4 S-boxes** using π
  - **XOR P-array** with key bits (reuse as needed)
  - loop repeatedly encrypting data using current P & S and replace successive pairs of P then S values
    - P1,P2 = $E_{P,S}[0]$
    - P3,P4 = $E_{P,S}[P1,P2]$
    - …
    - $S_{4,254},S_{4,255} = E_{PS}[S_{4,252},S_{4,253}]$
  - requires 521 encryptions, hence slow in rekeying

Blowfish Encryption

- uses two primitives: addition & XOR
- data is divided into two 32-bit halves L_0 & R_0

```
for i = 1 to 16 do
    R_i = L_{i-1} XOR P_i;
    L_i = F[R_i] XOR R_{i-1};
L_17 = R_16 XOR P_18;
R_17 = L_16 XOR i_17;
```

- where

$$F[a,b,c,d] = ((S_{1,a} + S_{2,b}) \text{ XOR } S_{3,c}) + S_{4,a}$$

Discussion

- **key dependent S-boxes and subkeys**, generated using cipher itself, makes analysis very difficult
- changing both halves in each round increases security
- provided key is large enough, brute-force key search is not practical, especially given the high key schedule cost

RC5

- a proprietary cipher owned by RSADSI
- designed by Ronald Rivest (of RSA fame)
- used in various RSADSI products
- can vary key size / data size / no rounds
- very clean and simple design
- easy implementation on various CPUs
- yet still regarded as secure

RC5 Ciphers

- RC5 is a family of ciphers RC5-w/r/b
  - w = word size in bits (16/32/64) nb data=2w
  - r = number of rounds (0..255)
  - b = number of bytes in key (0..255)
- nominal version is RC5-32/12/16
  - ie 32-bit words so encrypts 64-bit data blocks
  - using 12 rounds
  - with 16 bytes (128-bit) secret key

RC5 Key Expansion

- RC5 uses 2r+2 subkey words (w-bits)
- subkeys are stored in array `S[i]`, i=0..t-1
- then the key schedule consists of
  - initializing **S** to a fixed pseudorandom value, based on constants e and phi **(golden ratio)**
  - the byte key is copied (little-endian) into a c-word array L
  - a mixing operation then combines L and S to form the final S array

RC5 Encryption

- split input into two halves A & B

  $L_0 = A + S[0];$

  $R_0 = B + S[1];$

  for i = 1 to r do

  $L_i = ((L_{i-1} \text{ XOR } R_{i-1}) \lll R_{i-1}) + S[2 \times i];$

  $R_i = ((R_{i-1} \text{ XOR } L_i) \lll L_i) + S[2 \times i + 1];$

- each round is like 2 DES rounds
- Data dependent rotation (<<<)
  - main source of non-linearity
- need reasonable number of rounds (eg 12-16)

RC5 Modes

- RFC2040 defines 4 modes used by RC5
  - RC5 Block Cipher, is ECB mode
  - RC5-CBC, is CBC mode
  - RC5-CBC-PAD, is CBC with padding by bytes with value being the number of padding bytes
  - RC5-CTS, a variant of CBC which is the same size as the original message, uses ciphertext stealing to keep size same as original

Block Cipher Characteristics

- features seen in modern block ciphers are:
  - variable key length / block size / no rounds
  - mixed operators, data/key dependent rotation
  - key dependent S-boxes
  - more complex key scheduling
  - operation of full data in each round
  - varying non-linear functions

Stream Ciphers

- process the message bit by bit (as a stream)
- typically have a (pseudo) random **stream key**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message
  - $C_i = M_i$ XOR $StreamKey_i$
- what could be simpler!!!!
- but must never reuse stream key
  - otherwise can remove effect and recover messages

Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
  - correlation immunity
  - confusion
  - diffusion
  - use of highly non-linear boolean functions

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
- Initial value of S
  - given a key k of length l bytes

```
for i = 0 to 255 do
    S[i] = i
j = 0
for i = 0 to 255 do
    j = (j + S[i] + k[i mod l]) (mod 256)
    swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- XOR 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 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

Summary

- have considered:
  - some other modern symmetric block ciphers
  - Triple-DES
  - Blowfish
  - RC5
  - briefly introduced stream ciphers
  - RC4

Chapter 7 – Confidentiality Using Symmetric Encryption

Confidentiality using Symmetric Encryption

- traditionally symmetric encryption is used to provide message confidentiality
- consider typical scenario
  - workstations on LANs access other workstations & servers on LAN
  - LANs interconnected using switches/routers
  - with external lines or radio/satellite links
- consider attacks and placement in this scenario
  - snooping from another workstation
  - use dial-in to LAN or server to snoop
  - use external router link to enter & snoop
  - monitor and/or modify traffic one external links

Confidentiality using Symmetric 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

Traffic Analysis

- 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

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

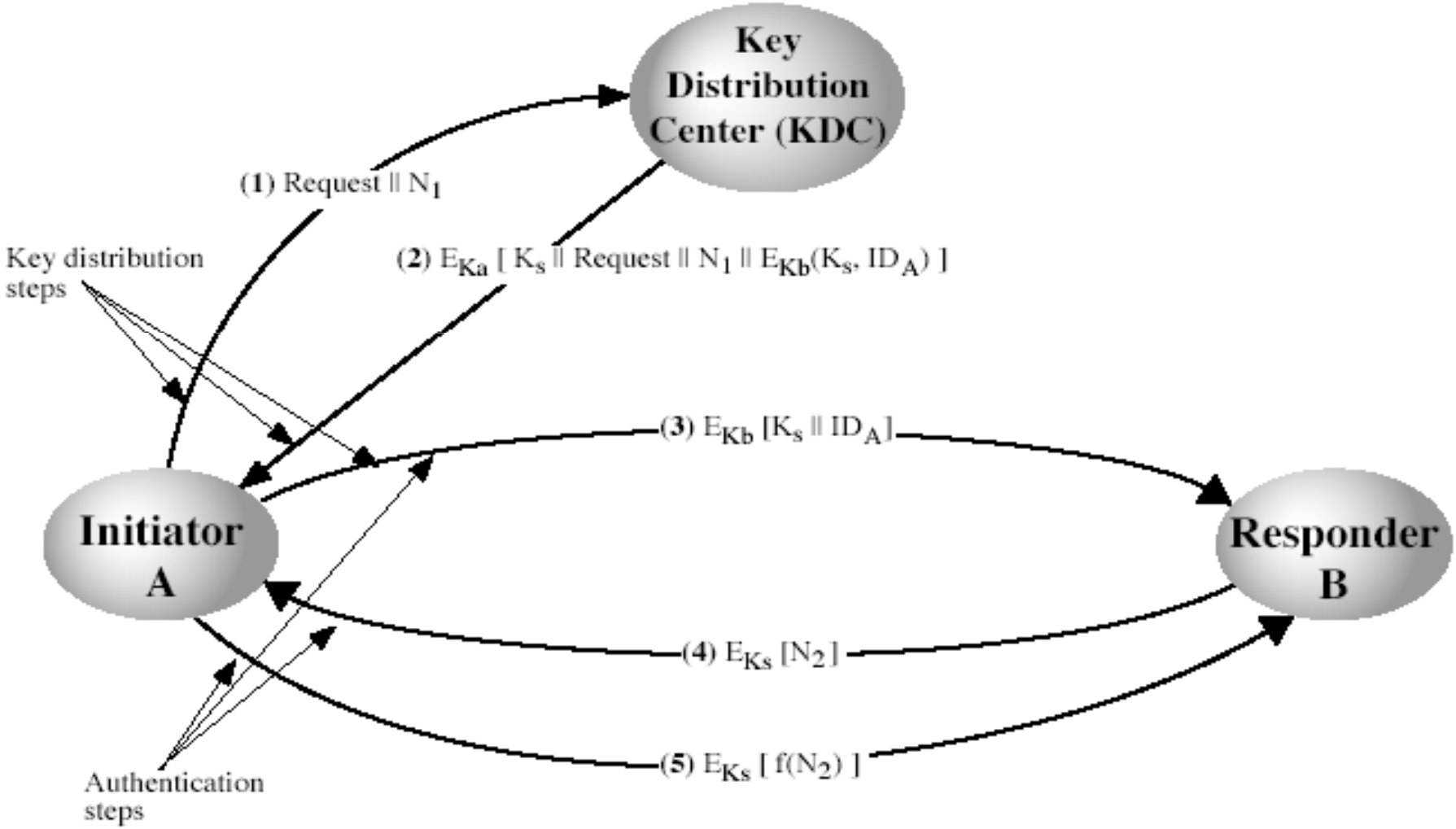
Key Distribution

- 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 Distribution Scenario



Note that many slides were derived from lecture slides by Lawrie Brown

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 purposes keys are used for

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
- in all cases its critical that these values be
  - statistically random
    - with uniform distribution, independent
  - unpredictable cannot infer future sequence on previous values

Natural Random Noise

- best source is natural randomness in real world
- find a regular but random event and monitor
- do generally need special h/w to do this
  - eg. radiation counters, radio noise, audio noise, thermal noise in diodes, leaky capacitors, mercury discharge tubes etc
- starting to see such h/w in new CPU's
- problems of **bias** or uneven distribution in signal
  - have to compensate for this when sample and use
  - best to only use a few noisiest bits from each sample

Published Sources

- a few published collections of random numbers
- Rand Co, in 1955, published 1 million numbers
  - generated using an electronic roulette wheel
  - has been used in some cipher designs cf Khafre
- earlier Tippett in 1927 published a collection
- issues are that:
  - these are limited
  - too well-known for most uses

Pseudorandom Number Generators (PRNGs)

- algorithmic technique to create "random numbers"
  - although not truly random
  - can pass many tests of "randomness"

Linear Congruential Generator

- common iterative technique using:
  - Xn+1 = (aXn + c) mod m
- given suitable values of parameters can produce a long random-like sequence
- suitable criteria to have are:
  - function generates a full-period
  - generated sequence should appear random
  - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values

Using Block Ciphers as Stream Ciphers

- can use block cipher to generate numbers
- use Counter Mode
  - Xi = EKm[i]
- use Output Feedback Mode
  - Xi = EKm[Xi-1]
- ANSI X9.17 PRNG
  - uses date-time + seed inputs and 3 triple-DES encryptions to generate new seed & random

Blum Blum Shub Generator

- based on public key algorithms
- use least significant bit from iterative equation:
  - xi+1 = xi2 mod n
  - where n=p.q, and primes p,q=3 mod 4
- unpredictable, passes next-bit test
- security rests on difficulty of factoring N
- is unpredictable given any run of bits
- slow, since very large numbers must be used
- too slow for cipher use, good for key generation

Summary

- have considered:
  - use of symmetric encryption to protect confidentiality
  - need for good key distribution
  - use of trusted third party KDC's
  - random number generation

Chapter 8 – Introduction to Number Theory

Note that many slides were derived from lecture slides by Lawrie Brown

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 × b × 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. **$91 = 7 \times 13$; $3600 = 2^4 \times 3^2 \times 5^2$**

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

Relatively Prime Numbers & GCD

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

Fermat's Theorem

- $a^{p-1} \bmod p = 1$
  - where `p` is prime
  - a not divisible by p (`gcd(a,p)=1`)

- also known as Fermat's Little Theorem
- useful in public key and primality testing

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 ø(n)

- to compute **ø(n)** need to count number of elements to be excluded
- in general need prime factorization, but
  - for p (p prime) ø(p) = p−1
  - for p.q (p,q prime) ø(p.q) = (p−1)(q−1)
- eg.
  - ø(37) = 36
  - ø(21) = (3−1)×(7−1) = 2×6 = 12

Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{\phi(n)} \bmod N = 1$
  - where gcd(a,N)=1
- eg.
  - $a=3$
  - $n=10$
  - $\phi(10)=4$;
  - hence $3^4 = 81 = 1 \bmod 10$
  - $a=2$
  - $n=11$
  - $\phi(11)=10$;
  - 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

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 **< ¼**
- 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
- since can immediately ignore evens and multiples of 5, in practice only need test **0.4 ln(n)** numbers of size n before locate a prime
  - note this is only the "average" sometimes primes are close together, at other times are quite far apart

Chinese Remainder Theorem

- used to speed up modulo computations
- working modulo a product of numbers
  - eg. **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 (**A mod M**) can firstly compute all (**a_i mod m_i**) separately and then combine results to get answer using:

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

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

Primitive Roots

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

Discrete Logarithms or Indices

- Discrete exponentiation modulo p
  - Calculate $\mathbf{a^m \bmod p}$
- Discrete logarithm (inverse problem)
  - find **x** where $\mathbf{a^x = b \bmod p}$
- written as $\mathbf{x = \log_a b \pmod p}$
- if a is a primitive root then always exists, otherwise may not
  - $\mathbf{x = \log_3 4 \pmod{13}}$ (x st 3x = 4 mod 13) has no answer
  - $\mathbf{x = \log_2 3 \pmod{13} = 4}$ by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a hard problem

Summary

- have considered:
  - prime numbers
  - Fermat's and Euler's Theorems
  - Primality Testing
  - Chinese Remainder Theorem
  - Discrete Logarithms

Chapter 9 – Public Key Cryptography and RSA

Note that many slides were derived from lecture slides by Lawrie Brown

Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

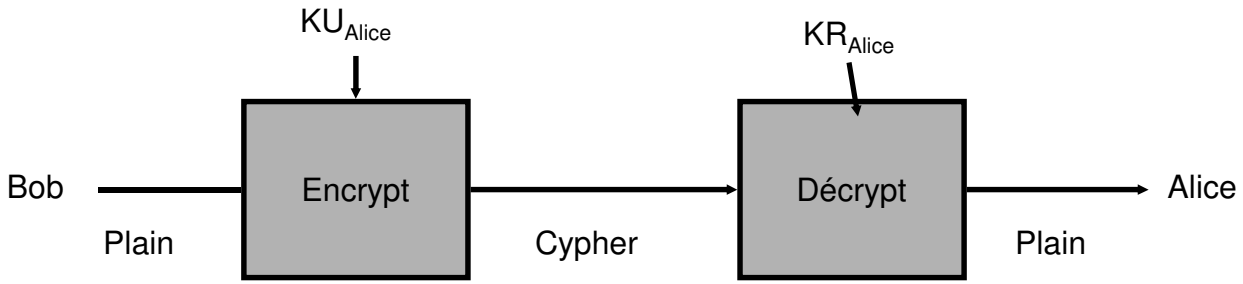
- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

Public-Key Cryptography

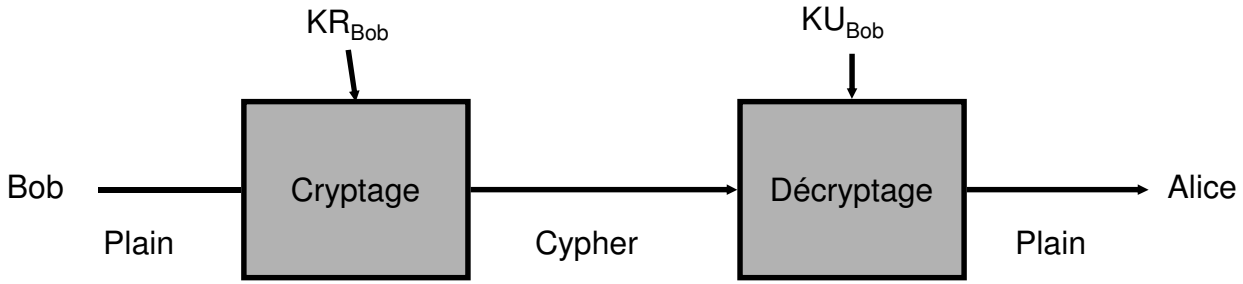
- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Techniques : (Public Key Infrastructure)

- Alice owns a public key KU_{Alice} and a private key KR_{Alice}
- Bob owns a public key KU_{Bob} and a private key KR_{Bob}
- Information than only Alice can read (confidentiality)

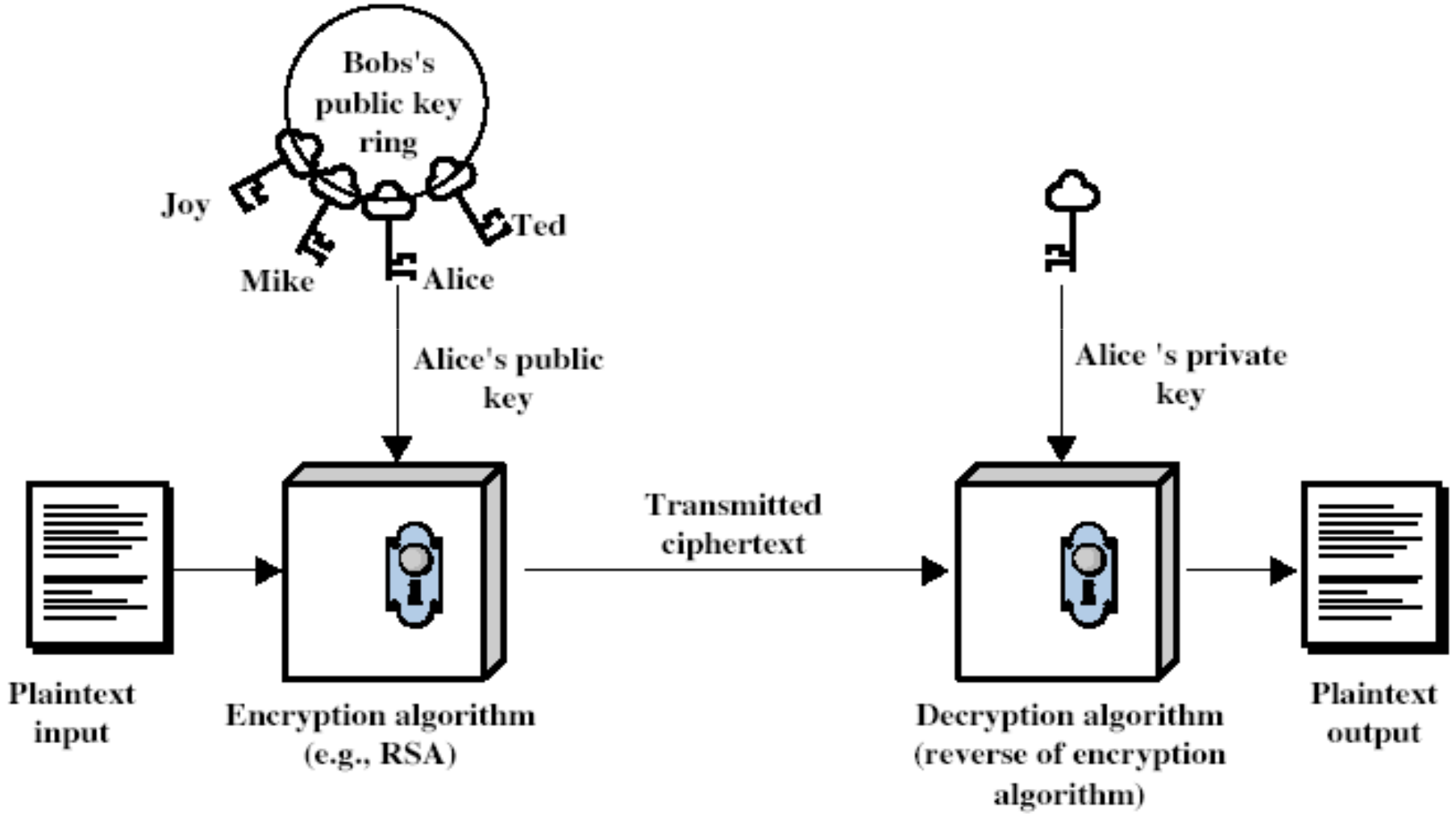


- Information that only Bob can send (authentication)



Note that many slides were derived from lecture slides by Lawrie Brown

Public-Key Cryptography



Note that many slides were derived from lecture slides by Lawrie Brown

Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

Public-Key Characteristics

- Public-Key algorithms rely on two keys with the characteristics that it is:
  - computationally infeasible to find decryption key knowing only algorithm & encryption key
  - computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (in some schemes)

Public-Key Cryptosystems

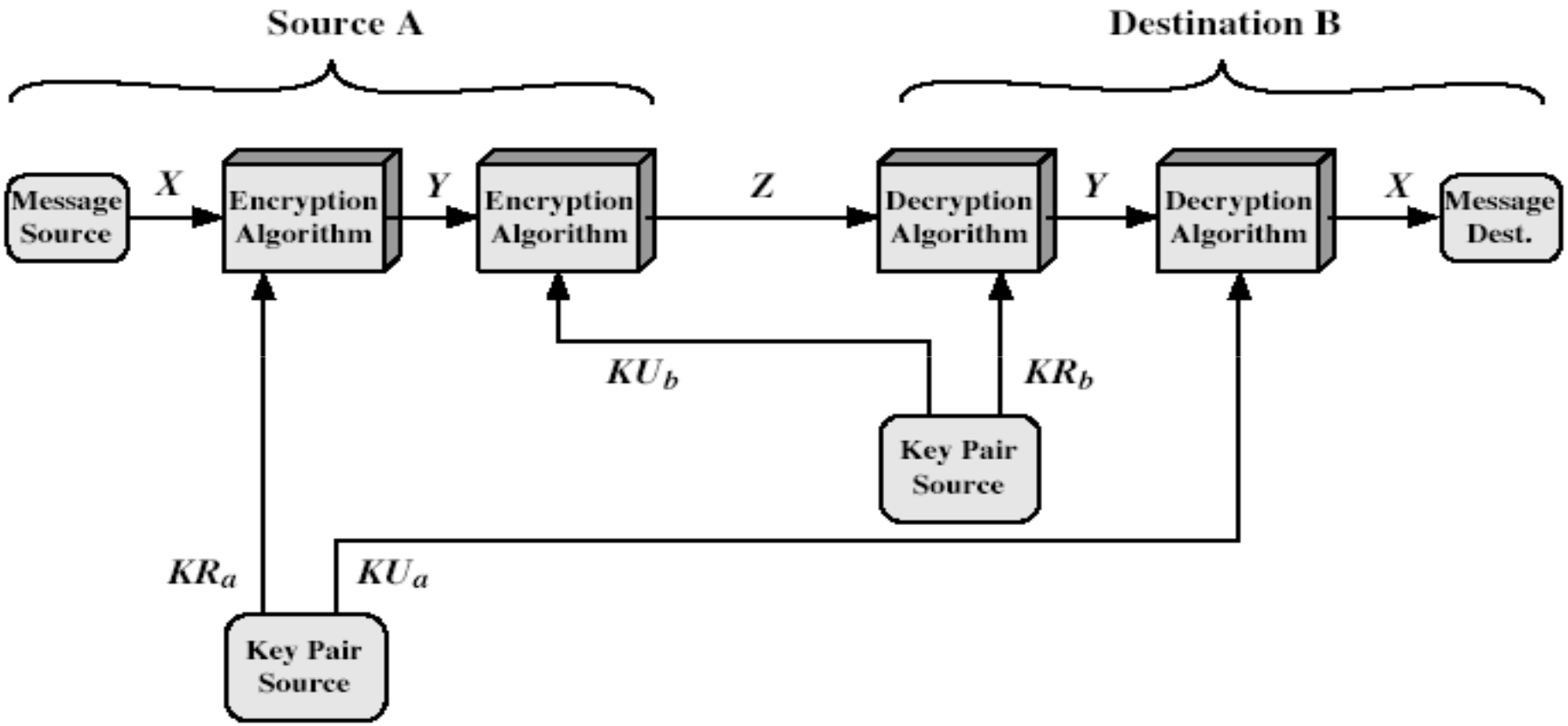


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication

Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, its just made too hard to do in practise
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite Galois field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA

- Approach
  - We choose p and q prime, very large
  - **n=pq**
  - **ø(n)** nombre d'entiers < n et premier avec n
  - **ø(pq) = (q-1)(p-1)**
  - on choisit **e < ø(n)** tel que e et ø(n) sont premiers entre eux
  - on choisit **d** tel que **ed = 1 mod ø(n)**
  - **KU = {e, n}**
  - **KR = {d, n}**
- Cryptage de M : **$C = M^e \bmod n$**
- Décryptage de **C : $M = C^d \bmod n$**
- Propriété : **$M = M^{ed} \bmod n$**

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - **p, q**
- computing their system modulus **N=p.q**
  - note **ø(N)=(p-1)(q-1)**
- selecting at random the encryption key e
    - where **1<e<ø(N), gcd(e,ø(N))=1**
- solve following equation to find decryption key d
  - **e.d=1 mod ø(N) and 0≤d≤N**
- publish their public encryption key: **KU={e,N}**
- keep secret private decryption key: **KR={d,N=pq}**

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient **KU={e,N}**
  - computes: **$C=M^e \bmod N$, where $0 \le M < N$**
- to decrypt the ciphertext C the owner:
  - uses their private key **KR={d,N=pq}**
  - computes: **$M=C^d \bmod N$**
- note that the message M must be smaller than the modulus N (block if needed)

Why RSA Works

- because of Euler's Theorem:
- $a^{\phi(n)} \bmod N = 1$
  - where gcd(a,N)=1
- in RSA have:
  - $N=p.q$
  - $\phi(N)=(p-1)(q-1)$
  - carefully chosen e & d to be inverses mod $\phi(N)$
  - hence $e.d=1+k.\phi(N)$ for some k
- hence :

$$C^d = (M^e)^d = M^{1+k.\phi(N)} = M^1.(M^{\phi(N)})^q = M^1.(1)^q = M^1 = M \bmod N$$

RSA Example

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n)=(p-1)(q-1)=16 \times 10=160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$
   Value is $d=23$ since $23 \times 7=161= 10 \times 160+1$
6. Publish public key $KU=\{7,187\}$
7. Keep secret private key $KR=\{23,187=17 \times 11\}$

RSA Example cont

- sample RSA encryption/decryption is:
- given message **M = 88** (nb. 88<187)
- encryption:

  $C = 88^7 \bmod 187 = 11$

- decryption:

  $M = 11^{23} \bmod 187 = 88$

Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation

$c \leftarrow 0$; $d \leftarrow 1$

for i $\leftarrow$ k **downto** 0

  do $c \leftarrow 2 \times c$

  $d \leftarrow (d \times d) \bmod n$

  if $b_i = 1$

    then $c \leftarrow c + 1$

    $d \leftarrow (d \times a) \bmod n$

return d

RSA Key Generation

- users of RSA must:
  - determine two primes at random - `p, q`
  - select either `e` or `d` and compute the other
- primes `p,q` must not be easily derived from modulus `N=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other

RSA Security

- three approaches to attacking RSA:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
  - timing attacks (on running of decryption)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor N=p.q, hence find ø(N) and then d
  - determine ø(N) directly and find d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of Aug-99 best is 130 decimal digits (512) bit with GNFS
  - biggest improvement comes from improved algorithm
    - cf "Quadratic Sieve" to "Generalized Number Field Sieve"
  - barring dramatic breakthrough 1024+ bit RSA secure
    - ensure p, q of similar size and matching other constraints

Timing Attacks

- developed in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Summary

- have considered:
  - principles of public-key cryptography
  - RSA algorithm, implementation, security

Chapter 10 – Key Management; Other Public Key Cryptosystems

Note that many slides were derived from lecture slides by Lawrie Brown

Key Management

- public-key encryption helps address key distribution problems
- have two aspects of this:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

Distribution of Public Keys

- can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

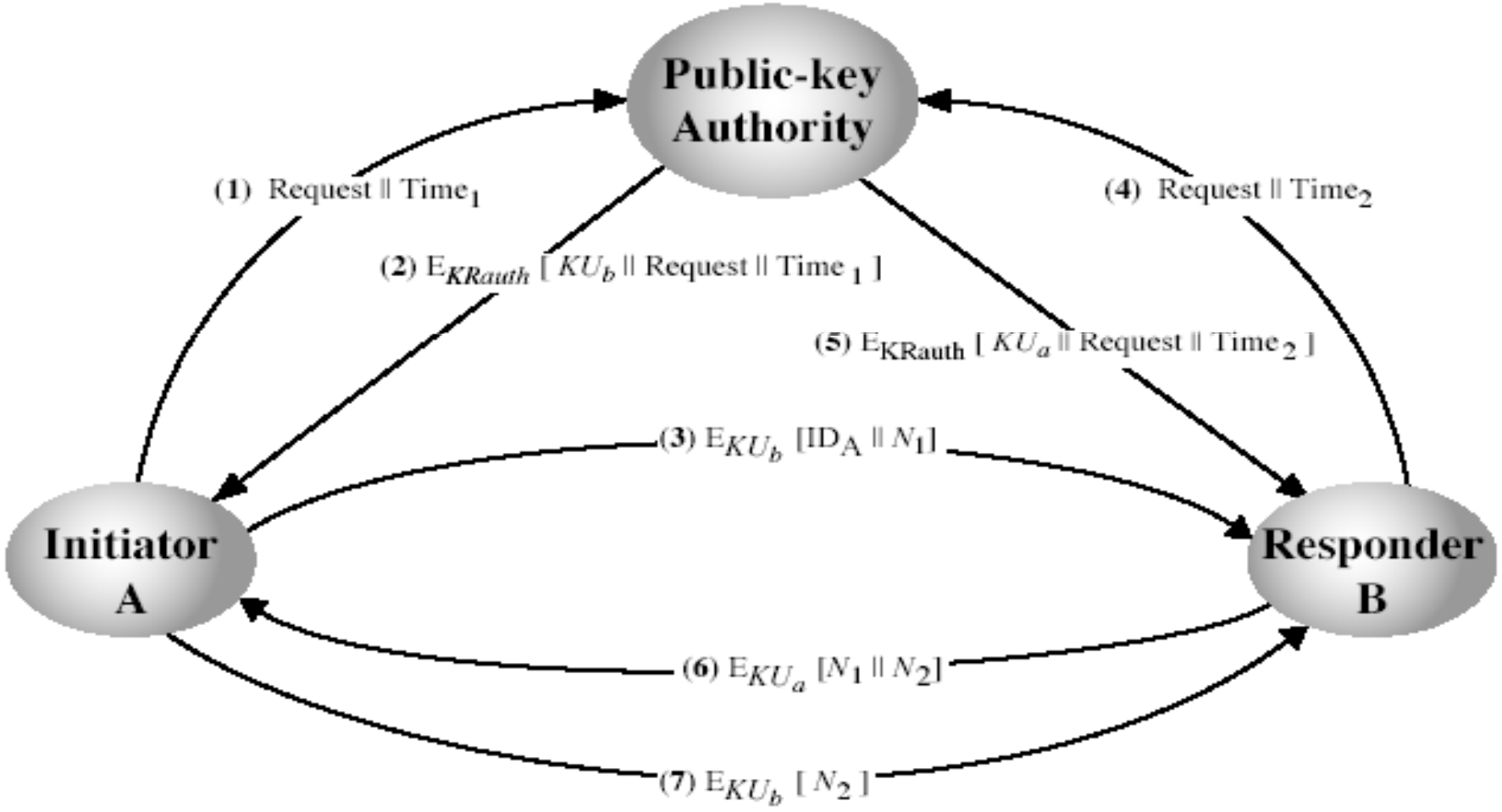
Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- still vulnerable to tampering or forgery

Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- has properties of directory
- and requires users to know public key for the directory
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed

Exchanging Key : One Scenario



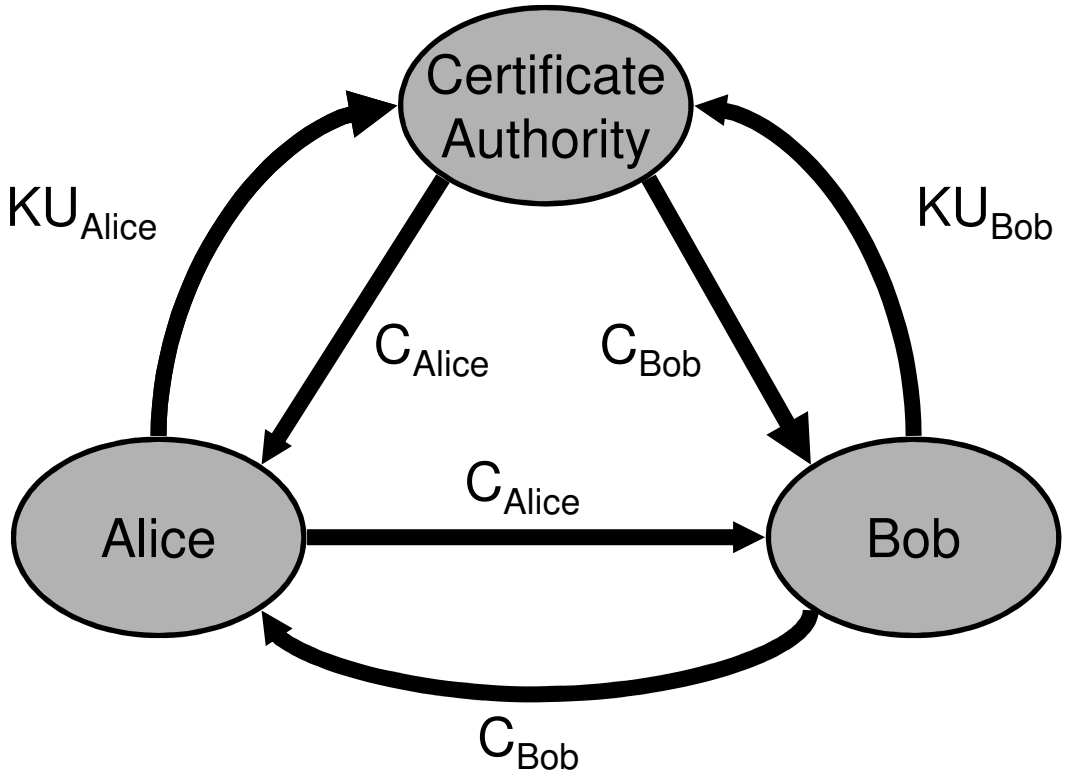
Note that many slides were derived from lecture slides by Lawrie Brown

Public-Key Certificates

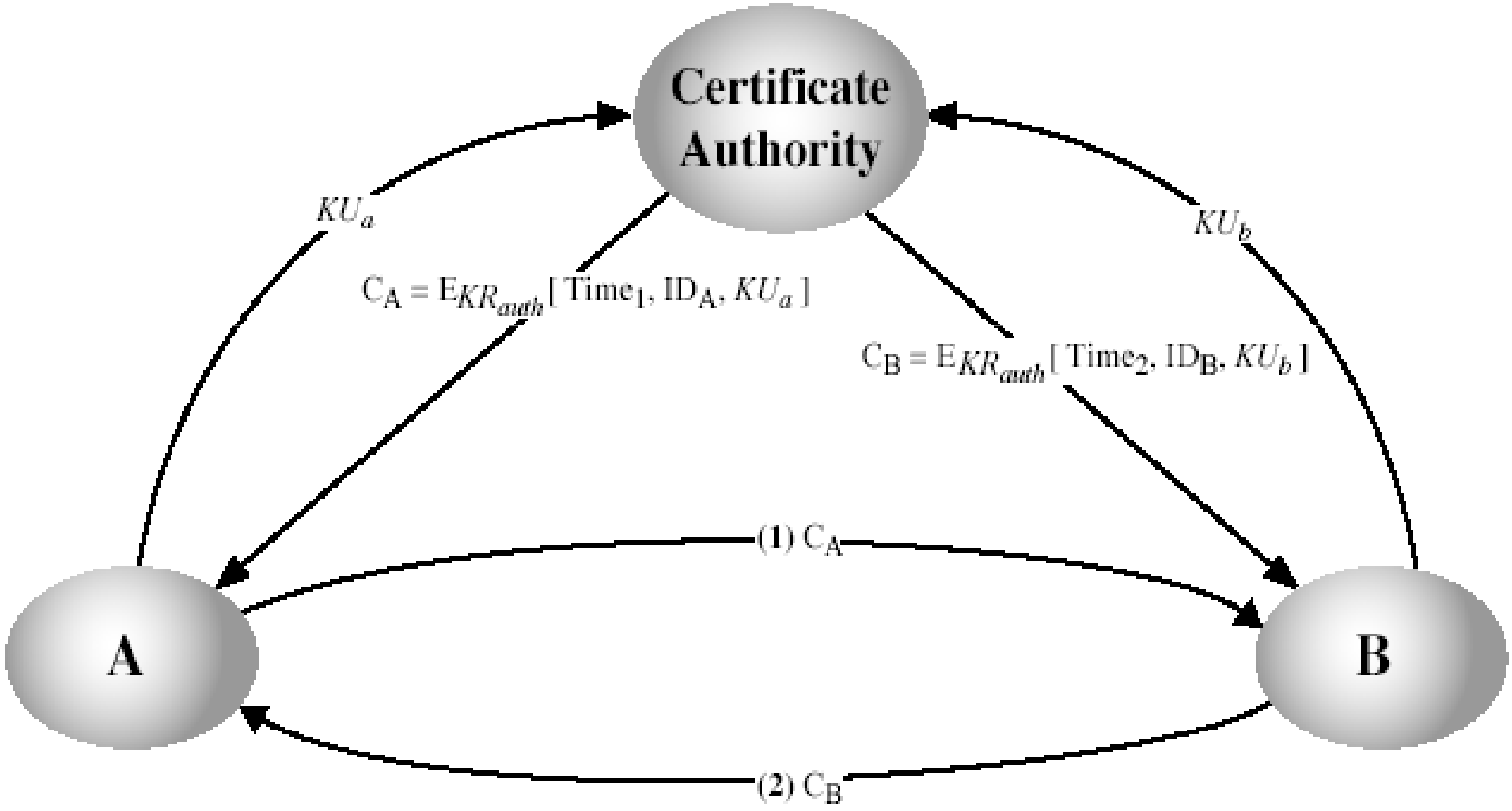
- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use etc
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key

Exchanging Key : Using Certificates

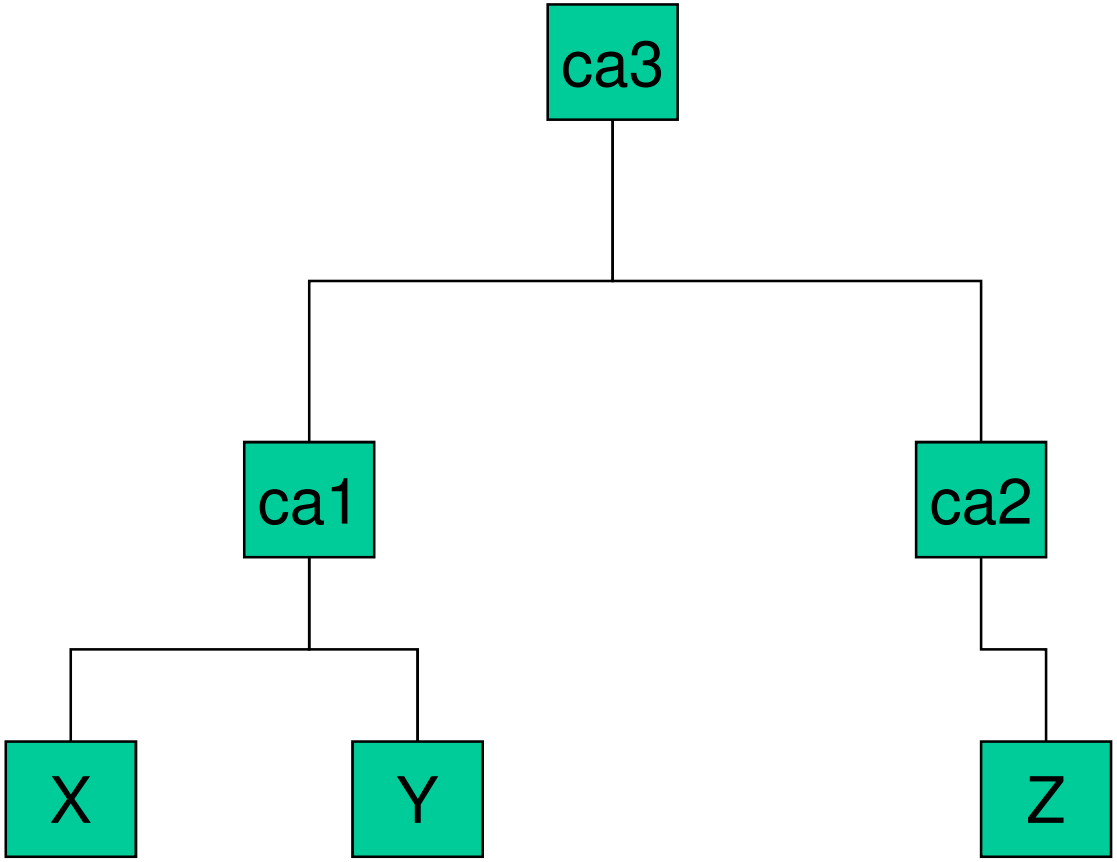
- $C_{Alice} = E_{KRCA}$(Date validité+ Identification Alice + KU_{Alice})
- $C_{bob} = E_{KRCA}$(Date validité+ Identification Bob + KU_{Bob})



Exchanging Key : Using Certificates



Certificate authority hierarchy



Public-Key Distribution of Secret Keys

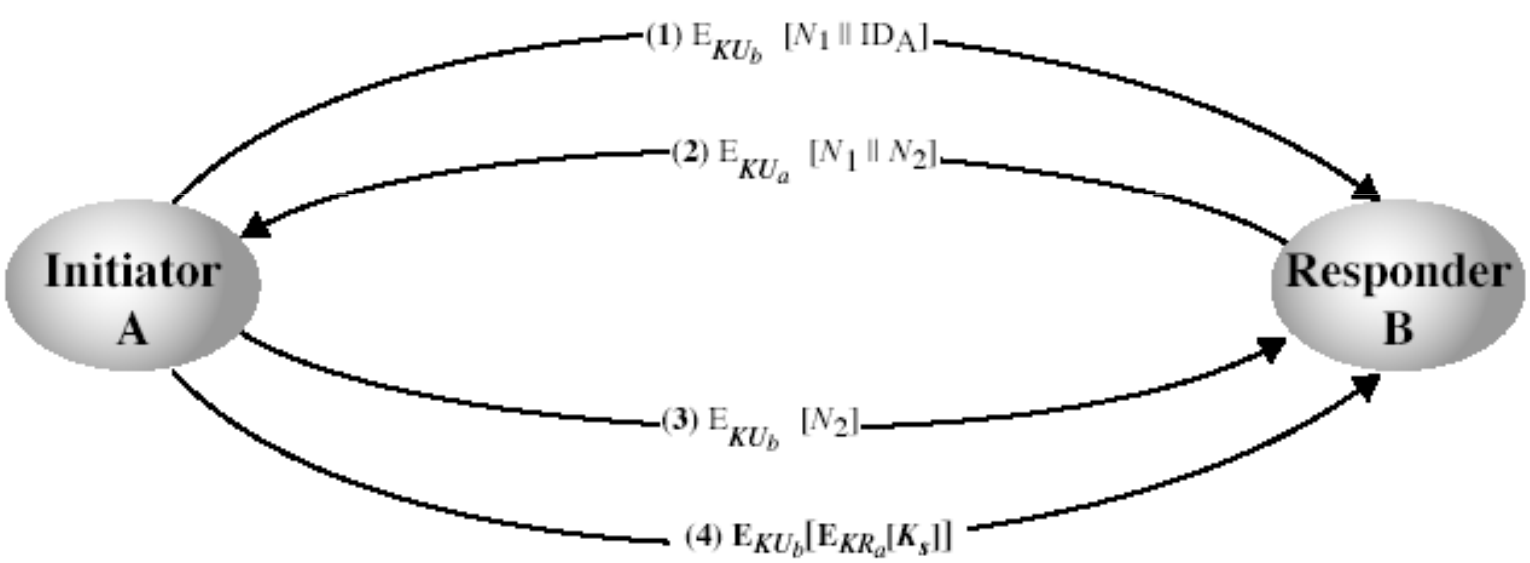
- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K sends it to A encrypted using the supplied public key
  - A decrypts the session key and both use
- problem is that an opponent can intercept and impersonate both halves of protocol

Public-Key Distribution of Secret Keys

- if have securely exchanged public-keys:



Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that James Ellis (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial **q**
  - **α** a primitive root **mod q**
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = \alpha^{x_A} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB}:

$$K_{AB} = \alpha^{x_A . x_B} \bmod q$$
$$= y_A^{x_B} \bmod q \quad \text{(which B can compute)}$$
$$= y_B^{x_A} \bmod q \quad \text{(which A can compute)}$$

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $\alpha=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute public keys:
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:

$$K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} = 160 \quad \text{(Alice)}$$

$$K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} = 160 \quad \text{(Bob)}$$

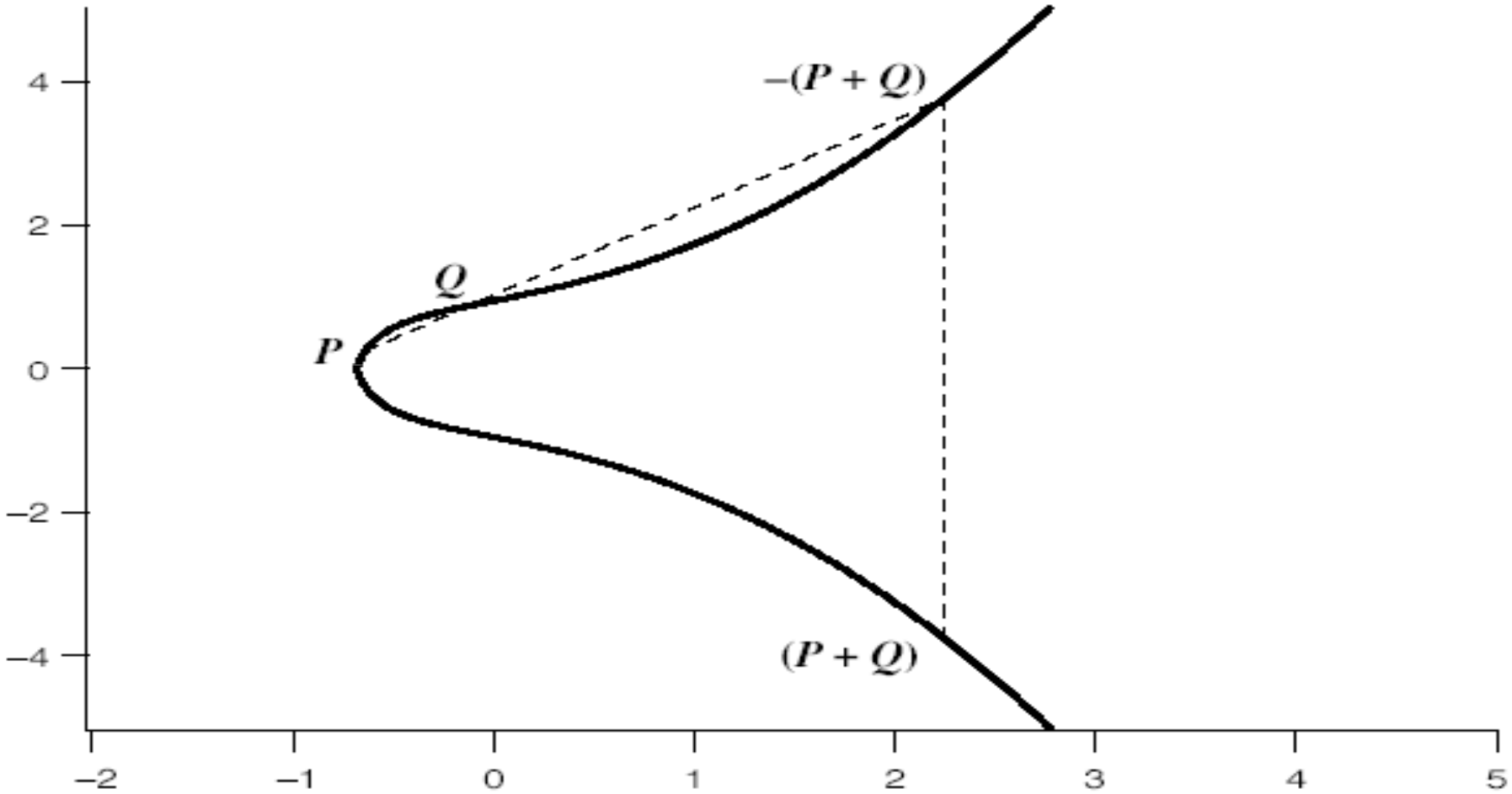
Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes

Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x,y,a,b are all real numbers
  - also define zero point O
- have **addition operation** for elliptic curve
  - geometrically sum of **Q+R** is reflection of intersection **R**

Real Elliptic Curve Example



(b) $y^2 = x^3 + x + 1$

Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
  - belong to finite field
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - $y^2 \bmod p = (x^3 + ax + b) \bmod p$
    - best in software
  - binary curves $E_{2^m}(a,b)$ defined over GF(2^n)
    - use polynomials with binary coefficients
    - best in hardware

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need "hard" problem equiv to discrete log
  - **Q=kP**, where Q,P belong to a prime curve
  - is "easy" to compute Q given k,P
  - but "hard" to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $\mathbf{E_{23}(9,17)}$

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_p(a,b)$
- select base point $G=(x_1,y_1)$ with large order n s.t. $nG=O$
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_A \times G$, $P_B=n_B \times G$
- compute shared key: $K=n_A \times P_B$, $K=n_B \times P_A$
  - same since $K=n_A \times n_B \times G$

ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message **M** as a point on the elliptic curve $\mathbf{P_m}$
- select suitable curve & point **G** as in D-H
- each user chooses private key $n_A < n$
- and computes public key $P_A = n_A \times G$
- to encrypt $\mathbf{P_m}$: $C_m = \{kG,\ P_m + kP_b\}$, k random
- decrypt C_m compute:

$$P_m + kP_b - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m$$

ECC Security

- relies on elliptic curve logarithm problem
- fastest method is "Pollard rho method"
- compared to factoring, can use much smaller key sizes than with RSA etc
- for equivalent key lengths computations are roughly equivalent
- hence for similar security ECC offers significant computational advantages

Summary

- have considered:
  - distribution of public keys
  - public-key distribution of secret keys
  - Diffie-Hellman key exchange
  - Elliptic Curve cryptography

Comparable Key Sizes for Equivalent Security

Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Fin