Lecture 24: Cryptography

Instructor: Mitch Neilsen

Office: N219D

Quote of the Day

“I understood the importance in principle of public key cryptography, but it's all moved much faster than I expected.

I did not expect it to be a mainstay of advanced communications technology.”

-- Whitfield Diffie

Chapter 15: Security

- The Security Problem
- Program Threats
- System and Network Threats
- **Cryptography as a Security Tool**
- User Authentication
- **Implementing Security Defenses**
- **Firewalls to Protect Systems and Networks**

Security Issues

Secrecy or **Confidentiality -** prevent unauthorized disclosure.

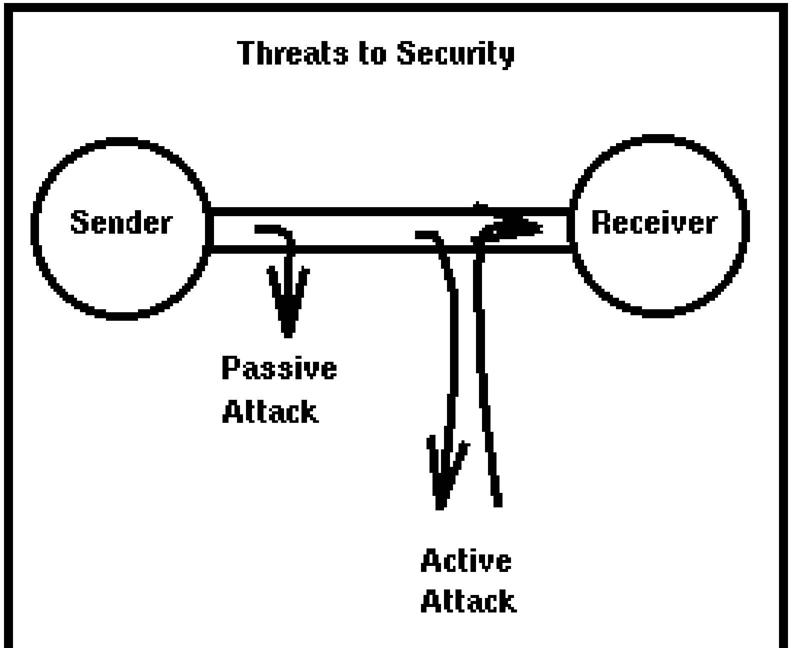
Authentication - ensure the identity of a user.

Integrity - prevent unauthorized modification of data.

Threats to Security

Passive Wiretapping - violates secrecy.

Active Wiretapping - violates integrity.



Cryptographic Algorithms

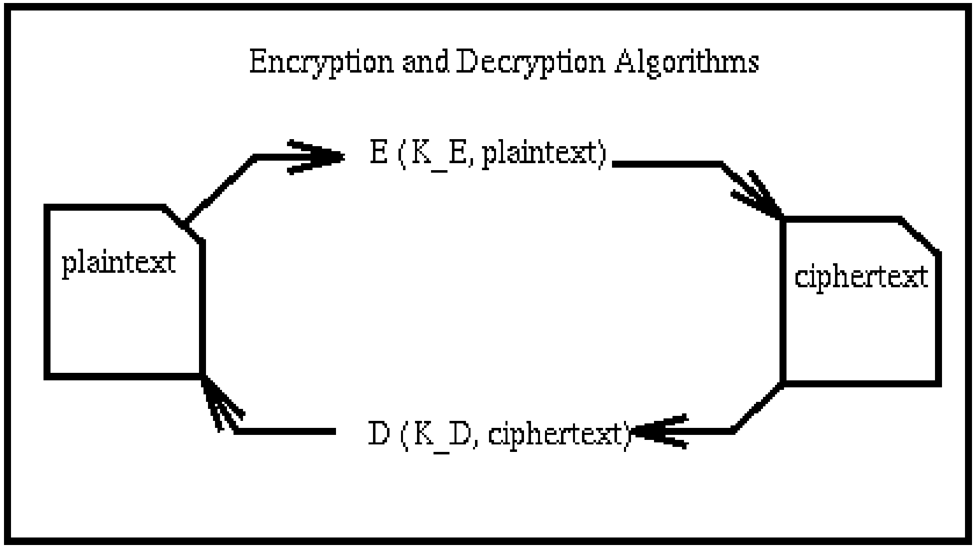
- **Cryptography** is the study of secret writing.
- **Cryptographic algorithms** can be used to help solve the problems of secrecy, authentication, and integrity.

Types of Cryptographic Algorithms

- **Hashing algorithms -** used for authentication and digital signatures.
- **Encryption algorithms -** used for secure voice and data transmission.

Encryption Algorithms

- User data (called plaintext) is encrypted before transmission.
- Upon receipt of encrypted data (called ciphertext), the recipient must decrypt the ciphertext to retrieve the original data (plaintext).



Cryptographic System Classification

Type of operation used to transform plaintext to ciphertext.

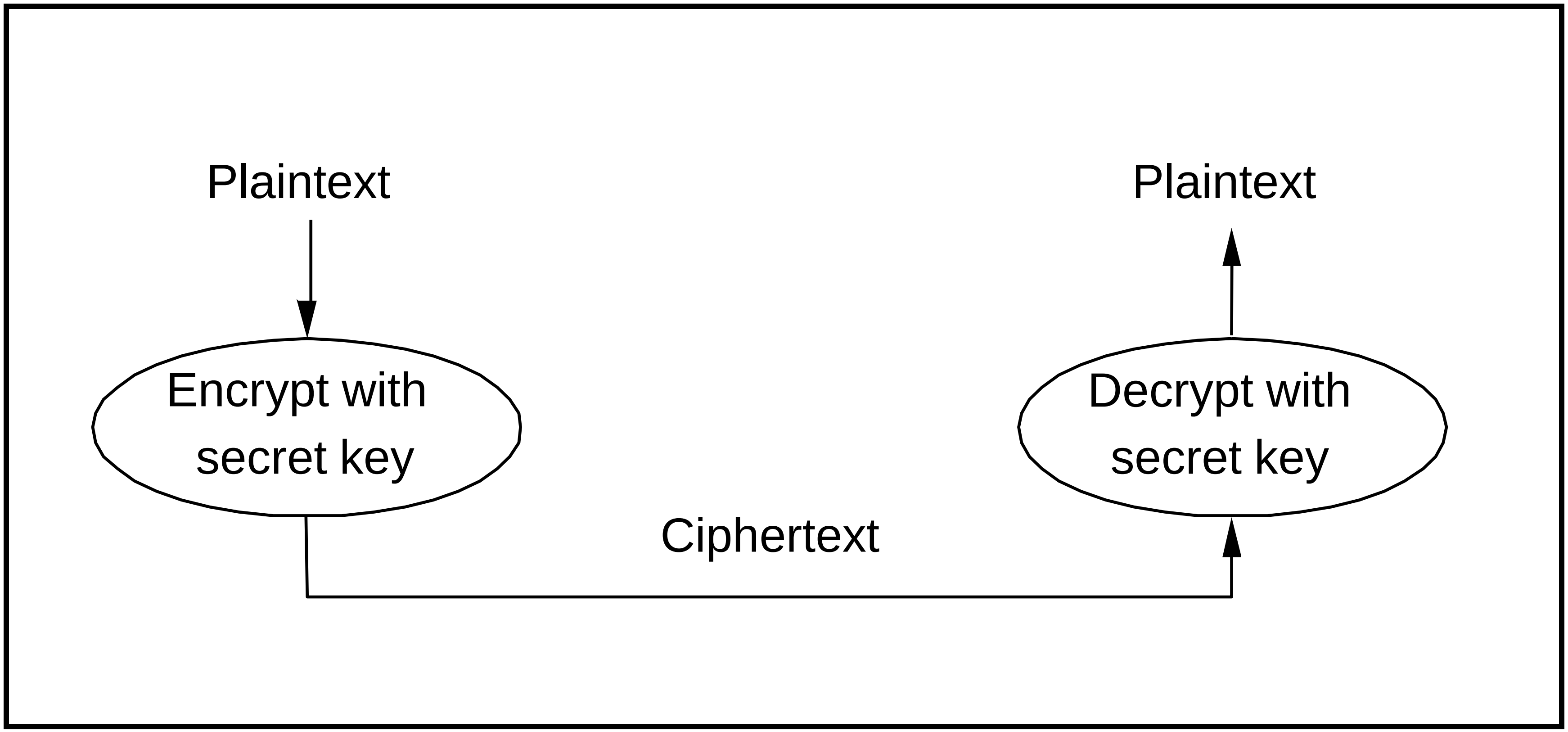
- **substitution** - replace each element in the plaintext by another element.
- **transposition** - rearrange the elements.
- **product systems use combinations of both operations.**

Number of keys used:

- **one key:** symmetric, single-key system ($K_E = K_D$).
- **two keys** (one for encryption and one for decryption): asymmetric, **two-key** (or **public-key**) system

  (K_E =/= K_D).

Single Key System (e.g., DES)



Substitution Ciphers

Substitution: Replace each symbol with another symbol.

Caesar Cipher:

- Let P = plaintext and C=ciphertext.
- Assign a numerical value to each letter in the text (a=0, b=1, c=2, ...).
- The encryption algorithm E(P) is given by: $C = E(P) = (P+k) \bmod 26$.

Example: ROT13 = E(P), if k=13.

Example

Question: Using k=2, compute the ciphertext for the following plaintext using a Caesar Cipher:

We are discovered, save yourself.

```
Yg ctg fkueqxgtgf, ucxg aqwtugnh.
```

Answer:

Yg ctg fkueqxgtgf, ucxg aqwtugnh.

Analysis: Caesar Cipher

A brute-force crypto-analysis of a Caesar cipher only requires us to test at most 25 keys before decrypting the ciphertext.

Thus, a Caesar cipher is far from secure.

Mono-alphabetic Ciphers

Mono-alphabetic Ciphers allow for an arbitrary substitution of the elements in the plaintext.

Example: **CryptoQuip** in newspaper.

Problem: A cryptanalyst can compare the relative frequency of letter occurence with the relative frequency of letters in English text to easily decipher.

Multiple-Letter Encryption

- **Multiple-Letter Encryption** allows different letters to be used for each instance of a letter in plaintext.
- **Example:** Playfair Algorithm - based on the use of a 5x5 matrix of letters constructed using a keyword. The Playfair Algorithm was used by the British Army in the First World War, and by the Allied Armies in World War II.

Playfair Algorithm

Multiple-Letter Encryption

Key = MONARCHY

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

Encrypt: This is the day.

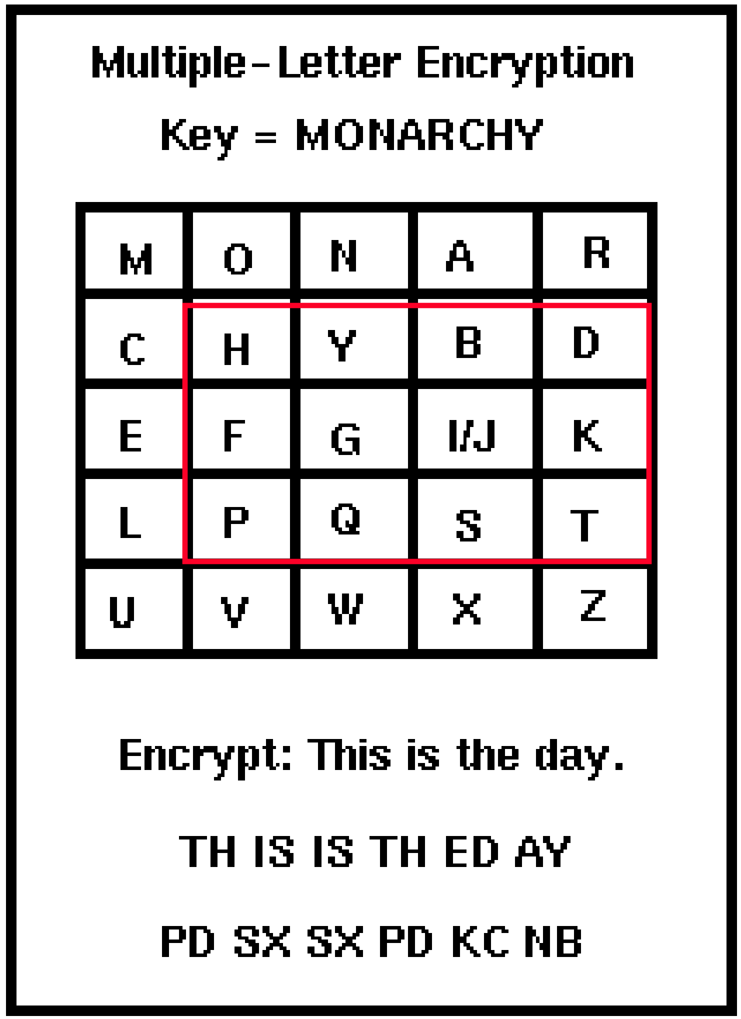
TH IS IS TH ED AY

PD SX SX PD KC NB

Playfair Algorithm

- Plaintext letters are paired up, and repeated letters in a pair are separated by a filler; e.g. "x".
- Plaintext letters that fall in the same row (column) are replaced by the next letters in that row (column) (by wrapping around if necessary).
- Any other letter is replaced by the letter that lies in its own row and the column occupied by the other letter in its pair.
- Playfair ciphertext can be easily broken using a computer.

Playfair Algorithm



Transposition Ciphers

- **Transposition Ciphers** rearrange the symbols without disguising them.
- **Example:** Rail fence cipher - write plaintext down as a sequence of columns, and read it off as a sequence of rows.
- **Another Transposition Cipher:**
  - Write a message down by row.
  - Use a key to determine the order in which the columns are read.

Transposition Cipher

Transposition Cipher

Key = 13425876

Plaintext = "THIS IS THE DAY TO WORK HARD"

1	3	4	2	5	8	7	6
T	H	I	S	I	S	T	H
E	D	A	Y	T	O	W	O
R	K	H	A	R	D	X	X

Ciphertext = "TERSYAHDKIAHITRHOXTWXSOD"

Example: Rotor Machines

- Sequentially apply several transpositions.
- Rotor machines were used by both Germany (enigma) and Japan (purple) during WWII.

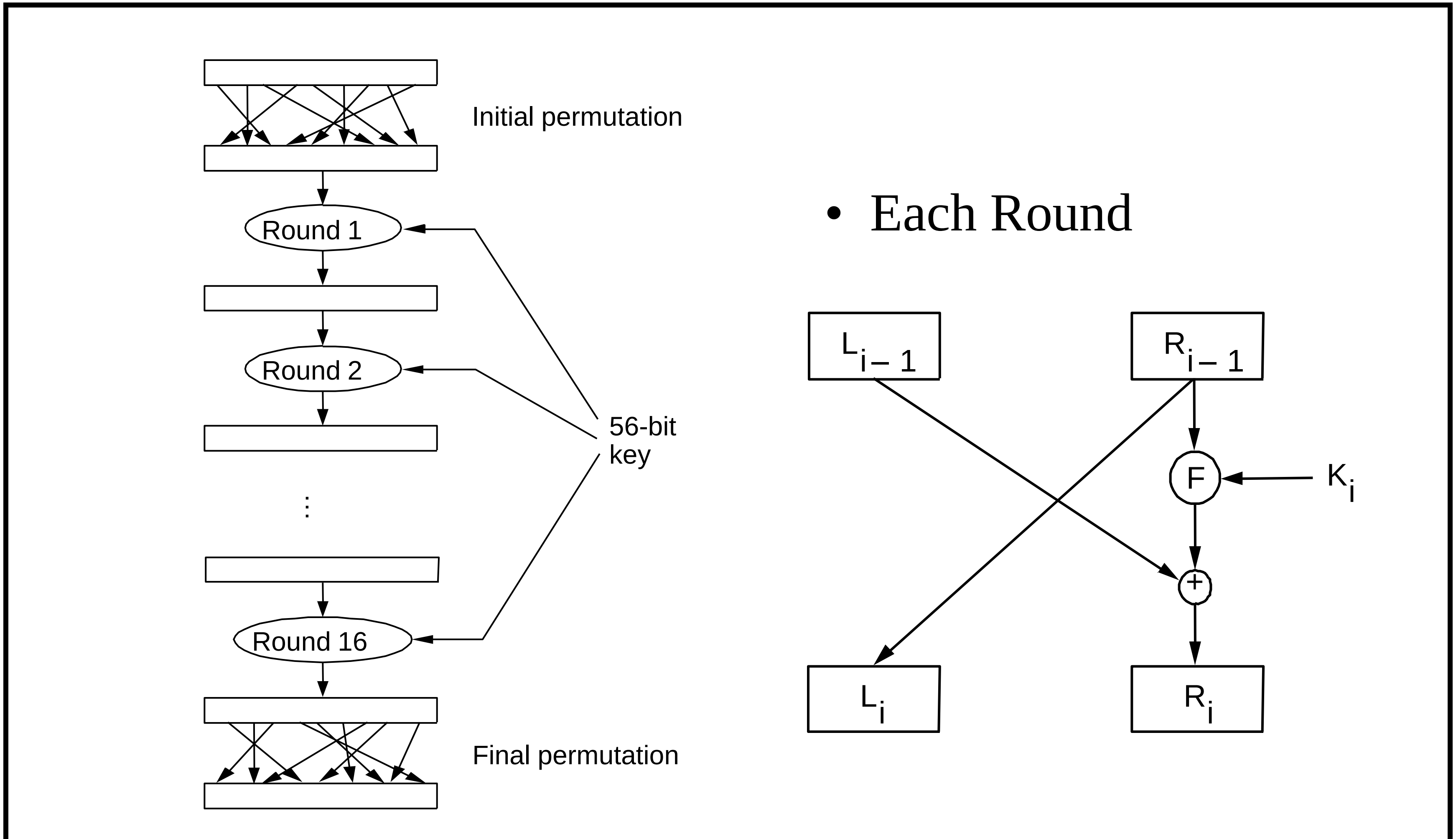
Data Encryption Standard (DES)

- The Data Encryption Standard is the most widely used encryption scheme and was adopted by the National Bureau of Standards in 1977 and extended for use in 1994.
- In 1999, it was recertified for legacy systems, along with Triple DES and Skipjack for new systems.
- DES uses a series of complex transpositions and substitutions.
- DES is based on a 56-bit key.

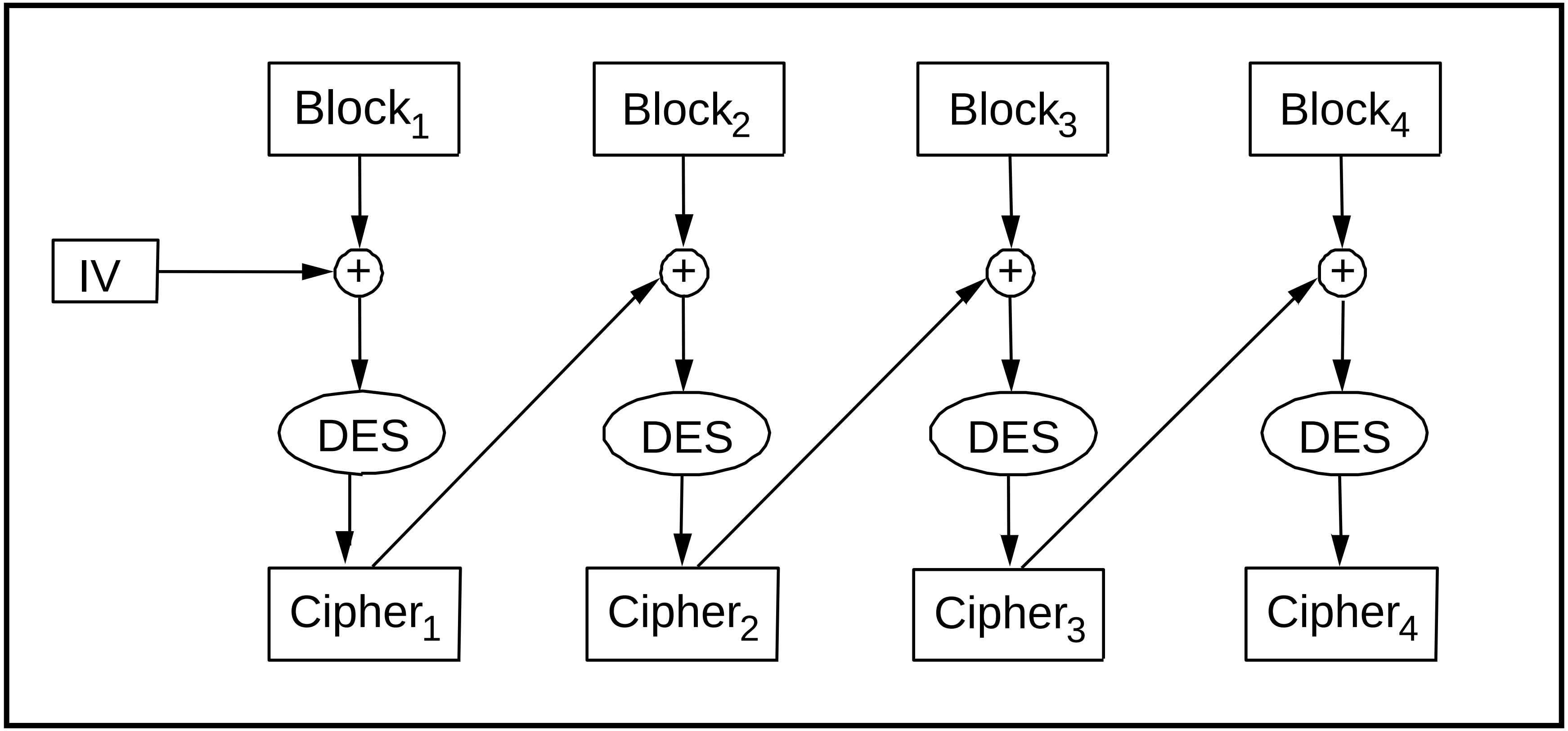
Data Encryption Standard (DES)

64-bit key (56-bits + 8-bit parity)

16 rounds



Repeat for larger messages



Example Single Key System:
Data Encryption Standard (DES)

History

- Developed by IBM, 1975
- Modified slightly by NSA
- U.S. Government (NIST) standard, 1977

Algorithm

- Uses 56-bit key (plus 8 parity bits)
- 16 “rounds”

  56-bit key used to generate 16 48-bit keys

  Each round does substitution and permutation using 8 S-boxes

Strength

- Difficult to analyze
- Cryptanalysis believed to be exponentially difficult in number of rounds
- No currently known attacks easier than brute force
- But brute force is now (relatively) easy

Other Ciphers

Triple-DES

- DES three times

  $m_c = E(D(E(m_p, k_1), k_2), k_3)$
- Effectively 112 bits
- Three times as slow as DES

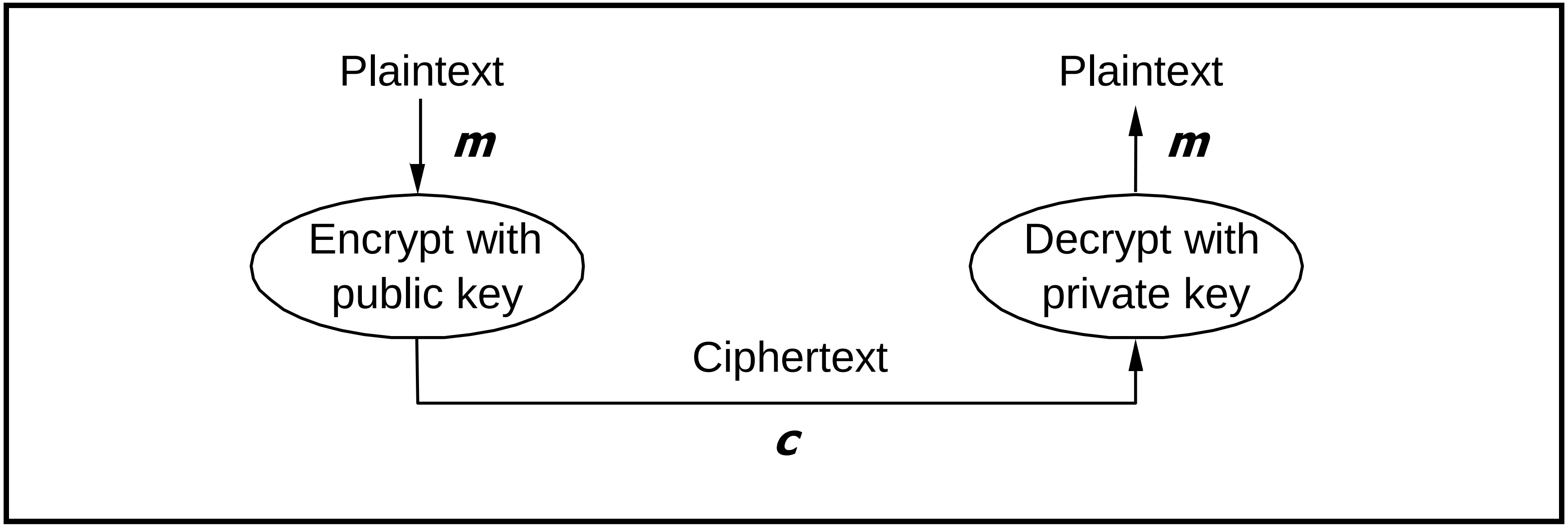
Blowfish

- Developed by Bruce Schneider circa 1993
- Variable key size from 32 to 448 bits
- Very fast on large general purpose CPUs (modern PCs)
- Not very easy to implement in small hardware

Advanced Encryption Standard (AES)

- Selected by NIST as replacement for DES in 2001
- Uses the Rijndael algorithm
- Keys of 128, 192 or 256 bits
- E.g., TrueCrypt

Two-Key System (e.g., RSA)



Encryption (m to c) and Decryption (c to m) :

$$c = m^{e} \bmod n$$

$$m = c^{d} \bmod n$$

Applications

- **Encryption/decryption** - encrypt a message using the recipient's public key - ensures secrecy.
- **Digital signature** - a sender "signs" a message with its private key by encrypting a hashing code generated from the data.

Public-Key Cryptosystem

- Each user has a public key (K_U) and a private key (K_R).
- Depending on the goal, the data may be encrypted using K_R and decrypted using K_U, or the data may be encrypted using K_U and decrypted using K_R.

Rivest-Shamir-Adleman (RSA)

- The only widely accepted and implemented approach for public-key encryption.
- A block cipher in which plaintext and ciphertext are integers from 0 to n-1 (for some large n).

RSA (cont.)

- Choose two large prime numbers *p* and *q*.
- Multiply *p* and *q* together to get *n*.
- Choose an encryption key *e*, such that *e* and Euler's totient function $\mathbf{\Phi(n) = (p - 1) * (q - 1)}$ are relatively prime. Two numbers are relatively prime if they have no common factor greater than one.
- Compute decryption key *d* such that

$$d = e^{-1} \bmod ((p - 1) * (q - 1))$$

$$d * e \bmod ((p - 1)*(q - 1)) = 1$$

- Construct public key K_U as (***e, n***).
- Construct private key K_R as (***d, n***).
- Discard (do not disclose) original primes *p* and *q*.

Example

Let p=7 and q=17; so, n=119.

Calculate $\Phi(n) = (p-1)*(q-1) = 6*16 = 96$.

Select e = 5 (note gcd(5,96)=1).

Determine d = 77. Note:5*77=385=4*96+1.

$K_U = (5,119)$, $K_R = (77,119)$.

Now, encrypt *m* = 19.

$$\boldsymbol{c = m^{e} \bmod n}$$

$$\boldsymbol{m = c^{d} \bmod n}$$

Since e = 5, compute $m^5 = 19^5 = 2{,}476{,}099$; and 2,476,099 mod 119 = 66. So, c = 66.

To decrypt, compute $m = c^{77} \bmod 119 = 19$.

$66^{77} \bmod 119$

$66^2 = 66*66 = 4356$, $4356 \bmod 119 = 72$

$66^4 \bmod 119 = 72^2 \bmod 119 = 5184 \bmod 119 = 67$

$66^8 \bmod 119 = 67^2 \bmod 119 = 4489 \bmod 119 = 86$

$66^{16} \bmod 119 = 86^2 \bmod 119 = 7396 \bmod 119 = 18$

$66^{32} \bmod 119 = 18^2 \bmod 119 = 324 \bmod 119 = 86$

$66^{64} \bmod 119 = 86^2 \bmod 119 = 18$

$66^{77} \bmod 119 = 66^{64} * 66^8 * 66^4 * 66^1 \bmod 119$

$= 18 * 86 * 67 * 66 \bmod 119$

$= 6{,}845{,}256 \bmod 119$

$= 19$

Problem

Let p=5 and q=17; so, n=85.

Calculate $\Phi(n) = (p-1)*(q-1) = 4*16 = 64$.

Select e = 3 (note gcd(3, 64)=1).

Determine d =

$K_U = (3, 85)$, $K_R = ($ __ $, 85)$.

Now, encrypt *m* = 25.

$$\boldsymbol{c = m^{e} \bmod n}$$

$$\boldsymbol{m = c^{d} \bmod n}$$

Since e = 3, compute $m^3 = 25^3 = 15{,}625$; and 15625 mod 85 = 70.

So, c = 70.

To decrypt, compute *m* = *c* — mod 85 = __ .

Problem

Let p=5 and q=17; so, n=85.

Calculate $\Phi(n) = (p-1)*(q-1) = 4*16 = 64$.

Select e = 3 (note gcd(3, 64)=1).

Determine d =

K_U = (3, 85), K_R = (**43** , 85).

Now, encrypt *m* = 25.

$$c = m^e \bmod n = 25^3 \bmod 85$$

$$m = c^d \bmod n$$

Since e = 3, compute $m^3 = 25^3$ = 15,625; and 15625 mod 85 = 70.

So, c = 70.

To decrypt, compute $m = c^{43}$ mod 85 = 25 .

Secret Key Authentication

Alice wants to talk to Bob

- Needs to convince him of her identity
- Both have same single (secret) key k

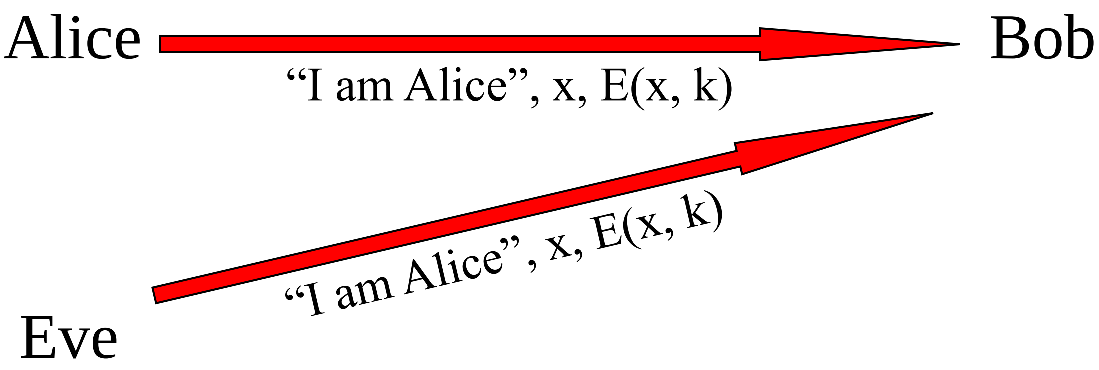
Naive scheme

Alice $\xrightarrow{\text{“I am Alice”, x, E(x, k)}}$ Bob

Vulnerability?

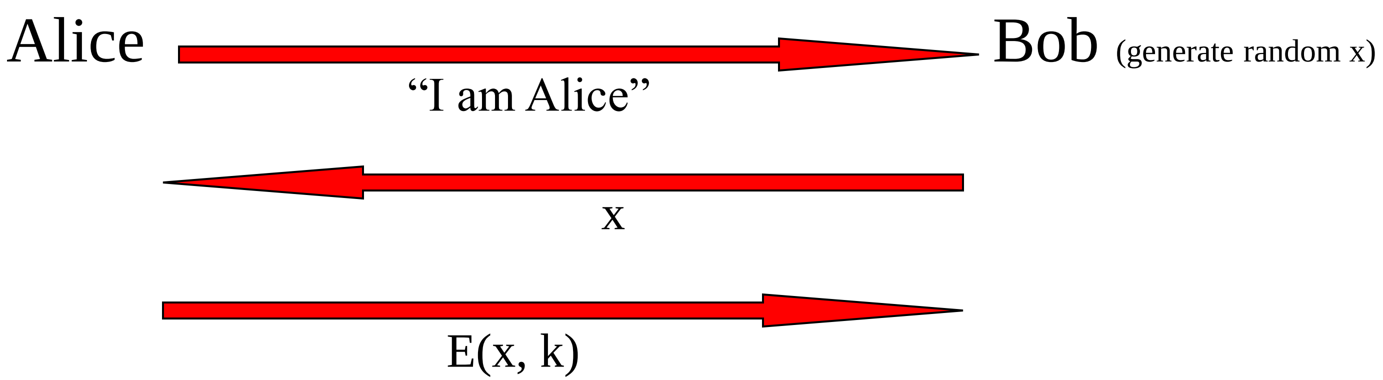
Replay Attack

Eve can listen in and impersonate Alice later



Preventing Replay Attacks

Bob can issue a challenge phrase to Alice

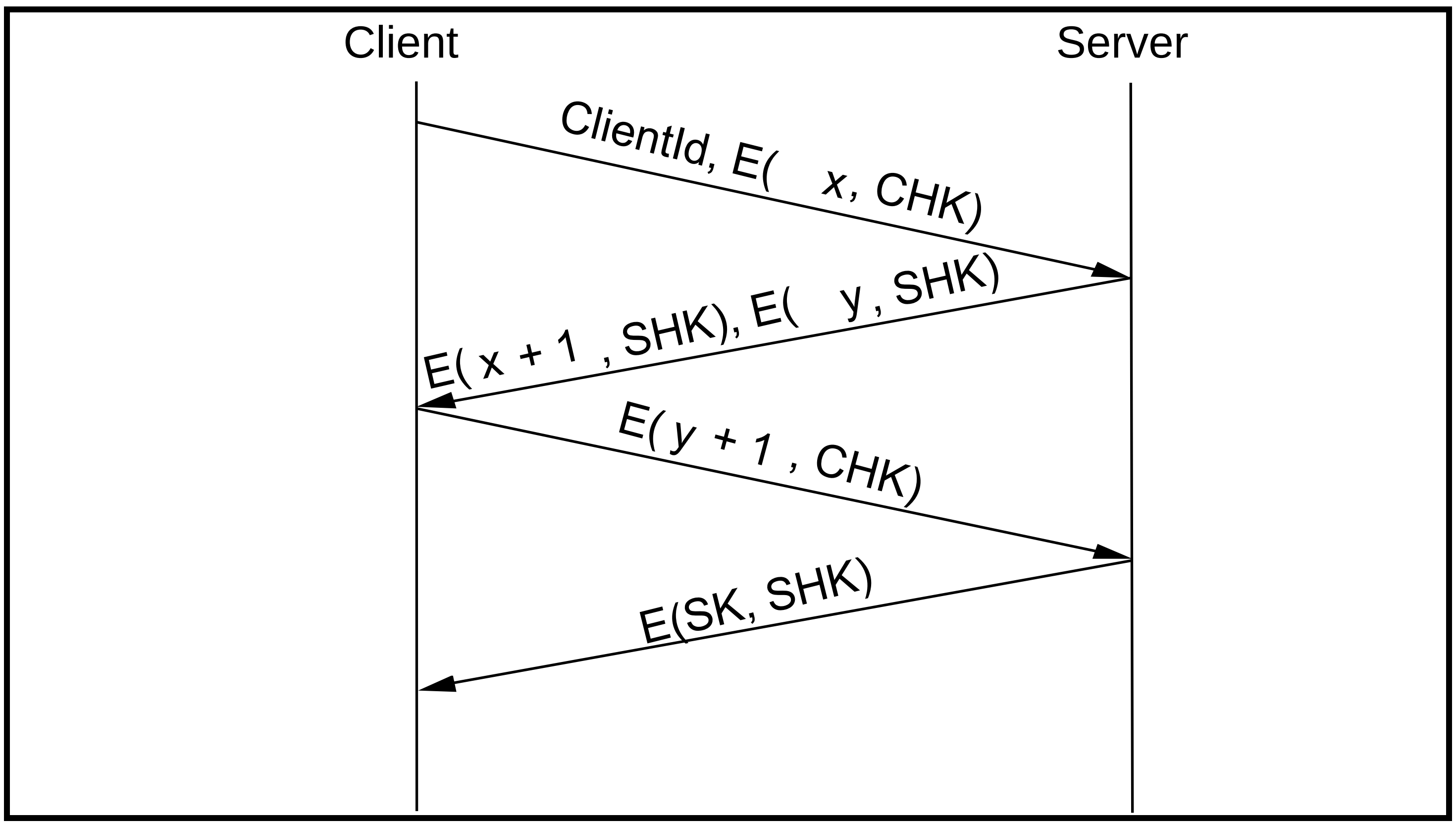


Authentication Protocols

Three-way handshake: E(msg, key) – encrypt msg with key

CHK = Client handshake key,   SHK = server handshake key

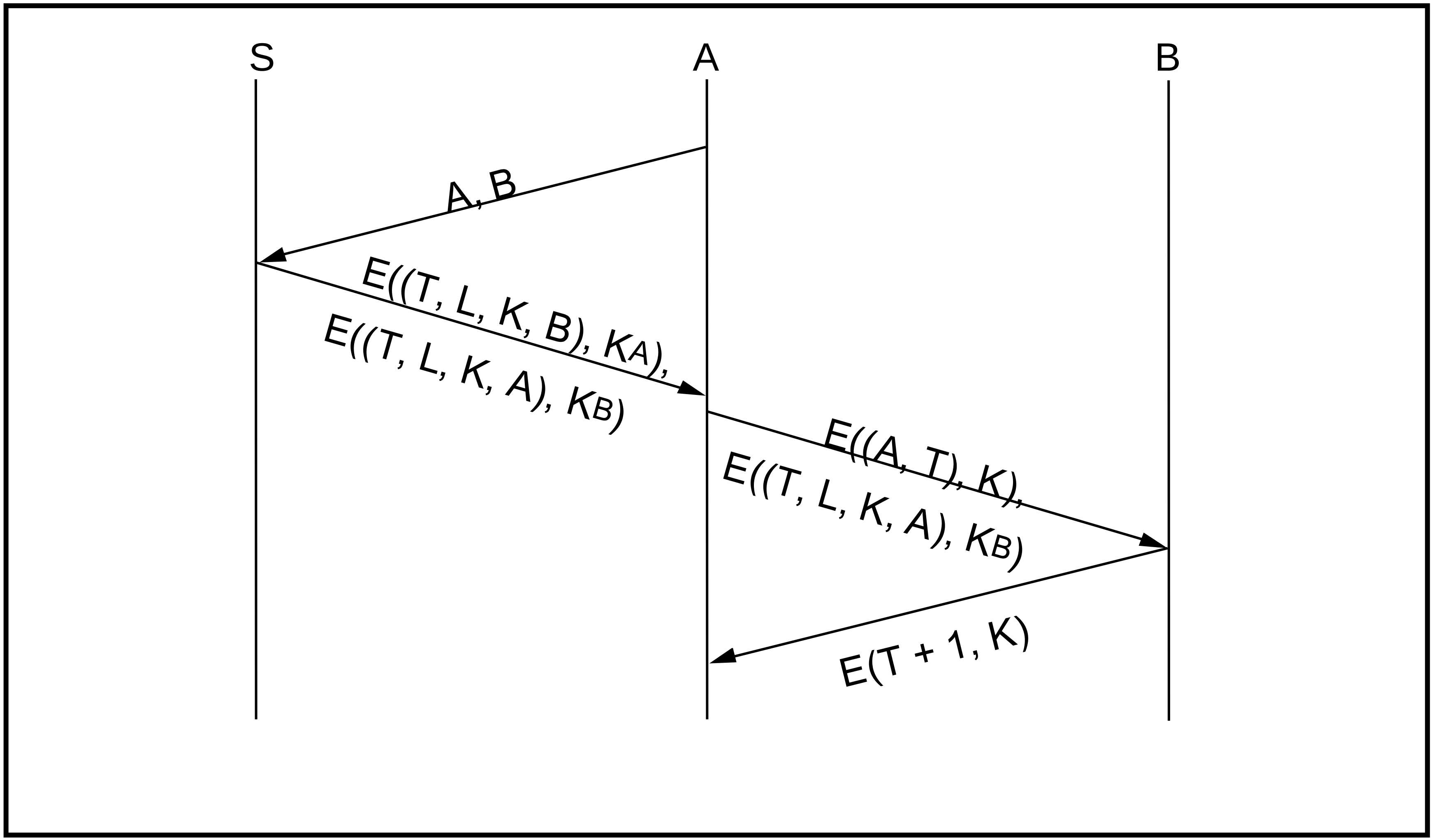
SK = Secret session key



Authentication Protocols (cont.)

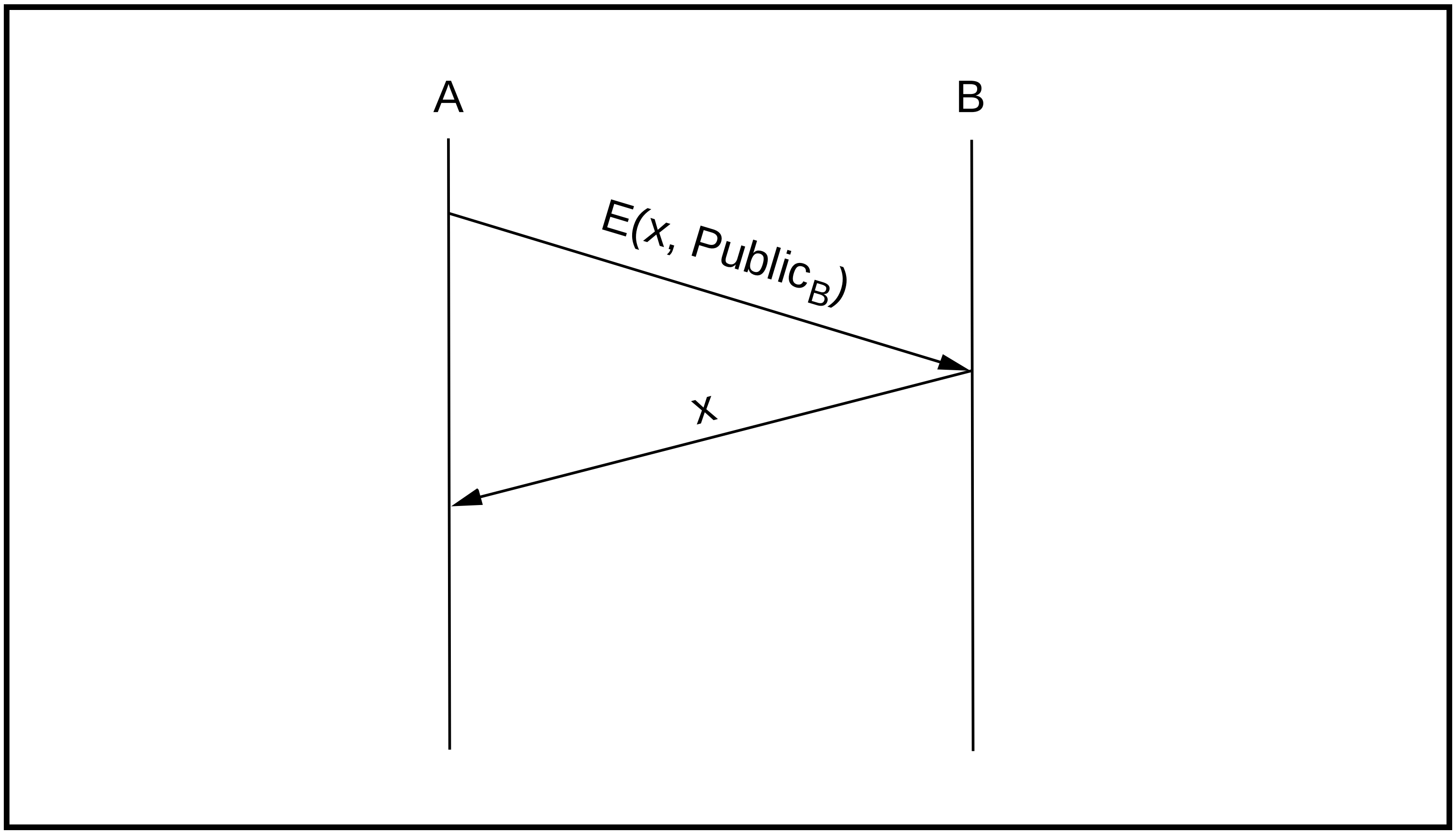
Trusted third party: S = authentication server using Kerberos

T = timestamp, L = lifetime, K = session key



Public Key Authentication

x = random value, $Public_B$ = B's public key
B authenticates itself by sending x back to A



Cryptographic Hash Functions

Given arbitrary length m, compute constant length digest d = h(m)

Desirable properties

- h(m) easy to compute given m
- One-way: given h(m), hard to find m
- Weakly collision free: given h(m) and m, hard to find m’ s.t. h(m) = h(m’)
- Strongly collision free: hard to find any x, y s.t. h(x) = h(y)

Example use: password database, file distribution

Common algorithms: MD5, SHA

Message Digest (MD5)

Cryptographic checksum

- just as a regular checksum protects the receiver from accidental changes to the message, a cryptographic checksum protects the receiver from malicious changes to the message.

One-way function

- given a cryptographic checksum for a message, it is virtually impossible to figure out what message produced that checksum; it is not computationally feasible to find two messages that hash to the same cryptographic checksum.

Relevance

- if you are given a checksum for a message and you are able to compute exactly the same checksum for that message, then it is highly likely this message produced the checksum you were given.

Message Integrity Protocols

Digital signature using RSA

- special case of a message integrity where the code can only have been generated by one participant
- compute signature with private key and verify with public key

Keyed MD5

- sender: $\boldsymbol{m} + \textbf{MD5}(\boldsymbol{m} + \boldsymbol{k}) + \textbf{E}(\boldsymbol{k}, \boldsymbol{k_{private}})$
- receiver

  recovers random key $\boldsymbol{k}$ using the sender’s public key $\boldsymbol{k_{public}}$

  applies MD5 to the concatenation of this random key with the message

MD5 with RSA signature

- sender: $\boldsymbol{m} + \textbf{E}(\textbf{MD5}(\boldsymbol{m}), \boldsymbol{k_{private}})$
- receiver

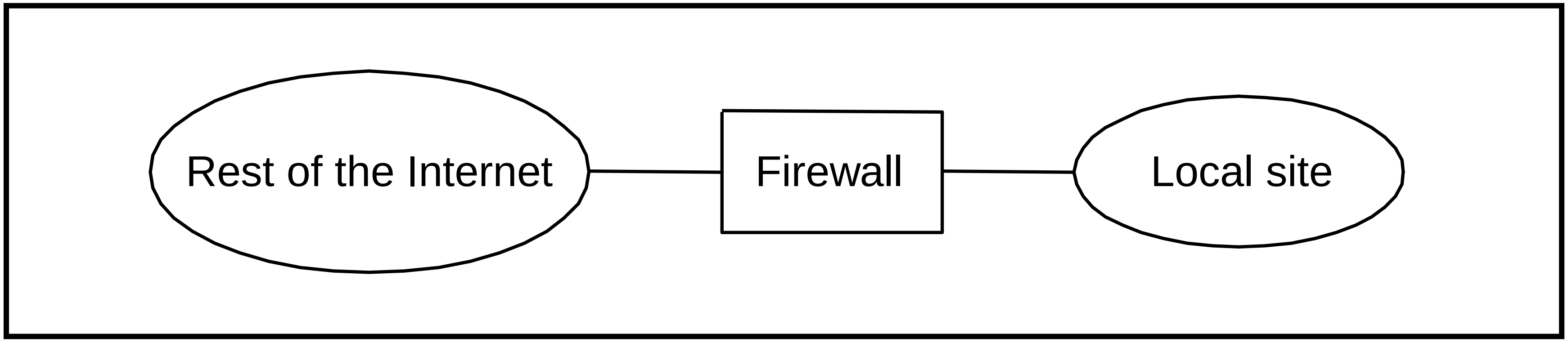
  decrypts signature with sender’s public key

  compares result with MD5 checksum sent with message

Firewalls

- Basic problem – many network applications and protocols have security problems that are fixed over time
  - Difficult for users to keep up with changes and keep host secure
  - Solution
    - Administrators limit access to end hosts by using a firewall
    - Firewall and limited number of machines at site are kept up-to-date by administrators

Firewalls



Filter-Based Solution

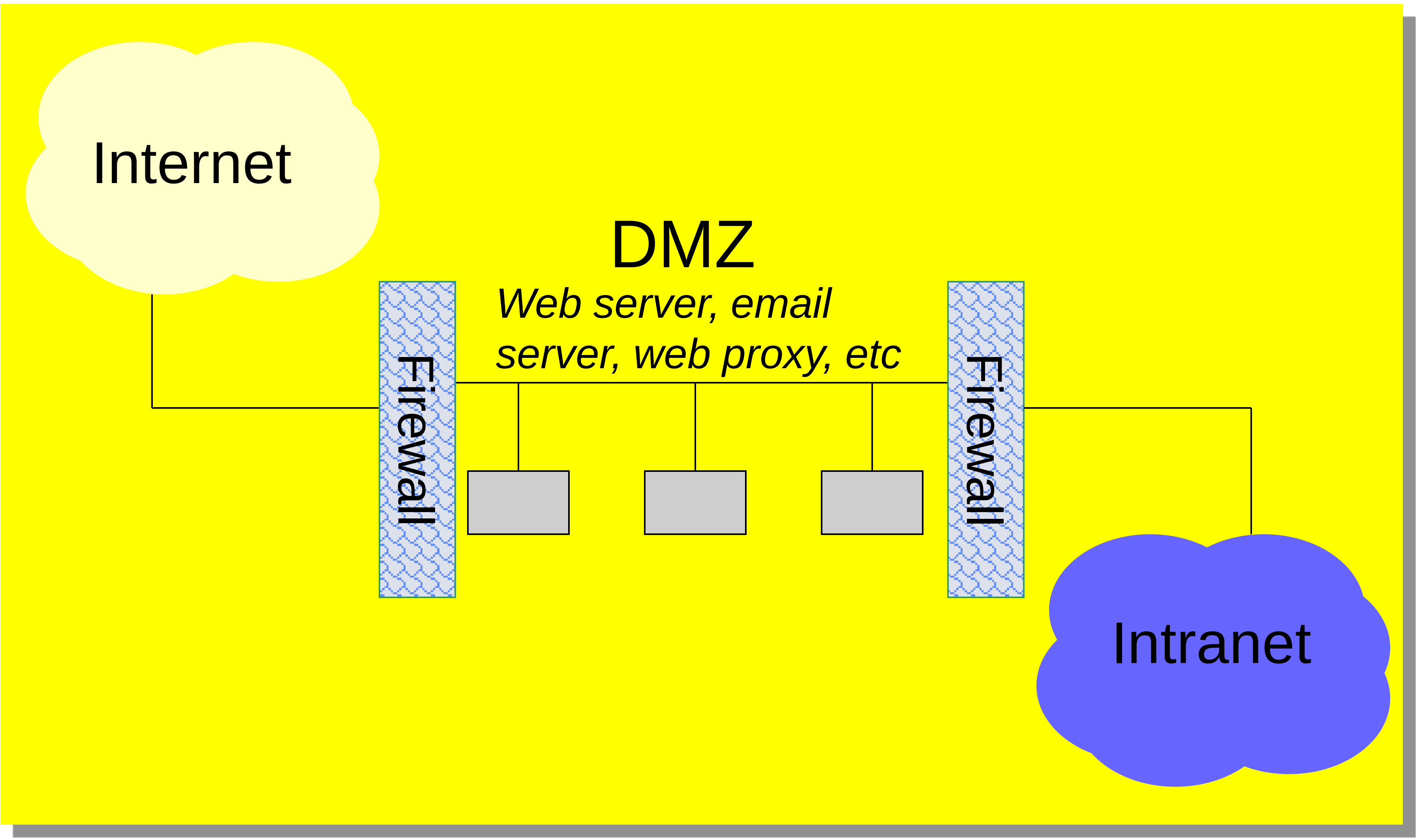
- example

  (128.130.10.5, 1234, 129.130.10.16, 80)

  (*,*, 129.130.10.16, 80)

- default: forward or not forward?
- how dynamic?

Typical Firewall Topology



Access Control

- **Discretionary Access Control -** restricting access to objects based on the identity of subjects and/or groups to which they belong; e.g., Unix perms. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (either directly or indirectly) on to any other subject; e.g., chmod.
- **Mandatory Access Control -** the operating system constrains access to objects.

DAC vs. MAC

- **Most people familiar with discretionary access control (DAC)**
  - Unix permission bits are an example
  - Might set a file permissions so that only group 'friends' can read it
- **Discretionary means anyone with access can propagate information:**
  - Mail sigint@enemy.gov < private
- **Mandatory Access Control (MAC)**
  - Security administrator can restrict propagation
  - Abbreviated MAC (NOT to be confused w. Message Authentication Code or Medium Access Control)

Bell-Lapadula model

- **View the system as subjects accessing objects**
  - The system input is requests, the output is decisions
  - Objects can be organized in one or more hierarchies, *H* (a tree enforcing the type of descendents)
- **Four modes of access are possible:**
  - execute – no observation or alteration
  - read – observation
  - append – alteration
  - write – both observation and modification
- **The current access set, *b*, is (subj, obj, attr) triples**
- **An access matrix *M* encodes permissible access types (as before, subjects are rows, objects columns)**

Security levels

- **A *security level* is a** (c, s) **pair:**
  - c = classification – E.g., unclassified, secret, top secret
  - s = category-set – E.g., Nuclear, Crypto
- (c_1, s_1) ***dominates*** (c_2, s_2) **iff** $c_1 \geq c_2$ **and** $s_2 \subseteq s_1$
  - L_1 *dominates* L_2 sometimes written $L_1 \sqsupseteq L_2$ or $L_2 \sqsubseteq L_1$
  - levels then form a *lattice* (partial order w. lub & glb)
- **Subjects and objects are assigned security levels**
  - level(S), level(O) – security level of subject/object
  - current-level(S) – subject may operate at lower level
  - level(S) bounds current-level(S) (current-level(S) $\sqsubseteq$ level(S))
  - Since level(S) is max, sometimes called S's *clearance*

Security properties

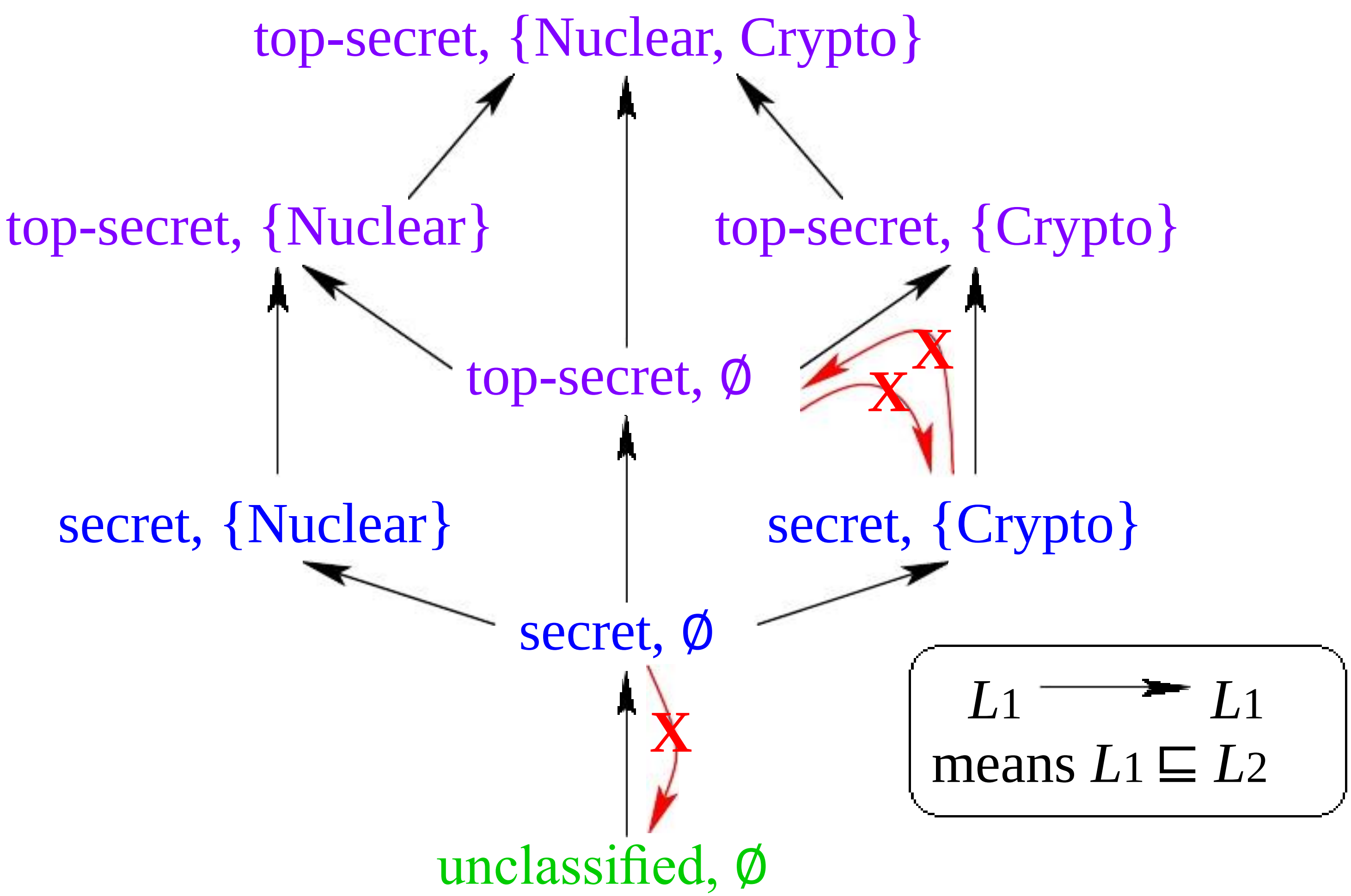
- **The simple security or *ss-property*:**
  - For any $(S, O, A) \in b$, if A includes observation, then level(S) must dominate level(O)
  - E.g., an unclassified user cannot read a top-secret document
- **The star security or **-property*:**
  - If a subject can observe O_1 and modify O_2 , then level(O_2) dominates level(O_1)
  - E.g., cannot copy top secret file into secret file
  - More precisely, given $(S, O, A) \in b$:

    if $A = r$ then current-level(S) $\sqsupseteq$ level(O) ("no read up")

    if $A = a$ then current-level(S) $\sqsubseteq$ level(O) ("no write down")

    if $A = w$ then current-level(S) = level(O)

The lattice model



- **Information can only flow up the lattice**
  - System enforces "No read up, no write down"
  - Think of $\sqsubseteq$ as "can flow to" relation

Straw-man MAC implementation

- **Take an ordinary Unix system**
- **Put labels on all files and directories to track levels**
- **Each user *U* has a security clearance,** level(*U*)
- **Determine current security level dynamically**
  - When U logs in, start with lowest curent-level
  - Increase current-level as higher-level files are observed (sometimes called a *floating label* system)
  - If U's level does not dominate current-level, kill program
  - Kill program that writes to file that doesn't dominate it
- **Is this secure?**

No: Covert channels

- **System rife with *storage channels***
  - Low current-level process executes another program
  - New program reads sensitive file, gets high current-level
  - High program exploits covert channels to pass data to low
- **E.g., High program inherits file descriptor**
  - Can pass 4-bytes of information to low prog. in file offset
- **Other storage channels:**
  - Exit value, signals, file locks, terminal escape codes, . . .
- **If we eliminate storage channels, is system secure?**

No: Timing channels

- **Example: CPU utilization**
  - To send a 0 bit, use 100% of CPU in busy-loop
  - To send a 1 bit, sleep and relinquish CPU
  - Repeat to transfer more bits
- **Example: Resource exhaustion**
  - High prog. allocates all physical memory if bit is 1
  - If low prog. slow from paging, knows less memory available
- **More examples: Disk head position, processor cache/TLB polution, . . .**

Reducing covert channels

- **Observation: Covert channels come from sharing**
  - If you have no shared resources, no covert channels
  - Extreme example: Just use two computers (common in DoD)
- **Problem: Sharing needed**
  - E.g., read unclassified data when preparing classified
- **Approach: Strict partitioning of resources**
  - Strictly partition and schedule resources between levels
  - Occasionally reapportion resources based on usage
  - Do so infrequently to bound leaked information
  - In general, only hope to bound bandwidth of covert channels
  - Approach still not so good if many security levels possible

Declassification

- **Sometimes need to prepare unclassified report from classified data**
- **Declassification happens outside of system**
  - Present file to security officer for downgrade
- **Job of declassification often not trivial**
  - E.g., Microsoft word saves a lot of undo information
  - This might be all the secret stuff you cut from document
  - Another bad mistake: Redacted PDF using black censor bars over or under text (but text still selectable)

Biba integrity model

- **Problem: How to protect integrity**
  - Suppose text editor gets trojaned, subtly modifies files, might mess up attack plans
- **Observation: Integrity is the converse of secrecy**
  - In secrecy, want to avoid writing less secret files
  - In integrity, want to avoid writing higher-integrity files
- **Use integrity hierarchy parallel to secrecy one**
  - Now *security level* is a c, i, s triple, i =integrity
  - $c_1, i_1, s_1 \sqsubseteq c_2, i_2, s_2$ iff $c_1 \leq c_2$ and $i_1 \geq i_2$ and $s_1 \subseteq s_2$
  - Only trusted users can operate at low integrity levels
  - If you read less authentic data, your current integrity level gets lowered (putting you up higher in the lattice), and you can no longer write higher-integrity files

DoD Orange Book

- **DoD requirements for certification of secure systems**
- **Four Divisions:**
  - D – been through certification and not secure
  - C – discretionary access control
  - B – mandatory access control
  - A – like B, but better verified design
  - Classes within divisions increasing level of security

Limitations of Orange book

- **How to deal with floppy disks, removable storage?**
- **How to deal with networking?**
- **Takes too long to certify a system**
  - People don't want to run *n*-year-old software
- **Doesn't fit non-military models very well**
- **What if you want high assurance & DAC?**

Today: Common Criteria

- **Replaced orange book around 1998**
- **Three parts to CC:**
  - CC Documents, including protection profiles with both functional and assurance requirements
  - CC Evaluation Methodology
  - National Schemes (local ways of doing evaluation)

Summary

- Read Ch. 14-15
- Project #3 – new due date, Mon., Dec. 16
  - Sensor Input Application
  - Kernel modifications – add new system call