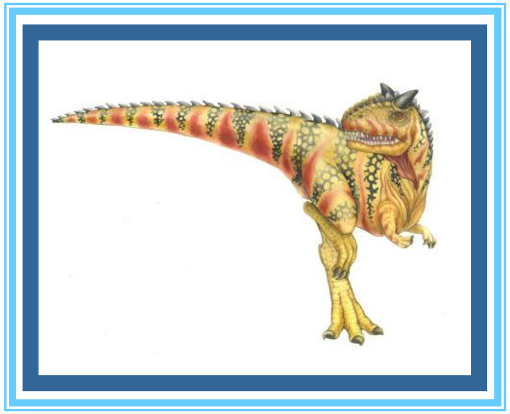
# Chapter 16:  Security



# The Security Problem

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

# Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use

# Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication
- **Privilege escalation**
  - Common attack type with access beyond what a user or resource is supposed to have

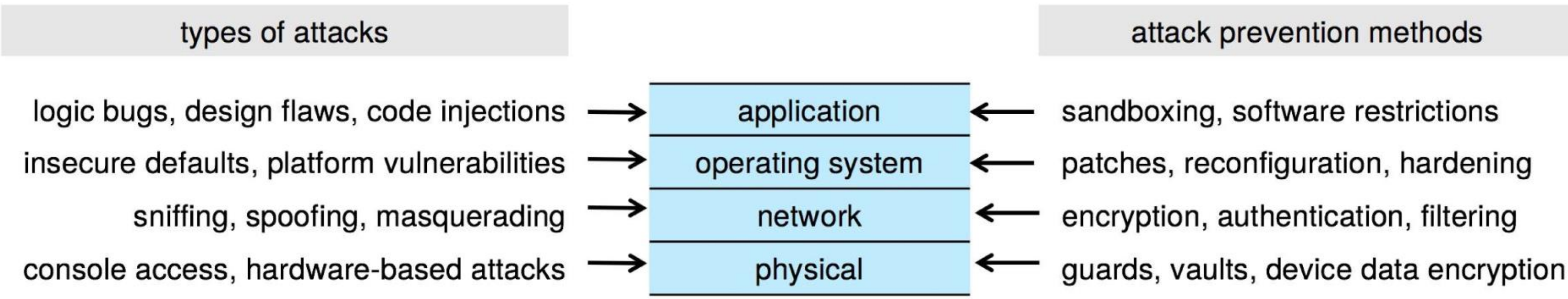
# Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Application**
    - Benign or malicious apps can cause security problems
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- Humans a risk too via **phishing** and **social-engineering** attacks
- But can too much security be a problem?

# Program Threats

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Spyware**, **pop-up browser windows**, **covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

# Four-layered Model of Security



# Program Threats (Cont.)

- **Malware -** Software designed to exploit, disable, or damage computer
- **Trojan Horse** – Program that acts in a clandestine manner
  - **Spyware** – Program frequently installed with legitimate software to display adds, capture user data
  - **Ransomware** – locks up data via encryption, demanding payment to unlock it
- Others include trap doors, logic boms
- All try to violate the Principle of Least Privilege

> ***THE PRINCIPLE OF LEAST PRIVILEGE***
>
> "The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur."—Jerome H. Saltzer, describing a design principle of the Multics operating system in 1974: https://pdfs.semanticscholar.org/1c8d/06510ad449ad24fbdd164f8008cc730cab47.pdf.

- Goal frequently is to leave behind Remote Access Tool (RAT) for repeated access

# C Program with Buffer-overflow Condition

```
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
   char buffer[BUFFER SIZE];
   if (argc < 2)
       return -1;
   else {
       strcpy(buffer,argv[1]);
       return 0;
   }
}
```

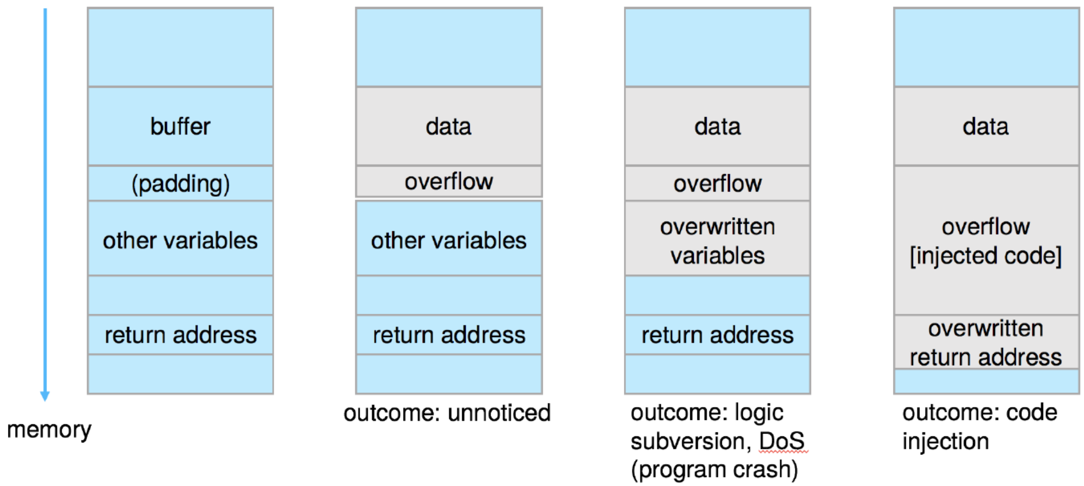
- **Code review** can help – programmers review each other's code, looking for logic flows, programming flaws

# Code Injection

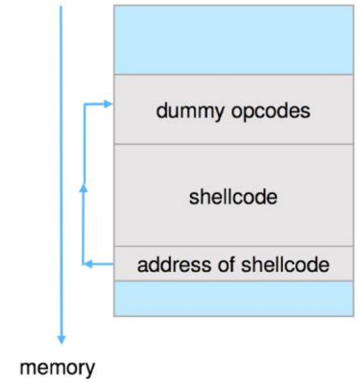
- **Code-injection attack** occurs when system code is not malicious but has bugs allowing executable code to be added or modified
  - Results from poor or insecure programming paradigms, commonly in low level languages like C or C++ which allow for direct memory access through pointers
  - Goal is a buffer overflow in which code is placed in a buffer and execution caused by the attack
  - Can be run by script kiddies – use tools written but exploit identifiers

# Code Injection (Cont.)

- Outcomes from code injection




- Frequently use trampoline to code execution to exploit buffer overflow:



# Great Programming Required?

- For the first step of determining the bug, and second step of writing exploit code, yes
- **Script kiddies** can run pre-written exploit code to attack a given system
- Attack code can get a shell with the processes' owner's permissions
  - Or open a network port, delete files, download a program, etc.
- Depending on bug, attack can be executed across a network using allowed connections, bypassing firewalls
- Buffer overflow can be disabled by disabling stack execution or adding bit to page table to indicate "non-executable" state
  - Available in SPARC and x86
  - But still have security exploits

# Program Threats (Cont.)

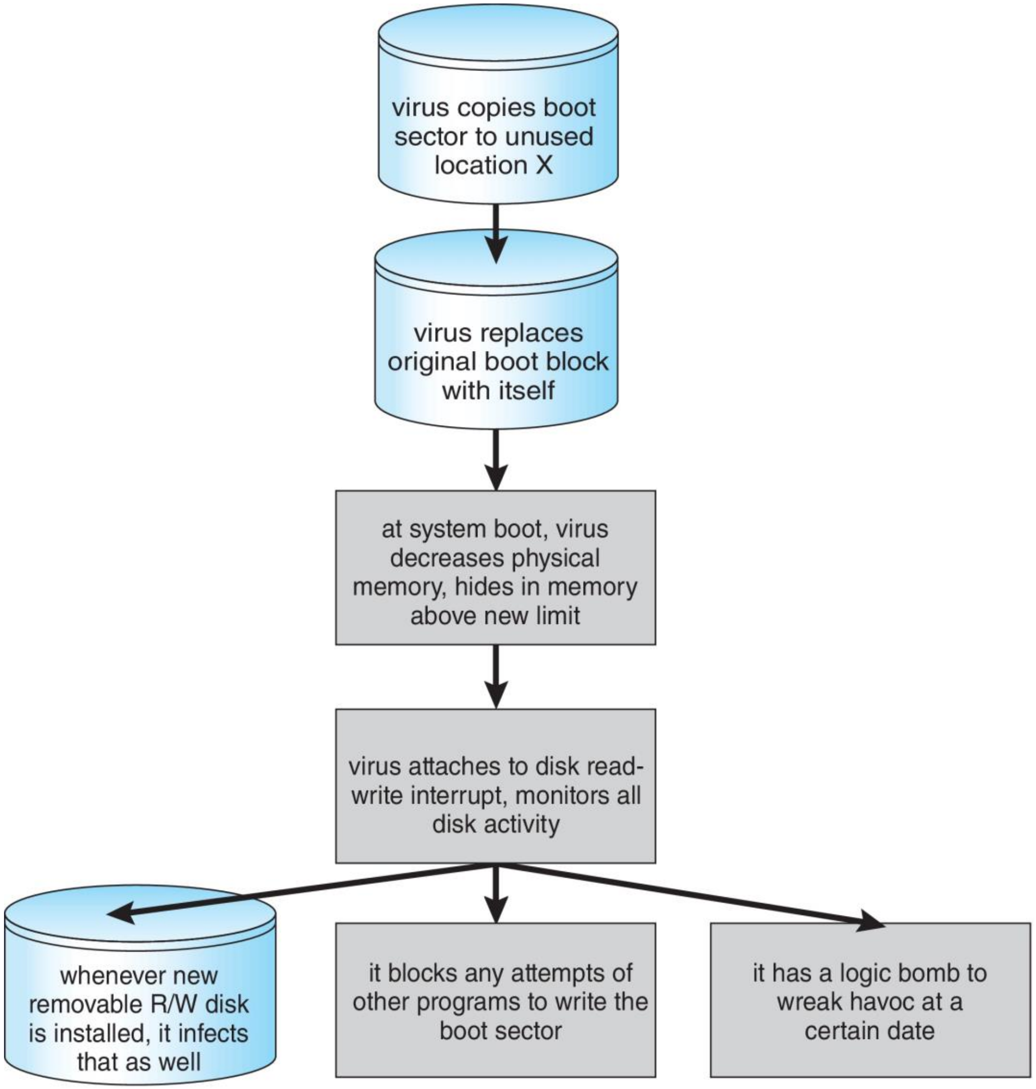
- **Viruses**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
  - Visual Basic Macro to reformat hard drive

```
Sub AutoOpen()
Dim oFS
  Set oFS = CreateObject(''Scripting.FileSystemObject'')
  vs = Shell(''c:command.com /k format c:'',vbHide)
End Sub
```

# Program Threats (Cont.)

- **Virus dropper** inserts virus onto the system
- Many categories of viruses, literally many thousands of viruses
  - File / parasitic
  - Boot / memory
  - Macro
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored

# A Boot-sector Computer Virus



# The Threat Continues

- Attacks still common, still occurring
- Attacks moved over time from science experiments to tools of organized crime
  - Targeting specific companies
  - Creating botnets to use as tool for spam and DDOS delivery
  - **Keystroke logger** to grab passwords, credit card numbers
- Why is Windows the target for most attacks?
  - Most common
  - Everyone is an administrator
    - Licensing required?
  - **Monoculture** considered harmful

# System and Network Threats

- Some systems "open" rather than **secure by default**
  - Reduce **attack surface**
  - But harder to use, more knowledge needed to administer
- Network threats harder to detect, prevent
  - Protection systems weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once system attached to internet
    - Or on network with system attached to internet
  - Even determining location of connecting system difficult
    - IP address is only knowledge

# System and Network Threats (Cont.)

- **Worms** – use **spawn** mechanism; standalone program
- Internet worm
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
  - Exploited trust-relationship mechanism used by *rsh* to access friendly systems without use of password
  - **Grappling hook** program uploaded main worm program
    - 99 lines of C code
  - Hooked system then uploaded main code, tried to attack connected systems
  - Also tried to break into other users accounts on local system via password guessing
  - If target system already infected, abort, except for every 7th time

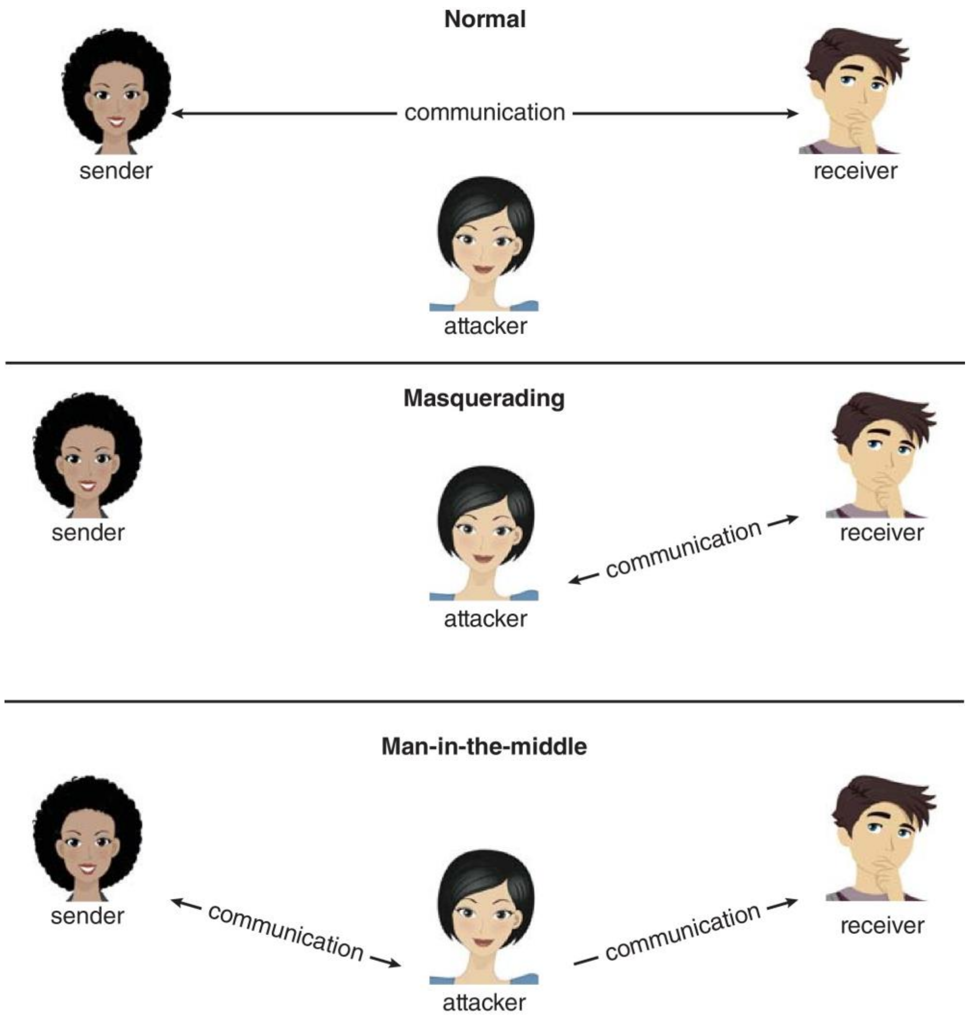
# System and Network Threats (Cont.)

- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Detection of answering service protocol
  - Detection of OS and version running on system
  - `nmap` scans all ports in a given IP range for a response
  - `nessus` has a database of protocols and bugs (and exploits) to apply against a system
  - Frequently launched from **zombie systems**
    - To decrease trace-ability

# System and Network Threats (Cont.)

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed Denial-of-Service** (**DDoS**) come from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
    - How many started-connections can the OS handle?
  - Consider traffic to a web site
    - How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment
- Port scanning
  - Automated tool to look for network ports accepting connections
  - Used for good and evil

# Standard Security Attacks



# Cryptography as a Security Tool

- Broadest security tool available
  - Internal to a given computer, source and destination of messages can be known and protected
    - OS creates, manages, protects process IDs, communication ports
  - Source and destination of messages on network cannot be trusted without cryptography
    - Local network – IP address?
      - Consider unauthorized host added
    - WAN / Internet – how to establish authenticity
      - Not via IP address

# Cryptography

- Means to constrain potential senders (*sources*) and / or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver

# Encryption

- Constrains the set of possible receivers of a message
- **Encryption** algorithm consists of
  - Set $K$ of keys
  - Set $M$ of Messages
  - Set $C$ of ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, $E_k$ is a function for generating ciphertexts from messages
    - Both $E$ and $E_k$ for any $k$ should be efficiently computable functions
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, $D_k$ is a function for generating messages from ciphertexts
    - Both $D$ and $D_k$ for any $k$ should be efficiently computable functions

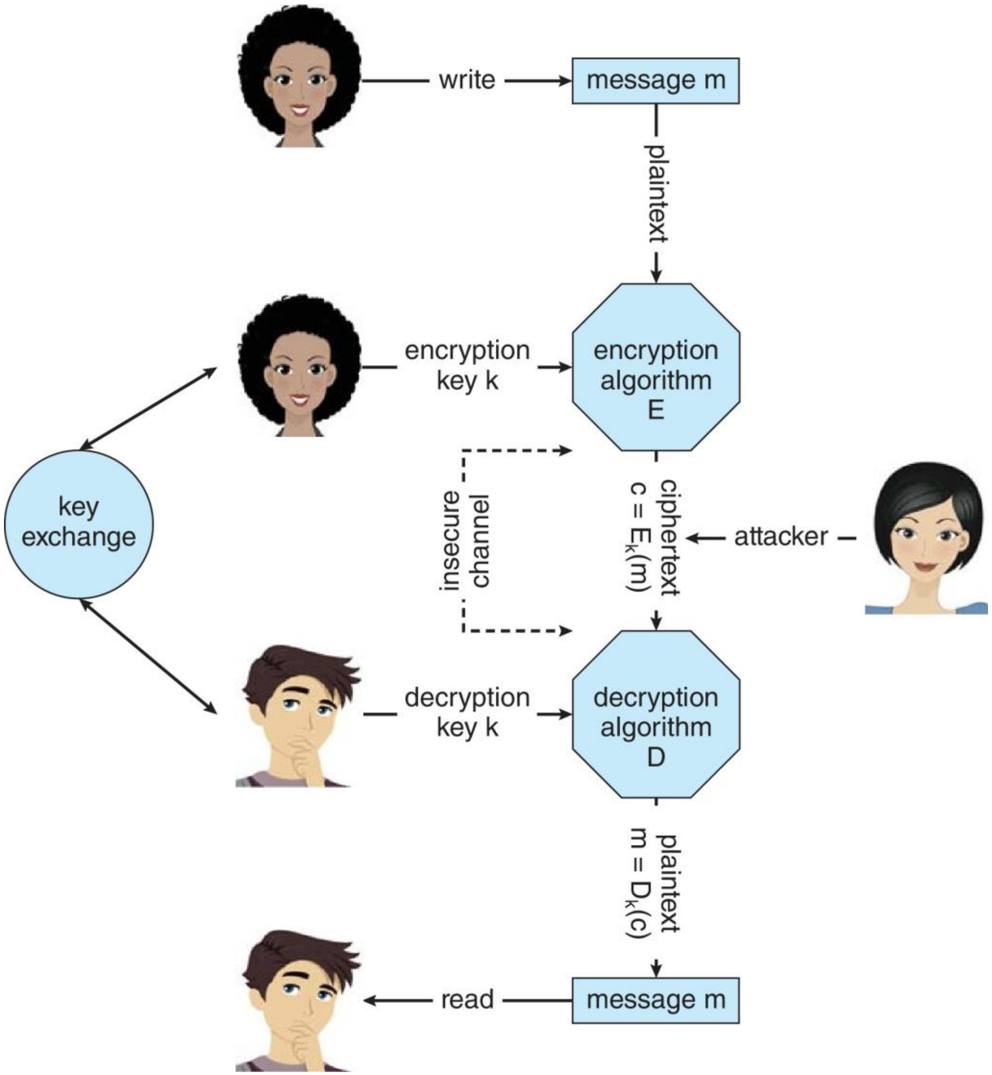
# Encryption (Cont.)

- An encryption algorithm must provide this essential property: Given a ciphertext $c \in C$, a computer can compute m such that $E_k(m) = c$ only if it possesses *k*
  - Thus, a computer holding *k* can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding *k* cannot decrypt ciphertexts
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive *k* from the ciphertexts

# Symmetric Encryption

- Same key used to encrypt and decrypt
  - Therefore *k* must be kept secret
- DES was most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
  - Keys too short so now considered insecure
- Triple-DES considered more secure
  - Algorithm used 3 times using 2 or 3 keys
  - For example $c = E_{k3}(D_{k2}(E_{k1}(m)))$
- 2001 NIST adopted new block cipher - Advanced Encryption Standard (**AES**)
  - Keys of 128, 192, or 256 bits, works on 128 bit blocks
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is a input to pseudo-random-bit generator
    - Generates an infinite **keystream**

# Secure Communication over Insecure Medium



# Asymmetric Encryption

- **Public-key encryption** based on each user having two keys:
  - **public key** – published key used to encrypt data
  - **private key** – key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is **RSA** block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number

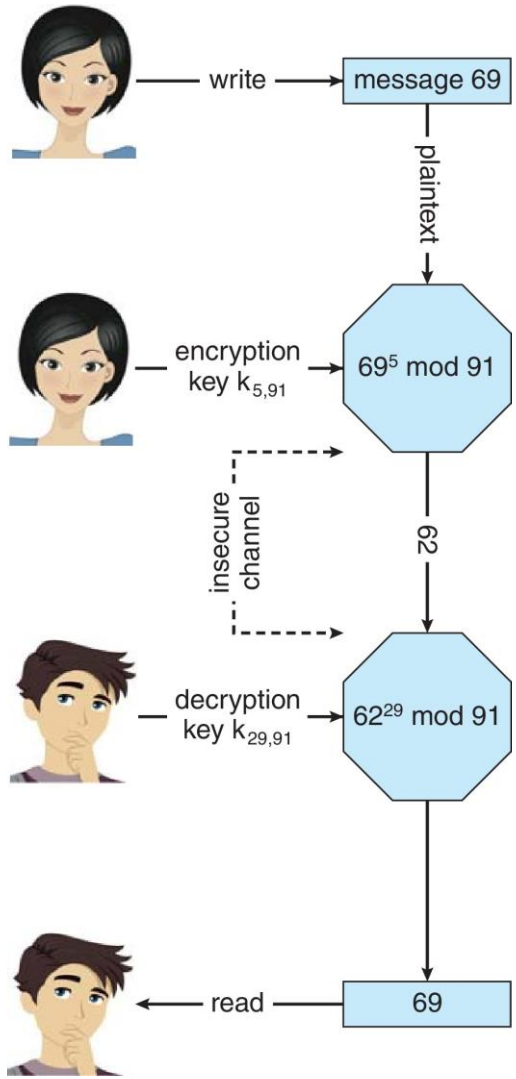
# Asymmetric Encryption (Cont.)

- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so $k_e$ need not be kept secret and can be widely disseminated
  - $k_e$ is the **public key**
  - $k_d$ is the **private key**
  - $N$ is the product of two large, randomly chosen prime numbers $p$ and $q$ (for example, $p$ and $q$ are 512 bits each)
  - Encryption algorithm is $E_{ke,N}(m) = m^{k_e} \bmod N$, where $k_e$ satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
  - The decryption algorithm is then $D_{kd,N}(c) = c^{k_d} \bmod N$

# Asymmetric Encryption Example

- For example. make $p$ = 7and $q$ = 13
- We then calculate $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select $k_e$ relatively prime to 72 and< 72, yielding 5
- Finally, we calculate $k_d$ such that $k_e k_d$ mod 72 = 1, yielding 29
- We how have our keys
  - Public key, $k_{e,N}$ = 5, 91
  - Private key, $k_{d,N}$ = 29, 91
- Encrypting the message 69 with the public key results in the cyphertext 62
- Cyphertext can be decoded with the private key
  - Public key can be distributed in cleartext to anyone who wants to communicate with holder of public key

# Encryption using RSA Asymmetric Cryptography



# Cryptography (Cont.)

- Note symmetric cryptography based on transformations, asymmetric based on mathematical functions
  - Asymmetric much more compute intensive
  - Typically not used for bulk data encryption

# Authentication

- Constraining set of potential senders of a message
  - Complementary to encryption
  - Also can prove message unmodified
- Algorithm components
  - A set $K$ of keys
  - A set $M$ of messages
  - A set $A$ of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, $S_k$ is a function for generating authenticators from messages
    - Both $S$ and $S_k$ for any $k$ should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$. That is, for each $k \in K$, $V_k$ is a function for verifying authenticators on messages
    - Both $V$ and $V_k$ for any $k$ should be efficiently computable functions

# Authentication (Cont.)

- For a message *m*, a computer can generate an authenticator $a \in A$ such that $V_k(m, a)$ = `true` only if it possesses *k*
- Thus, computer holding *k* can generate authenticators on messages so that any other computer possessing *k* can verify them
- Computer not holding *k* cannot generate authenticators on messages that can be verified using $V_k$
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive *k* from the authenticators
- Practically, if $V_k(m,a)$ = **`true`** then we know *m* has not been modified and that send of message has *k*
  - If we share *k* with only one entity, know where the message originated

# Authentication – Hash Functions

- Basis of authentication
- Creates small, fixed-size block of data **message digest** (**hash value**) from *m*
- Hash Function *H* must be collision resistant on *m*
  - Must be infeasible to find an $m' \neq m$ such that $H(m) = H(m')$
- If $H(m) = H(m')$, then $m = m'$
  - The message has not been modified
- Common message-digest functions include **MD5**, which produces a 128-bit hash, and **SHA-1**, which outputs a 160-bit hash
- Not useful as authenticators
  - For example *H(m)* can be sent with a message
    - But if *H* is known someone could modify *m* to *m'* and recompute *H(m')* and modification not detected
    - So must authenticate *H(m)*

# Authentication - MAC

- Symmetric encryption used in **message-authentication code** (**MAC**) authentication algorithm
- Cryptographic checksum generated from message using secret key
  - Can securely authenticate short values
- If used to authenticate *H(m)* for an *H* that is collision resistant, then obtain a way to securely authenticate long message by hashing them first
- Note that *k* is needed to compute both $S_k$ and $V_k$, so anyone able to compute one can compute the other

# Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- Very useful – ***anyone*** can verify authenticity of a message
- In a digital-signature algorithm, computationally infeasible to derive $k_s$ from $k_v$
  - $V$ is a one-way function
  - Thus, $k_v$ is the public key and $k_s$ is the private key
- Consider the RSA digital-signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message $S_{ks}(m) = H(m)^{k_s} \bmod N$
  - The key $k_s$ again is a pair $(d, N)$, where $N$ is the product of two large, randomly chosen prime numbers $p$ and $q$
  - Verification algorithm is $V_{kv}(m, a)$ $(a^{k_v} \bmod N = H(m))$
    - Where $k_v$ satisfies $k_v k_s \bmod (p - 1)(q - 1) = 1$

# Authentication (Cont.)

- Why authentication if a subset of encryption?
  - Fewer computations (except for RSA digital signatures)
  - Authenticator usually shorter than message
  - Sometimes want authentication but not confidentiality
    - Signed patches et al
  - Can be basis for **non-repudiation**

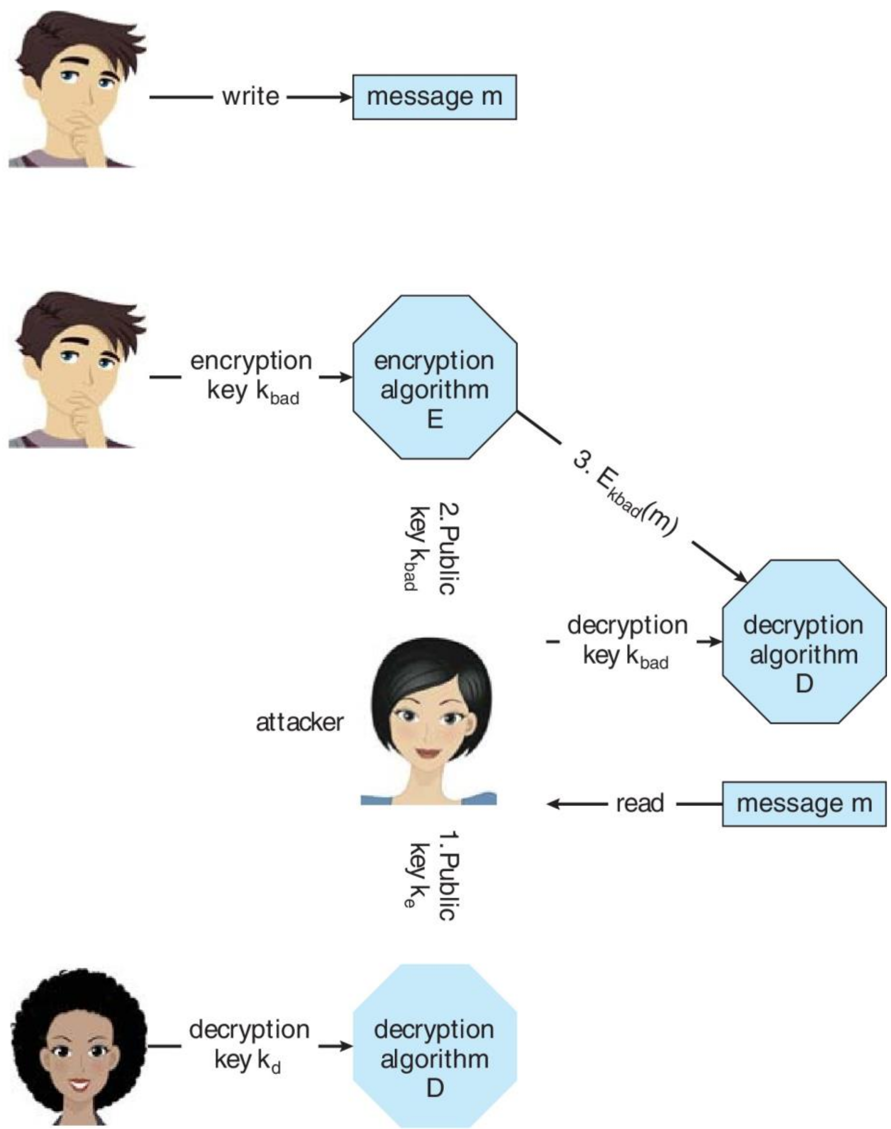
# Key Distribution

- Delivery of symmetric key is huge challenge
  - Sometimes done **out-of-band**
- Asymmetric keys can proliferate – stored on **key ring**
  - Even asymmetric key distribution needs care – man-in-the-middle attack

# Digital Certificates

- Proof of who or what owns a public key
- Public key digitally signed a trusted party
- Trusted party receives proof of identification from entity and certifies that public key belongs to entity
- **Certificate authority** are trusted party – their public keys included with web browser distributions
  - They vouch for other authorities via digitally signing their keys, and so on

# Man-in-the-middle Attack on Asymmetric Cryptography



# User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through **passwords**, can be considered a special case of either keys or capabilities
- Passwords must be kept secret
  - Frequent change of passwords
  - History to avoid repeats
  - Use of "non-guessable" passwords
  - Log all invalid access attempts (but not the passwords themselves)
  - Unauthorized transfer
- Passwords may also either be encrypted or allowed to be used only once
  - Does encrypting passwords solve the exposure problem?
    - Might solve **sniffing**
    - Consider **shoulder surfing**
    - Consider Trojan horse keystroke logger
    - How are passwords stored at authenticating site?

# Passwords

- Encrypt to avoid having to keep secret
  - But keep secret anyway (i.e. Unix uses superuser-only readably file `/etc/shadow`)
  - Use algorithm easy to compute but difficult to invert
  - Only encrypted password stored, never decrypted
  - Add "salt" to avoid the same password being encrypted to the same value
- One-time passwords
  - Use a function based on a seed to compute a password, both user and computer
  - Hardware device / calculator / key fob to generate the password
    - Changes very frequently
- Biometrics
  - Some physical attribute (fingerprint, hand scan)
- Multi-factor authentication
  - Need two or more factors for authentication
    - i.e., USB "dongle", biometric measure, and password

# Passwords (Cont.)

### *STRONG AND EASY TO REMEMBER PASSWORDS*

It is extremely important to use strong (hard to guess and hard to shoulder surf) passwords on critical systems like bank accounts. It is also important to not use the same password on lots of systems, as one less important, easily hacked system could reveal the password you use on more important systems. A good technique is to generate your password by using the first letter of each word of an easily remembered phrase using both upper and lower characters with a number or punctuation mark thrown in for good measure. For example, the phrase "My girlfriend's name is Katherine" might yield the password "Mgn.isK!". The password is hard to crack but easy for the user to remember. A more secure system would allow more characters in its passwords. Indeed, a system might also allow passwords to include the space character, so that a user could create a **passphrase** which is easy to remember but difficult to break.

# Security Defenses Summarized

- By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers. In summary, these layers may include the following:
  - Educate users about safe computing—don't attach devices of unknown origin to the computer, don't share passwords, use strong passwords, avoid falling for social engineering appeals, realize that an e-mail is not necessarily a private communication, and so on
  - Educate users about how to prevent phishing attacks—don't click on email attachments or links from unknown (or even known) senders; authenticate (for example, via a phone call) that a request is legitimate
  - Use secure communication when possible
  - Physically protect computer hardware
  - Configure the operating system to minimize the attack surface; disable all unused services
  - Configure system daemons, privileges applications, and services to be as secure as possible

# Security Defenses Summarized (Cont.)

- Use modern hardware and software, as they are likely to have up-to-date security features
- Keep systems and applications up to date and patched
- Only run applications from trusted sources (such as those that are code signed)
- Enable logging and auditing; review the logs periodically, or automate alerts
- Install and use antivirus software on systems susceptible to viruses, and keep the software up to date
- Use strong passwords and passphrases, and don’t record them where they could be found
- Use intrusion detection, firewalling, and other network-based protection systems as appropriate
- For important facilities, use periodic vulnerability assessments and other testing methods to test security and response to incidents

# Security Defenses Summarized (Cont.)

- Encrypt mass-storage devices, and consider encrypting important individual files as well
- Have a security policy for important systems and facilities, and keep it up to date

# End of Chapter 16