Chapter 3: Principles of Network Security

Chapter goals:

- understand principles of network security:
  - cryptography and its many uses beyond "confidentiality"
  - authentication
  - message integrity
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication

Network Security

☒ The field of network security is about:
   ☒ how bad guys can attack computer networks
   ☒ how we can defend networks against attacks
   ☒ how to design architectures that are immune to attacks

☒ Internet not originally designed with (much) security in mind
   ☒ original vision: “a group of mutually trusting users attached to a transparent network” 😊
   ☒ Internet protocol designers playing “catch-up”
   ☒ Security considerations in all layers!
Bad guys can put malware into hosts via Internet

- Malware can get in host from a virus, worm, or Trojan horse
- Spyware malware can record keystrokes, web sites visited, upload info to collection site
- Infected host can be enrolled in a botnet, used for spam and DDoS attacks
- Malware is often self-replicating: from an infected host, seeks entry into other hosts

Digital pest

- Trojan Horse
  - Instructions, hidden inside an otherwise useful program, that do bad things. Usually this refers to malicious instructions installed at the time the program is written (≠virus)

- Virus
  - A set of instructions that, when executed, insert copies of itself into other programs (self-replication)
  - In particular, instructions in email messages that, when executed, cause malicious code to be sent in email to other users

- Worm
  - A program that replicates itself by installing copies of itself on other machines across a network
Digital pest

- **Trapdoor**
  - An undocumented entry point intentionally written into a program (often for debugging purposes), which can be exploited as a security flaw.

- **Logic bomb**
  - Malicious instructions that trigger on some event in the future, such as a particular time occurring.

- **Zombie**
  - Malicious instructions installed on a system that can be remotely triggered to carry out some attack with less traceability because the attack comes from another victim.
  - Often a large number of zombies are installed.

More on digital pest

- Is it possible to detect a digital pest in a program?
  - One of the famous results in computer science is that it is impossible to be able to tell what an arbitrary program will do by looking at it!
  - In fact, it is impossible in general to discern any nontrivial property of a program by looking at it (e.g., if the program will halt).
  - Anyway, nobody looks!

- A virus can be installed in any program as follows:
  - Replace any instruction, say the instruction at location x, by a jump to some free space in memory, say location y; then
  - Write the virus program starting at location y; then
  - Place the instruction that was originally at location x at the end of the virus program, followed by a jump to x+1

- Replication
  - Besides the delayed planned damage, the virus replicates itself silently.
  - If it did not wait before damaging the infected system, it would not spread as far!
Do you always realize you’re running a program?

- Modern Email clients often process the attachments
  - Which may be infected by a virus

- PostScript is a complete programming language
  - Displaying a ps file is running a program that could contain a Trojan horse

Covert channels

- A covert channel is a method for a Trojan horse to circumvent the automatic confinement of information within a security perimeter
  - Assume the Trojan horse program has not enough privileges to directly send confidential data outside the system

- The timing channel
  - The Trojan horse program alternately loops and waits, in cycles of, say one minute per bit (of the confidential data)
  - When the bit is 1: the program loops for one minute
  - When the bit is 0: the program waits for a minute
  - Another program running on the same computer (but without access to the sensitive data) constantly tests the loading of the system!
  - Also possible from a distant computer, by testing the reaction time of the infected system to some requests (possibly averaged over a minute)
  - Other processes running at the same time are adding noise to the timing channel
  - But communication people can deal with noisy channels!
Covert channels (2)

Variant: the storage channel
- The Trojan horse program loads a (printer) queue to send a 1, and deletes its jobs to send a 0.
- Easy to check the queue status and get the information.

Yet another one: the error channel
- The Trojan horse program creates a file to send a 1, and deletes it to send a 0.
- The external process tries to read the file: if different error messages are reported when the file exists (but its access is not permitted) or not, we have a channel.

Bad guys can attack servers and network infrastructure

Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic.

1. select target
2. break into hosts around the network
3. send packets to target from compromised hosts (botnet)
The bad guys can sniff packets

**Packet sniffing:**
- broadcast media (shared Ethernet, wireless)
- promiscuous network interface reads/records all packets (e.g., including passwords!) passing by

- Wireshark software is a (free) packet-sniffer

![Image of packet sniffing]

The bad guys can use false source addresses

- **IP spoofing:** send packet with false source address

![Image of IP spoofing]

Impersonation, masquerading

Allows **Hijacking:** “taking over” ongoing connection by removing sender or receiver, inserting himself in place
The bad guys can record and playback

- **record-and-playback**: sniff sensitive info (e.g., password), and use later
  - password holder is that user from system point of view

![Network Diagram]

**Security services**

- **Confidentiality**: only sender, intended receiver should "understand" message contents
  - sender encrypts message
  - receiver decrypts message
- **Authentication**: sender, receiver want to confirm identity of each other
- **Message integrity**: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
- **Nonrepudiation**: neither the sender nor the receiver of a message should be able to deny the transmission (nonrepudiation with proof of origin/delivery)
- **Access control**: access to information resources may be controlled by or for the target system
- **Availability**: services must be accessible and available to users when needed
Generic types of attacks

- **Information source** → **Information destination**
  - **Normal flow**
  - **Interception**
    - Attack on confidentiality
    - (could be only traffic analysis)
  - **Interruption**
    - Attack on availability
    - (DoS: Denial of Service)
  - **Modification**
    - Attack on integrity
  - **Fabrication**
    - Attack on authenticity
    - (masquerading, replay)

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice 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., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication
The language of cryptography

plaintext message
\( m \)
ciphertext, encrypted with key \( K_A \)
\( m = K_B(K_A(m)) \)

Simple encryption scheme

substitution cipher: substituting one thing for another
- monalphabetic cipher: substitute one letter for another

plaintext:  abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq

E.g.: Plaintext: bob. i love you. alice

ciphertext: nkn. s gktc wky. mgsbc

Key: the mapping from the set of 26 letters to the set of 26 letters

Q: How hard to break this simple cipher?
- brute force (how hard?)
- other?
Polyalphabetic encryption

- A more sophisticated encryption approach
- \( n \) monoalphabetic ciphers, \( M_1, M_2, ..., M_n \)
- Cycling pattern:
  - E.g., \( n=4 \), \( M_1, M_3, M_4, M_2 \), \( M_1, M_3, M_4, M_3, M_2 \)
- For each new plaintext symbol, use subsequent monoalphabetic pattern in cyclic pattern
  - Dog: d from \( M_1 \), o from \( M_3 \), g from \( M_4 \)
- **Key**: the \( n \) ciphers and the cyclic pattern
  - Key need not be just \( n \)-bit pattern

Breaking an Encryption Scheme

- Three basic attacks of increasing strength
- **Ciphertext Only**
  - Trudy has ciphertext that she can analyse
  - Trudy must recognize when she has succeeded
  - Possible only if there is some redundancy in the plaintext to make it recognizable (e.g. the text is in English, or has some starting keyword)
- **Known Plaintext**
  - Trudy knows some (ciphertext, plaintext) pairs
  - Sometimes easier to break
  - E.g., a monoalphabetic cipher can be broken easily with a small amount of plaintext
- **Chosen Plaintext**
  - Trudy may have the opportunity to ask the system to encrypt some chosen plaintext
  - E.g., “The quick brown fox jumps over the lazy dog” would break a monoalphabetic cipher
  - Ditto if Trudy knows some plaintexts normally encrypted by the system. In this case, she would ask the system to encrypt them to see the resulting ciphertexts
Types of Cryptography

- Crypto often uses keys:
  - Algorithm is known to everyone
  - Only "keys" are secret
- Public key cryptography
  - Involves the use of two keys
- Symmetric key cryptography
  - Involves the use of one key
- Hash functions
  - Involves the use of no keys
  - Nothing secret: How can this be useful?

Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naïve approaches
   - Secret-key authentication
   - Public-key authentication
Symmetric key cryptography

**Symmetric key crypto:** Bob and Alice share/know same (symmetric) key: $K_{AB}$
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- **Q:** how do Bob and Alice agree on key value?

**Two types of symmetric ciphers**

- **Stream ciphers**
  - encrypt one bit at time
- **Block ciphers**
  - Break plaintext message in equal-size blocks
  - Encrypt each block as a unit
Stream Ciphers

- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- $m(i) = i^{th}$ bit of message
- $ks(i) = i^{th}$ bit of keystream
- $c(i) = i^{th}$ bit of ciphertext
- $c(i) = ks(i) \oplus m(i)$ ($\oplus$ = exclusive or)
- $m(i) = ks(i) \oplus c(i)$

RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Key can be from 1 to 256 bytes
  - Used in WEP for 802.11
  - Frequently used (± 50%) in SSL
  - Extensively analyzed and considered good
    - Until 2013...
    - Now considered as not sufficiently random:
      - Statistical flaws in the keystream generated by the RC4 algorithm, which become apparent in SSL ciphertexts when the same plaintext is repeatedly encrypted at a fixed location across many SSL sessions
Block ciphers

- Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

**Example with k=3:**

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>110</td>
</tr>
<tr>
<td>001</td>
<td>111</td>
</tr>
<tr>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>111</td>
<td>001</td>
</tr>
</tbody>
</table>

What is the ciphertext for 010110001111?

- How many possible mappings are there for k=3?
  - How many 3-bit inputs?
  - How many permutations of the 3-bit inputs?
  - Answer: 40,320; not very many!
- In general, $2^k!$ mappings; huge for k=64
- Problem:
  - Table approach requires table with $2^{64}$ entries, each entry with 64 bits
  - Table too big: instead use function that simulates a randomly permuted table
Block Cipher

- S₁,...,S₈: permutation tables
- 8-bit to 8-bit mapping
- One pass through: one input bit affects eight output bits

What is the key (assuming scrambler known)?

- Block ciphers: DES, 3DES, AES
- Used in SSL, IPsec, ...

Why rounds in prototype?

- If only a single round, then one bit of input affects at most 8 bits of output
- In 2nd round, the 8 affected bits get scattered and input into multiple substitution boxes
- How many rounds?
  - How many times do you need to shuffle cards?
  - Becomes less efficient as n increases
Encrypting a large message

- Why not just break message in 64-bit blocks, encrypt each block separately?
  - If same block of plaintext appears twice, will give same ciphertext

- How about:
  - Generate random 64-bit number \( r(i) \) for each plaintext block \( m(i) \)
  - Calculate \( c(i) = K_S( m(i) \oplus r(i) ) \)
  - Transmit \( c(i), r(i), i=1,2,... \)
  - At receiver: \( m(i) = K_S(c(i)) \oplus r(i) \)
  - Problem: inefficient, need to send \( c(i) \) and \( r(i) \)

Cipher Block Chaining (CBC)

- **CBC generates its own random numbers**
  - Have encryption of current block depend on result of previous block
  - \( c(i) = K_S( m(i) \oplus c(i-1) ) \)
  - \( m(i) = K_S(c(i)) \oplus c(i-1) \)

- **How do we encrypt first block?**
  - Initialization vector (IV): random block = \( c(0) \)
  - IV does not have to be secret

- **Change IV for each message (or session)**
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time
**Cipher Block Chaining (CBC)**

- **cipher block**: if input block repeated, will produce same cipher text:

  - \(m(1) = \text{"HTTP/1.1"} \quad c(1) = \text{"k329aM02"}\)
  
- **cipher block chaining**:
  - XOR \(i^{th}\) input block, \(m(i)\), with previous block of cipher text, \(c(i-1)\)
  - \(c(0)\) transmitted to receiver in clear
  - \(c(0)\) = Initialisation Vector (IV)
  - what happens in "HTTP/1.1" scenario from above?

- Receiver computes \(m(i) = K_S( (m(i) \oplus c(i-1)) \oplus c(i-1) )\)

---

**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

"keyed" function replaces tables

DES operation
- initial permutation
- 16 identical "rounds" of function application, each using different 48 bits of key
- final permutation

AES: Advanced Encryption Standard
- "new" (Nov. 2001) symmetric-key NIST standard, replacing DES
- 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
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
     - Establishing a shared key
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naïve approaches
   - Secret-key authentication
   - Public-key authentication

Prerequisite: modular arithmetic

- \( x \mod n = \) remainder of \( x \) when divide by \( n \)
- Facts:
  - \( [(a \mod n) + (b \mod n)] \mod n = (a+b) \mod n \)
  - \( [(a \mod n) - (b \mod n)] \mod n = (a-b) \mod n \)
  - \( [(a \mod n) \times (b \mod n)] \mod n = (a\times b) \mod n \)
- Thus
  - \( (a \mod n)^d \mod n = a^d \mod n \)
- Example: \( x=14, n=10, d=2: \)
  - \( (x \mod n)^d \mod n = 4^2 \mod 10 = 6 \)
  - \( x^d = 14^2 = 196 \mod 10 = 6 \)
Establishing a shared key: Diffie-Hellman Key Exchange (1976)

Two strangers can establish a shared key in broad daylight, even with an intruder carefully recording every message.

Private Value, \( x_A \)
Public Value, \( y_A \)

\[ y_A = g^{x_A} \mod p \]

Private Value, \( x_B \)
Public Value, \( y_B \)

\[ y_B = g^{x_B} \mod p \]

\[ y_A^{x_B} = (g^{x_B})^{x_A} = g^{x_B x_A} = g^{x_B x_A} = (g^{x_A})^{x_B} = y_A^{x_B} \mod p \]

Some conditions on \( p \) and \( g \)

Note, this scheme leads to as many shared keys as people to talk with.

Security hole in Diffie-Hellman

The basic Diffie-Hellman scheme is subject to a man-in-the-middle (or bucket brigade) attack, in which a third party (Trudy, T) impersonates B while communicating with A, and impersonates A while communicating with B.

Shared key between A and T

\[ y_T^{x_A} = (g^{x_T})^{x_A} = g^{x_T x_A} = g^{x_A} x_T = (g^{x_A})^{x_T} = y_A^{x_T} \mod p \]

Shared key between B and T

\[ y_B^{x_T} = (g^{x_B})^{x_T} = g^{x_B x_T} = g^{x_T x_B} = (g^{x_T})^{x_B} = y_T^{x_B} \mod p \]
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication

Public key cryptography

**symmetric key crypto**

- Requires sender, receiver know shared secret key
- Not easy to agree on key in first place (particularly if never "met")
- In the late 60's, most researchers thought a better system was impossible

**public key cryptography**

- radically different approach [Diffie-Hellman76, RSA78]
  - But real inventors are J. Ellis, C. Cocks and M. Williamson (between 1969 and 1975, at British IA, declassified in 1997)
- 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, \( m \)  \rightarrow \text{encryption algorithm}  \rightarrow \text{ciphertext}  \rightarrow \text{decryption algorithm}  \rightarrow \text{plaintext message} \( m = K_B(K_B^+(m)) \)

Note: Anyone can send \( K_B^+(m) \) to B!

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, Adleman algorithm
RSA: getting ready

- A message is a bit pattern
- A 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 \( n = pq \), \( z = (p-1)(q-1) \)
   No known algo to find \( p, q \) knowing \( n \)
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 bit pattern, \(m < n\), compute
\[
c = m^e \mod n
\]

2. To decrypt received bit pattern, \(c\), compute
\[
m = c^d \mod n
\]

```
Magic happens!  m = \left( m^e \mod n \right)^d \mod n
```

RSA example:

Bob chooses \(p=5, q=7\). Then \(n=35, z=24\).
\(e=5\) (so \(e, z\) relatively prime).
\(d=29\) (so \(ed-1\) exactly divisible by \(z\)).

Encrypting 8-bit messages.

<table>
<thead>
<tr>
<th>encrypt: bit pattern</th>
<th>m</th>
<th>(m^e)</th>
<th>(c = m^e \mod n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001100</td>
<td>12</td>
<td>24832</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>decrypt: (c)</th>
<th>(c^d)</th>
<th>(m = c^d \mod n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>481968572106750915091411825223071697</td>
<td>12</td>
</tr>
</tbody>
</table>
RSA: Why is that \( m = (m^e \mod n)^d \mod n \)

Useful number theory result: If \( p, q \) prime and \( n = pq \), then:
\[
x^y \mod n = x \mod (p-1)(q-1) \mod n
\]

\[
(m^e \mod n)^d \mod n = m^{ed} \mod n
\]

\[
= m^{ed} \mod (p-1)(q-1) \mod n
\]

(using number theory result above)

\[
= m^1 \mod n
\]

(since we chose \( ed \) to be divisible by \( (p-1)(q-1) \) with remainder 1)

\[
= m
\]

RSA: another important property

The following property will be very useful later:

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

use public key first, followed by private key
use private key first, followed by public key

Result is the same!
Why?
Why \( K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m)) \) ?

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

**Generating RSA keys**

- Have to find big primes \(p\) and \(q\)
- Approach: make good guess then apply testing rules (see Kaufman)
Session keys

- Exponentiation 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

Computational difficulty

- Breaking a cryptographic algorithm is possible by trying all possible keys (brute force)
- If computers get $N$ times faster:
  - Making a key $\log_2(N)$ longer will make the bad guy's job as hard as it was before the advance in computer speed
  - However, it will be much easier for the good guys to encrypt: basically his job is almost $N$ times faster (not quite because the increase of the length of the key slows down the process a bit)
- So, the faster computers get, the better life gets for the good guys!
Chapter 3 roadmap

3.1 What is network security?

3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography

3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification

3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication

Message Integrity

- Allows communicating parties to verify that received messages are authentic
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained

- Let's first talk about message digests
Message Digests

- Function $H(\ )$ that takes as input an arbitrary length message and outputs a fixed-length string: “message digest”
- Note that $H(\ )$ is a many-to-1 function
- $H(\ )$ is often called a “hash function”

Desirable properties:
- Easy to calculate
- Irreversibility: Can't determine $m$ from $H(m)$
- Collision resistance: Computationally difficult to produce $m$ and $m'$ such that $H(m) = H(m')$
- Seemingly random output

Internet checksum: poor message digest

Internet checksum has some properties of hash function:
- Produces fixed length digest (16-bit sum) of input
- Is many-to-one

But given message with given hash value, it is easy to find another message with same hash value.

Example: Simplified checksum: add 4-byte chunks at a time:

<table>
<thead>
<tr>
<th>message</th>
<th>ASCII format</th>
<th>message</th>
<th>ASCII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I O U 1</td>
<td>49 4F 55 31</td>
<td>I O U 9</td>
<td>49 4F 55 39</td>
</tr>
<tr>
<td>0 0 . 9</td>
<td>30 30 2E 39</td>
<td>0 0 . 1</td>
<td>30 30 2E 31</td>
</tr>
<tr>
<td>9 B O B</td>
<td>39 42 D2 42</td>
<td>9 B O B</td>
<td>39 42 D2 42</td>
</tr>
<tr>
<td>B2 C1 D2 AC</td>
<td>different messages</td>
<td>B2 C1 D2 AC</td>
<td>but identical checksums!</td>
</tr>
</tbody>
</table>
Hash Function Algorithms

- **MD5 hash function widely used (RFC 1321)**
  - Computes 128-bit message digest in 4-step process.
  - Recent (2005) attacks on MD5
- **SHA-1 is also used.**
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Length of a Hash

- The hash should be at least 128 bits long, preferably more
- Would take trying approx $2^{128}$ possible messages before finding one that maps to a given hash
- Note however, that it would take only trying $2^{64}$ possible messages before finding two that map to the same hash with probability 50%
- **Birthday attack:**
  - With roughly $\sqrt{365}$ people, the odds are 50% that two have the same birthday
  - With $N$ people and $k = 365$ days, there are $N(N-1)/2$ pairs, and for each pair the probability is roughly $1/k$ to map to the same day (for large $k$).
  - So, if $N$ is such that $N(N-1) = k$, the odds are 50% that two have the same birthday. So $N$ is approximately the square root of $k$.
    - Note that the exact probability that at least 2 among $N$ people have the same birthday is $1 - (k! / ((k-N)! * k^N))$, which can be approximated by $1 - \exp(-N^2/2k)$.
      - This probability equals 0.5 when $N^2 = 2\ln(2) * k$, which is a better approximation than $N(N-1) = k$. 
Message Authentication Code (MAC)

- **s** = shared secret

- **Authenticates sender**
- **Verifies message integrity**
- **No encryption**
- **Also called “keyed hash”**
- **Notation:** \( MD_m = H(s||m) \); send \( m||MD_m \)

HMAC

- **Popular MAC standard, RFC 2104**
- **Can be used with either MD5 or SHA-1**
- **Addresses some subtle security flaws**

1. Concatenates secret to front of message
2. Hashes concatenated message
3. Concatenates the secret to front of digest
4. Hashes the combination again
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication

Digital Signatures

cryptographic technique analogous to handwritten signatures

- sender (Bob) digitally signs document, establishing he is document owner/creator
- Goal is similar to that of a MAC, except now use cryptography
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
Digital signatures

- A signature has a triple purpose:
  - **Authenticity**: the receiver can verify the claimed identity of the creator
  - **Nonrepudiation of origin**: the creator cannot later repudiate the contents of the message, the receiver cannot possibly have concocted the message himself
  - **Integrity**: the message has not been altered during transit

- Did MACs provide these 3 services?

- Two digital signature schemes:
  - Secret-key signatures
  - Public-key signatures

Secret-key signatures

- A wants to send a signed plaintext $P$ to $B$
- $K_{A-T}$ (resp. $K_{B-T}$) is a secret key shared by $A$ (resp. $B$) and $T$
- $K_T$ is a secret known by $T$ only
- $B$ has $K_T(A, B, t, P)$ as a proof of origin
- Problems:
  - Vulnerable to replay attack (solutions exist: see next slides)
  - Everyone must trust $T$
    - Does everyone trust a government, a bank, a lawyer, …?
    - $T$ can impersonate anyone to anyone!
  - Performance concern: $T$ must transcode every message
  - Single-point of failure
  - If $T$ is compromised, all the network resources are vulnerable
Protection against playback

- Some measures must be taken to prevent active intruders from playing back old messages

- Simple solution: timestamp
  - Including in every message a timestamp
  - The receiver stores all messages received during the last $T$ units
  - If a received message has a timestamp < Now - $T$, it is discarded: old message
  - If it has a timestamp in $[\text{Now} - T, \text{Now}]$ it is compared to the previously received messages

- Need to store many messages!

Protection against playback (2)

- Improvement
  - Every message contains an additional nonce
    - A nonce is a "random" number used once per interval of (at least) duration $T$
  - It suffices to store the received nonces during $T$, not the complete messages
Secret-key signatures (2)

- Add nonce and timestamp:
  - R is a nonce and t is a timestamp

\[
A, K_{A,T}(B, R, t, P) \rightarrow \text{Trustee } T \rightarrow K_{B,T}(A, R, t, P, K_{T}(A, B, t, P))
\]

Signing a hash of P is enough

\[
A, K_{A,T}(B, R, t, P) \rightarrow K_{B,T}(A, R, t, P, K_{T}(A, B, t, H(P)))
\]
Public-key Signatures

Simple digital signature for message m:

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

Everyone can verify signature (decrypt)

Dear Alice

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

Bob

Bob's message, m

Bob's private key

H: hash function

Digital signature (encrypt)

encrypted msg digest

equal?

Bob sends digitally signed message:

large message m

H: hash function

H(m)

Bob's private key $K_B$

encrypted msg digest $K_B(H(m))$

Alice verifies signature and integrity of digitally signed message:

large message m

H: hash function

H(m)

Bob's public key $K_B$

digital signature (decrypt)

encrypted msg digest $K_B(H(m))$

Bob's message, m, signed (encrypted) with his private key

Again: Signing a Hash is enough
Digital Signatures (more)

- suppose Alice receives msg m, digital signature $K_B(m)$
- 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

Encryption and Signature Combined in Public-Key Cryptography

P $\rightarrow$ Alice’s private key $K_A$ $\rightarrow$ Bob’s public key $K_B$ $\rightarrow$ Bob’s private key $K_B$ $\rightarrow$ Alice’s public key $K_A$ $\rightarrow$ P

Sign    Encrypt    Decrypt    Check signature
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naive approaches
   - Secret-key authentication
   - Public-key authentication

Public-key certification

❖ Motivation: Trudy plays pizza prank on Bob
   ❖ Trudy creates e-mail order:
     Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
   ❖ Trudy signs order with her private key
   ❖ Trudy sends order to Pizza Store
   ❖ Trudy sends to Pizza Store her public key, but says it’s Bob’s public key
   ❖ Pizza Store verifies signature; then delivers four pizzas to Bob
   ❖ Bob doesn’t even like Pepperoni
Public Key Certification

**public key problem:**
- When Alice obtains Bob’s public key (from web site, e-mail, USB stick), how does she **know** it is Bob’s public key, not Trudy’s?
- Distributing public keys is easier than distributing secret keys but some care is needed
- Even though they are public, they cannot just be exchanged by partners over the network or put in a public (even protected) database
  - The trouble with this approach is that it is subject to the bucket-brigade attack
  - Trudy can always intercept any request sent to the database and send back her own public key instead
  - This would allow this intruder to decrypt messages encrypted with that public key (but in fact intended for someone else)

**solution:**
- trusted certification authority (CA)

Certification Authorities

- **Certification Authority (CA):** binds public key to particular entity, E
- E (person, router) registers its public key with CA
  - E provides "proof of identity" to CA
  - CA creates certificate binding E to its public key
  - certificate containing E's public key digitally signed by CA: CA says "This is E's public key."

![Diagram of Certification Authorities]

```
Bob's public key  K_B

CA:
- CA private key  K_CA
- Certificate for Bob's public key, signed by CA

digital signature (encrypt)

K_CA(B, K_B)

B: Bob's identifying information

CA:
- CA private key  K_CA
- Certificate for Bob's public key, signed by CA

digital signature (encrypt)

K_CA(B, K_B)
```
Certification Authorities

- when Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere)
  - apply CA’s public key to Bob’s certificate, get Bob’s public key

A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)
- info about certificate issuer
- valid dates
- digital signature by issuer
Certificates: summary

- Primary standard X.509 (RFC 2459)
- Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity’s public key
  - Digital signature (signed with issuer’s private key)
- Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered “heavy”

PKCS standards

- PKCS = Public Key Cryptography Standard
- A set of standards (PKCS#1 to #15) for the encoding of information that will be signed or encrypted through RSA
- Encoding of
  - RSA public key
  - RSA private key
  - RSA signature
  - Short RSA-encrypted message (typically a secret key)
  - Short RSA-signed message (typically a message digest)
- Defined in ASN.1
Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naïve approaches
   - Secret-key authentication
   - Public-key authentication

Authentication

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

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

Failure scenario?
Authentication

**Goal:** Bob wants Alice to "prove" her identity to him

**Protocol ap1.0:** Alice says "I am Alice"

in a network, Bob cannot "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

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.

Authentication: another try

**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.

![Diagram showing the protocol ap3.0](image)

- Alice's IP addr
- Alice's password
- 'I'm Alice'
- 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.

![Diagram showing the protocol ap3.1](image)

- Alice's IP addr
- Encrypted password
- 'I'm Alice'

Failure scenario?
Authentication: another try

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

Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
   - Secret-key Cryptography
   - Public-key Cryptography
3.3 Message integrity
   - Message Authentication Code (MAC)
   - Digital Signatures
   - Public-key Certification
3.4 End point authentication
   - Naïve approaches
   - Secret-key authentication
   - Public-key authentication
Authentication: yet another try

**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

```
R
```

```
K_{A-B}(R)
```

```
“A I am Alice”
```

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

Failures, drawbacks?

Authentication: a variant with a MAC

**Goal:** replace secret key cryptography by cryptographic hash (no export problem)

**ap4.1:** to prove Alice “live”, Bob sends Alice nonce, R. Alice must return a MAC based on R and the shared secret key

```
“ I am Alice”
```

```
R
```

```
H(R+K_{A-B})
```

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

Failures, drawbacks?
**Bidirectional authentication**

**ap4.2:** run 2 separate authentications in //

```
“I am Alice”
```

```
“I am Bob”, R_B
```

```
R_A
```

```
H(R_B+K_{A-B})
```

```
H(R_A+K_{A-B})
```

Failures, drawbacks?

---

**Reflection attack in bidirectional authentication**

Combining two robust unidirectional protocols does not always result in a robust bidirectional protocol!

```
“I am Alice”
```

```
“I am Bob”, R_B
```

```
Take R_A = R_B
```

```
R_B
```

```
H(R_B+K_{A-B})
```

Postpone reply. Replay MAC!

```
H(R_B+K_{A-B})
```

Solution?
Using a Key Distribution Centre (KDC)

- All previous schemes require a shared key per pair of users
  - N users \(\rightarrow\) \(O(N^2)\) shared keys
- Solution:
  - Each user has only one secret key shared with a trusted KDC
  - \(K_S\) is a (secret) session key picked up by Alice
- \(K_{A,KDC}\) (resp. \(K_{B,KDC}\)) is a secret key shared by A (resp. B) and KDC

```
I am Alice. I want to use this session key for authentication: K_{A,KDC}(B, K_S)
```

```
I want to use this session key for authentication: K_{B,KDC}(A, K_S)
```

As is, it is subject to the replay attack
Requires timestamps and nonces

Combining Key Distribution and Secret-Key Authentication: Needham-Shroeder

1: A, B, R
2: \(K_A(B, K_S, K_B(A, K_S))\)
3: \(K_A(A, K_S), R_A\)
4: \(K_S(R_A), R_A\)
5: \(K_S(R_B)\)

- Steps 1-2: Alice asks the KDC a session key to talk to Bob
- Steps 3-5: Double authentication between Alice and Bob
Needham-Shroeder (2)

Steps 1-2: Alice asks the KDC a session key to talk to Bob
- Request-response between Alice and the KDC
  - Authenticated thanks to Alice’s secret key
  - Freshness guaranteed by the presence of the nonce R
- Alice gets a session key $K_s$ both in clear and encrypted with Bob’s secret key $K_B(A, K_s)$, the so-called ticket.
- The whole message being encrypted using Alice’s secret key.

Steps 3-5: Double authentication between Alice and Bob
- Alice sends the encrypted session key to Bob and initiates a challenge-response protocol with him (challenge = $R_A$)
- Bob proves he knows $K_s$ by sending back $K_s(R_A)$
- Bob challenges Alice too (with $R_B$), based on the same session key

Many possible variants of steps 3-4-5, e.g.:
- 3: $R_A$ or $K_s(R_A)$ or $3$: $K_s(R_A)$
- 4: $K_s(R_A, R_B)$ or 4": $K_s(R_A^{-1}), K_s(R_A)$
- 5: $K_s(R_A)$ or 5": $K_s(R_A^{-1})$

The Kerberos system is a variant of this scheme

Chapter 3 roadmap

3.1 What is network security?
3.2 Principles of cryptography
  - Secret-key Cryptography
  - Public-key Cryptography
3.3 Message integrity
  - Message Authentication Code (MAC)
  - Digital Signatures
  - Public-key Certification
3.4 End point authentication
  - Naïve approaches
  - Secret-key authentication
  - Public-key authentication
**Authentication with public key techniques: ap5.0**

ap4.* require shared symmetric key

Can we authenticate using public key techniques?

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

"I am Alice"

Bob computes $K_A^{-1}(R) = R$ and knows only Alice could have the private key, that encrypted $R$ such that $K_A^+(K_A^{-1}(R)) = R$

---

**ap5.0: security hole**

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

"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

$m = K_A^{-1}(K_T^+(m))$
ap5.0: security hole

Man (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!

Authentication with public key techniques: ap5.1

ap5.1: use certified public keys

"I am Alice"

Bob computes

\[ K_A(K_A(R)) = R \]

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

\[ K_A(K_A(R)) = R \]

"send me your certified public key"

\[ K_{CA}(A, K_A) \]
Authentication using Public-Key Cryptography:

ap5.2 Needham-Schroeder

We limit the description to the authentication part, omitting the exchanges of certified public keys.

This protocol designed in 1978 was in all the textbooks. In 1995, it was proved incorrect.

Security hole in ap5.2

The end result of this interchange is that:
- Alice believes she has established a session with Trudy (thinking Trudy is e.g. a server, but Trudy will just redirect traffic to Bob who will actually provide the service).
- Bob believes he has authenticated Alice and will provide the service to her.

Later on:
- Trudy will send the bill to Alice who will pay (Trudy is her provider).
- Bob will send the bill to Alice who will refuse (Alice never asked any service to Bob).
Authentication using Public-Key Cryptography: ap5.3 Needham-Schroeder

Corrected protocol: add originator in encrypted part

“\text{I am Alice}$$, K_B^*(A, R_A)$$

“\text{I am Bob}$$, K_A^*(B, R_A, R_B)$$

Only Bob could get $R_A$, so it must be Bob. Alice extracts $R_B$

$K_B^*(A, R_B)$$

Bob extracts $R_A = K_B^*(K_B^*(R_A))$

Only Alice could get $R_B$, so it must be Alice.

Moral of the story: designing a correct authentication protocol is much harder than it looks!

Advantages of CAs over KDCs

- The CA is the public key equivalent to the KDC
- The CA does not need to be on-line
- If the CA crashes, the network is not disabled
- Certificates are not security-sensitive
- A compromised CA cannot decrypt conversations
  - But a compromised CA can fool Alice into accepting an incorrect public key for Bob, and then the CA can impersonate Bob to Alice

3: Principles of Network Security 111

3: Principles of Network Security 112
Principles of Network Security

(summary)

Basic techniques:
- cryptography (symmetric-key and public-key)
- message integrity (MAC, digital signature, certification)
- end-point authentication

Next we’ll see how to use them in many different security scenarios
- secure email
- secure DNS
- secure transport (SSL)
- secure network (IPsec, secure routing)
- secure link (802.11)

We’ll also study techniques not based on cryptography
- Securing Ethernet switches