Network Security

Goals:

- understand principles of network security:
  - cryptography and its *many* uses beyond “confidentiality”
  - authentication
  - message integrity

- security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers
## Motivation

<table>
<thead>
<tr>
<th>Adversary</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>To have fun snooping on people’s email</td>
</tr>
<tr>
<td>Cracker</td>
<td>To test out someone’s security system; steal data</td>
</tr>
<tr>
<td>Sales rep</td>
<td>To claim to represent all of Europe, not just Andorra</td>
</tr>
<tr>
<td>Businessman</td>
<td>To discover a competitor’s strategic marketing plan</td>
</tr>
<tr>
<td>Ex-employee</td>
<td>To get revenge for being fired</td>
</tr>
<tr>
<td>Accountant</td>
<td>To embezzle money from a company</td>
</tr>
<tr>
<td>Stockbroker</td>
<td>To deny a promise made to a customer by email</td>
</tr>
<tr>
<td>Con man</td>
<td>To steal credit card numbers for sale</td>
</tr>
<tr>
<td>Spy</td>
<td>To learn an enemy’s military or industrial secrets</td>
</tr>
<tr>
<td>Terrorist</td>
<td>To steal germ warfare secrets</td>
</tr>
</tbody>
</table>
What is network security?

- **confidentiality:** only sender, intended receiver should “understand” message contents
  - **Method** — encrypt at sender, decrypt at receiver
  - A protocol that prevents an adversary from understanding the message contents is said to provide **confidentiality**.
  - Concealing the quantity or destination of communication is called **traffic confidentiality**.

- **message integrity:** sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
  - A protocol that detects message tampering provides **data integrity**.
  - The adversary could alternatively transmit an extra copy of your message in a **replay attack**.
  - A protocol that detects message tampering provides **originality**.
  - A protocol that detects delaying tactics provides **timeliness**.
What is network security?

- **authentication**: sender, receiver want to confirm identity of each other
  - A protocol that ensures that you really are talking to whom you think you’re talking is said to provide **authentication**.
  - Example: DNS Attack [correct URL gets converted to malicious IP]

- **access and availability**: services must be accessible and available to users
  - A protocol that ensures a degree of access is called **availability**.
  - Denial of Service (DoS) Attack
  - Example: SYN Flood attack (Client not transmitting 3rd message in TCP 3-way handshake, thus consuming server’s resource)
  - Example: Ping Flood (attacker transmits ICMP Echo Request packets)
Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages
Who might Bob, Alice be?

- ... well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., online purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?
There are bad guys (and girls) out there!

**Q:** What can a “bad guy” do?

**A:** A lot!

- **eavesdrop:** intercept messages
- actively **insert** messages into connection
- **impersonation:** can fake (spoof) source address in packet (or any field in packet)
- **hijacking:** “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service:** prevent service from being used by others (e.g., by overloading resources)
Cryptography in Insecure Network
The language of cryptography

plaintext message $m$  \rightarrow$K_A$ encryption algorithm \rightarrow ciphertext $K_A(m)$ \rightarrow decryption algorithm \rightarrow plaintext $m = K_B(K_A(m))$

Alice’s encryption key $K_A$  \rightarrow $Bob’s decryption key $K_B$
Symmetric key cryptography

Symmetric key crypto: Bob and Alice share same (symmetric) key: $K_S$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?
**Simple encryption scheme**

**substitution cipher:** substituting one thing for another

- **monoalphabetic** cipher: substitute one letter for another

\[
\begin{align*}
\text{plaintext: } & \quad \text{abcdefghijklmnopqrstuvwxyz} \\
\downarrow & \quad \downarrow \\
\text{ciphertext: } & \quad \text{mnbvcxzasdfghjklpoiuytrewq}
\end{align*}
\]

**Example:**

- Plaintext: bob. i love you. alice
  - Ciphertext: nkn. s gktc wky. mgsbc

**Encryption key:** mapping from set of 26 letters to set of 26 letters
Breaking an encryption scheme

- cipher-text only attack: Trudy has ciphertext she can analyze
  - two approaches:
    - brute force: search through all keys
    - statistical analysis
- known-plaintext attack: Trudy has plaintext corresponding to ciphertext [when an intruder knows some of the (plain, cipher) pairings]
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext
  - If Trudy could get Alice to send encrypted message, “The quick brown fox jumps over the lazy dog”, then the encryption is broken.
## Polyalphabetic Cipher

- **n** substitution ciphers, $C_1, C_2, \ldots, C_n$
- **cycling pattern:**
  - e.g., $n=4$ [$C_1-C_4$], $k$=key length=5:
  - $C_1, C_3, C_4, C_3, C_2$; $C_1, C_3, C_4, C_3, C_2$; 
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - **dog:** d from $C_1$, o from $C_3$, g from $C_4$

### Encryption key:
- n substitution ciphers, and cyclic pattern
- key need not be just n-bit pattern

---

<table>
<thead>
<tr>
<th>Plaintext letter:</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1(k=5)$:</td>
<td>f g h i j k l m n o p q r s t u v w x y z a b c d e</td>
</tr>
<tr>
<td>$C_2(k=19)$:</td>
<td>t u v w x y z a b c d e f g h i j k l m n o p q r s</td>
</tr>
</tbody>
</table>

---

*Figures and images are not included in the plain text representation.*
Block vs Stream Cipher

- **Block ciphers** process messages into blocks, each of which is then en/decrypted
  - 64-bits or more
  - Example: DES, AES

- **Stream ciphers** process messages a bit or byte at a time when en/decrypting
  - Example: WEP (used in 802.11)

- Brute Force attack is possible if few number of bits are chosen
Cipher Block Chaining

- Plaintext block is XORed with the previous block’s ciphertext before being encrypted.
  - Each block’s ciphertext depends on the preceding blocks
  - First plaintext block is XORed with a random number.
    ✓ That random number, called an *initialization vector (IV)*, is included with the series of ciphertext blocks so that the first ciphertext block can be decrypted.

- Provides better efficiency for brute force attack
Symmetric key crypto: DES

**DES: Data Encryption Standard**
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - **DES Challenge**: 56-bit-key-encrypted phrase, decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - **3DES**: encrypt 3 times with 3 different keys
Symmetric key crypto: DES

**DES operation**

- initial permutation (on 64 bits)
- 16 identical “rounds” of function application
  - each using different 48 bits of key
  - rightmost 32 bits are moved to leftmost 32 bits
- final permutation (on 64 bits)

Kaufman, Schneier, 1995
AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES
Public Key Cryptography

**symmetric key crypto**

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

**public key crypto**

- radically different approach
  [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver
Public key cryptography

plaintext message, \( m \)

\[ c = K_B^+(m) \]

decryption algorithm

\[ m = K_B^-(K_B^+(m)) \]

Bob’s public key

Bob’s private key

encryption algorithm

ciphertext

plaintext message

\[ c = K_B^+(m) \]
Public key encryption algorithms

requirements:

1. need \( K_B^+ (\cdot) \) and \( K_B^- (\cdot) \) such that
   \[
   K_B^- (K_B^+ (m)) = m
   \]

2. given public key \( K_B^+ \), it should be impossible to compute private key \( K_B^- \)

**RSA**: Rivest, Shamir, Adelson algorithm [1999]
Prerequisite: modular arithmetic

- $x \mod n = \text{remainder of } x \text{ when divide by } n$
- facts:
  
  
  $[(a \mod n) + (b \mod n)] \mod n = (a+b) \mod n$
  $[(a \mod n) - (b \mod n)] \mod n = (a-b) \mod n$
  $[(a \mod n) \times (b \mod n)] \mod n = (a*b) \mod n$

- thus

  $(a \mod n)^d \mod n = a^d \mod n$

- example: $x=14$, $n=10$, $d=2$:

  $(x \mod n)^d \mod n = 4^2 \mod 10 = 6$
  $x^d = 14^2 = 196 \mod 10 = 6$
RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number

**example:**

- \( m = 10010001 \). This message is uniquely represented by the decimal number 145.
- to encrypt \( m \), we encrypt the corresponding number, which gives a new number (the ciphertext).
RSA: Creating public/private key pair

1. choose two large prime numbers \( p, q \).
   (e.g., 1024 bits each)

2. compute \( n = pq, \quad z = (p-1)(q-1) \)

3. choose \( e \) (with \( e < n \)) that has no common factors with \( z \) (\( e, z \) are “relatively prime”).

4. choose \( d \) such that \( ed - 1 \) is exactly divisible by \( z \).
   (in other words: \( ed \mod z = 1 \)).

5. public key is \((n, e)\). private key is \((n, d)\).
RSA: encryption, decryption

0. given $(n,e)$ and $(n,d)$ as computed above

1. to encrypt message $m (<n)$, compute
   \[ c = m^e \mod n \]

2. to decrypt received bit pattern, $c$, compute
   \[ m = c^d \mod n \]

\[ m = \underbrace{(m^e \mod n)}_{\text{magic happens!}} \quad d \mod n \]
RSA example:


- $e=5$ (so $e, z$ relatively prime).
- $d=29$ (so $ed-1$ exactly divisible by $z$).

encrypting 8-bit messages.

\[
\begin{align*}
\text{encrypt:} & \quad \text{bit pattern} & \quad m & \quad m^e & \quad c = m^e \mod n \\
00001000 & \quad 12 & \quad 24832 & \quad 17
\end{align*}
\]

\[
\begin{align*}
\text{decrypt:} & \quad c & \quad c^d & \quad m = c^d \mod n \\
17 & \quad 481968572106750915091411825223071697 & \quad 12
\end{align*}
\]
Why does RSA work?

- must show that $c^d \mod n = m$
  where $c = m^e \mod n$
- fact: for any $x$ and $y$: $x^y \mod n = x^{(y \mod z)} \mod n$
  - where $n = pq$ and $z = (p-1)(q-1)$
- thus,
  
  $c^d \mod n = (m^e \mod n)^d \mod n$

  $= m^{ed} \mod n$

  $= m^{(ed \mod z)} \mod n$

  $= m^1 \mod n$

  $= m$
RSA: another important property

The following property will be very useful later:

\[ \mathcal{K}_B^-(\mathcal{K}_B^+(m)) = m = \mathcal{K}_B^+(\mathcal{K}_B^-(m)) \]

use public key first, followed by private key

use private key first, followed by public key

result is the same!
How is it possible?

follows directly from modular arithmetic:

\[(m^e \mod n)^d \mod n = m^{ed} \mod n\]
\[= m^{de} \mod n\]
\[= (m^d \mod n)^e \mod n\]
Why is RSA secure?

- suppose you know Bob’s public key (n,e). How hard is it to determine d?
- essentially need to find factors of n without knowing the two factors p and q
  - fact: factoring a big number is hard
RSA in practice: session keys

- Exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

**Session key, \( K_S \)**

- Bob and Alice use RSA to exchange a symmetric key \( K_S \)
- Once both have \( K_S \), they use symmetric key cryptography
Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”

Failure scenario??
Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”

in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice
Authentication: another try

*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address

<table>
<thead>
<tr>
<th>Alice’s IP address</th>
<th>“I am Alice”</th>
</tr>
</thead>
</table>

Failure scenario??
Authentication: another try

*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address.

Trudy can create a packet “spoofing” Alice’s address.
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.

Failure scenario??
**Authentication: another try**

*Protocol ap3.0:* Alice says “I am Alice” and sends her secret password to “prove” it.

*Playback attack:* Trudy records Alice’s packet and later plays it back to Bob.
**Authentication: yet another try**

**Protocol ap3.1:** Alice says “I am Alice” and sends her **encrypted** secret password to “prove” it.

Failure scenario??

Alice’s IP addr | encrypted password | “I’m Alice”
--- | --- | ---
Alice’s IP addr | OK
Protocol ap3.1: Alice says “I am Alice” and sends her encrypted secret password to “prove” it. 

record and playback still works!
**Goal:** avoid playback attack

**nonce:** number (R) used only *once-in-a-lifetime*

**ap4.0:** to prove Alice “live”, Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?
ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

**ap5.0:** use nonce, public key cryptography

---

**Authentication: ap5.0**

- "I am Alice"
- "send me your public key"

Bob computes

$$K_A^-(K_A^+(R)) = R$$

and knows only Alice could have the private key, that encrypted R such that

$$K_A^-(K_A^+(R)) = R$$
**man (or woman) in the middle attack:** Trudy poses as Alice (to Bob) and as Bob (to Alice)

I am Alice

Send me your public key

I am Alice

Send me your public key

Trudy gets

\[ m = K_A^{-1}(K_T^+(m)) \]

sends m to Alice encrypted with Alice’s public key
**ap5.0: security hole**

*man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)

**difficult to detect:**
- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!
Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
simple digital signature for message m:

- Bob signs m by encrypting with his private key $K_B^-$, creating “signed” message, $K_B^-(m)$

Dear Alice

Oh, how I have missed you. I think of you all the time! ...

(blah blah blah)

Bob
Digital signatures

- suppose Alice receives msg m, with signature: m, $K_B(m)$
- Alice verifies m signed by Bob by applying Bob’s public key $K_B$ to $K_B(m)$
  then checks $K_B(K_B(m)) = m$.
- If $K_B(K_B(m)) = m$, whoever signed m must have used Bob’s private key.

Alice thus verifies that:
- Bob signed m
- no one else signed m
- Bob signed m and not m’

non-repudiation:
- Alice can take m, and signature $K_B(m)$ to court and prove that Bob signed m
Firewalls

**firewall**

isolates organization’s internal net from larger Internet, allowing some packets to pass, blocking others

trusted “good guys”

untrusted “bad guys”

administered network

public Internet

firewall
Firewalls: why do we need it?

prevent denial of service attacks:
- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data
- e.g., attacker replaces CIA’s homepage with something else

allow only authorized access to inside network
- set of authenticated users/hosts

three types of firewalls:
- stateless packet filters
- stateful packet filters
- application gateways
Stateless packet filtering

- Internal network connected to Internet via router firewall
- Router filters packet-by-packet, decision to forward/drop packet based on:
  - Source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits
Stateless packet filtering: example

- **example 1:** block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - **result:** all incoming, outgoing UDP flows and telnet connections are blocked

- **example 2:** block inbound TCP segments with ACK=0.
  - **result:** prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.
<table>
<thead>
<tr>
<th><strong>Policy</strong></th>
<th><strong>Firewall Setting</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>No outside Web access.</td>
<td>Drop all outgoing packets to any IP address, port 80</td>
</tr>
<tr>
<td>No incoming TCP connections, except those for institution’s public Web server only.</td>
<td>Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80</td>
</tr>
<tr>
<td>Prevent Web-radios from eating up the available bandwidth.</td>
<td>Drop all incoming UDP packets - except DNS and router broadcasts.</td>
</tr>
<tr>
<td>Prevent your network from being used for a smurf DoS attack.</td>
<td>Drop all ICMP packets going to a “broadcast” address (e.g. 130.207.255.255).</td>
</tr>
<tr>
<td>Prevent your network from being tracerouted</td>
<td>Drop all outgoing ICMP TTL expired traffic</td>
</tr>
</tbody>
</table>
**Access Control Lists**

**ACL:** table of rules, applied top to bottom to incoming packets: (action, condition) pairs

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>protocol</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>TCP</td>
<td>&gt; 1023</td>
<td>80</td>
<td>any</td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
</tr>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>UDP</td>
<td>&gt; 1023</td>
<td>53</td>
<td>---</td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>UDP</td>
<td>53</td>
<td>&gt; 1023</td>
<td>----</td>
</tr>
<tr>
<td>deny</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
</tr>
</tbody>
</table>
Stateful packet filtering

- **stateless packet filter**: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>protocol</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
</tr>
</tbody>
</table>

- **stateful packet filter**: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets
Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>proto</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
<th>check conxion</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>TCP</td>
<td>&gt; 1023</td>
<td>80</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
<td>X</td>
</tr>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>UDP</td>
<td>&gt; 1023</td>
<td>53</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>UDP</td>
<td>53</td>
<td>&gt; 1023</td>
<td>----</td>
<td>X</td>
</tr>
<tr>
<td>deny</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td></td>
</tr>
</tbody>
</table>
Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.
Limitations of firewalls, gateways

- **IP spoofing**: router can’t know if data “really” comes from claimed source
- If multiple app’s. need special treatment, each has own app. gateway
- Client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- Filters often use all or nothing policy for UDP
- **Tradeoff**: degree of communication with outside world, level of security
- Many highly protected sites still suffer from attacks