COMP 3331/9331: Computer Networks and Applications

Week 11
Network Security

Reading Guide: Chapter 8: 8.1 – 8.6, 8.9
Announcements

- Assignment 2
  - Deadline: Friday, 3rd June 2016 (FIRM)

- NO LECTURES IN WEEK 12 DUE TO TRAVEL

- Final Lecture in Week 13 on Wednesday, 2nd June in Law Theatre G04
  - Any leftover content
  - Discussion about final exam
  - Some practice problems

- Final Exam:
  - Start preparations well in advance

- CATEI Surveys – Feedback is important for us
  - Accessible via myUNSW
  - Form A – course, Form B – teaching
  - PLEASE FILL IN BOTH FORMS
Chapter 8: Network Security

Chapter 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
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL (TIME PERMITTING)
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
What is network security?

**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

**access and availability:** services must be accessible and available to users
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., on-line 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! See section 1.6

- **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)
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
The language of cryptography

plaintext message

\[ K_A(m) \] ciphertext, encrypted with key \( K_A \)

\[ m = K_B(K_A(m)) \]
Symmetric key cryptography

Plaintext message, $m$ → encryption algorithm $K_S$ → ciphertext $K_S(m)$ → decryption algorithm $K_S$ → plaintext $m = K_S(K_S(m))$

**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

  plaintext:  abcdefghijklmnopqrstuvwxyz
  \[\text{ciphertext: } mnbcxzasdfghjklpouiuytrewq\]

  e.g.:  Plaintext: bob. i love you. alice
  \[\text{ciphertext: } nkn. s gktc wky. mgsbc\]

*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
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,b
- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext
A more sophisticated encryption approach

- $n$ substitution ciphers, $M_1, M_2, \ldots, M_n$
- cycling pattern:
  - e.g., $n=4$: $M_1, M_3, M_4, M_3, M_2; \quad M_1, M_3, M_4, M_3, M_2; \ldots$
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from $M_1$, o from $M_3$, g from $M_4$

Encryption key: $n$ substitution ciphers, and cyclic pattern
- key need not be just $n$-bit pattern
Two types of symmetric ciphers

- **Stream ciphers**
  - encrypt one bit at a 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
- \( m(i) = \) ith bit of message
- \( ks(i) = \) ith bit of keystream
- \( c(i) = \) ith bit of ciphertext
- \( c(i) = ks(i) \oplus m(i) \quad (\oplus = \text{exclusive or})\)
- \( m(i) = ks(i) \oplus c(i)\)
RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Extensively analyzed and considered good
  - Key can be from 1 to 256 bytes
  - Used in WEP for 802.11
  - Can be used in SSL
Block Cipher

- Ciphertext processed as $k$ bit blocks
- 1-to-1 mapping is used to map $k$-bit block of plaintext to $k$-bit block of ciphertext
- E.g: $k=3$ (see table)
  - \[ \begin{array}{c|c}
  000 & 110 \\
  111 & 001 \\
  001 & 111 \\
  010 & 101 \\
  011 & 100 \\
  100 & 011 \\
  101 & 010 \\
  110 & 000 \\
  \end{array} \]
- Possible permutations $= 8!$ (40,320)
- To prevent brute force attacks
  - Choose large $K$ (64, 128, etc)
- Full-block ciphers not scalable
  - E.g., for $k = 64$, a table with $2^{64}$ entries required
  - Instead use function that simulates a randomly permuted table
If only a single round, then one bit of input affects at most 8 bits of output.

In the 2nd round, the 8 affected bits get scattered and inputted into multiple substitution boxes.

How many rounds?
- How many times do you need to shuffle cards?
- Becomes less efficient as n increases.

Examples: DES, 3DES, AES

From Kaufman et al
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
- 16 identical “rounds” of function application, each using different 48 bits of key
- final permutation
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
Cipher Block Chaining

- cipher block: if input block repeated, will produce same cipher text:
  - $t=1$
    - $m(1) = \text{"HTTP/1.1"}$
    - $c(1) = \text{"k329aM02"}$
  - $t=17$
    - $m(17) = \text{"HTTP/1.1"}$
    - $c(17) = \text{"k329aM02"}$

- cipher block chaining: XOR ith input block, $m(i)$, with previous block of cipher text, $c(i-1)$
  - $c(0)$ is an initialisation vector transmitted to receiver in clear
  - what happens in “HTTP/1.1” scenario from above?
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
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, $m$ → encryption algorithm $\rightarrow$ ciphertext $\rightarrow$ decryption algorithm $\rightarrow$ plaintext message $m = K_B^-(K_B^+(m))$

$K_B^+$: Bob’s public key

$K_B^-$: Bob’s private key
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
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: \( x=14, n=10, d=2: \)
  \( (x \mod n)^d \mod n = 4^2 \mod 10 = 6 \)
  \( x^d = 14^2 = 196 \quad x^d \mod 10 = 6 \)
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 \( n = pq, \ z = (p-1)(q-1) \)

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
   \[ m = c^d \mod n \]

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

*magic happens!*
RSA example:


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

Encrypting 8-bit messages.

<table>
<thead>
<tr>
<th>bit pattern</th>
<th>$m$</th>
<th>$m^e$</th>
<th>$c = m^e \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001000</td>
<td>12</td>
<td>24832</td>
<td>17</td>
</tr>
</tbody>
</table>

Decrypt:

<table>
<thead>
<tr>
<th>$c$</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>
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,
  \[
  c^d \mod n = (m^e \mod n)^d \mod n \\
  = m^{ed} \mod n \\
  = m^{(ed \mod z)} \mod n \\
  = m^l \mod n \\
  = m
  \]
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 \( K_B^- (K_B^+ (m)) = m = K_B^+ (K_B^- (m)) \)?

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
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
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
Authentication

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

**Protocol ap 1.0:** Alice says “I am Alice”

Failure scenario??
Authentication

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

**Protocol ap 1.0:** 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.

![Diagram showing Alice and her IP address sending a password, followed by a response and a failure scenario.](image)
**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??
Authentication: yet another try

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

record and playback *still* works!
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

 Failures, drawbacks?

```
  “I am Alice”
              R
              K_{A-B}(R)                          Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!
```

Authentication: ap5.0

ap4.0 requires shared symmetric key
- can we authenticate using public key techniques?

*ap5.0* use nonce, public key cryptography

```
“I am Alice”

R

K_A^-(R)

“send me your public key”

K_A^+(R)

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
```
ap5.0: security hole

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

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

Trudy gets \( m = K_T^-(K_T^+(m)) \)

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!
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
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, $K_B^-(m)$

Bob’s message, m

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

Bob’s private key

Public key encryption algorithm

$m, K_B^-(m)$

Bob’s message, m, signed (encrypted) with his private key
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
Message digests

computationally expensive to public-key-encrypt long messages

**goal:** fixed-length, easy-to-compute digital “fingerprint”

- apply hash function $H$ to $m$, get fixed size message digest, $H(m)$.

Hash function properties:
- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest $x$, computationally infeasible to find $m$ such that $x = H(m)$
Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:
 ➤ produces fixed length digest (16-bit sum) of message
 ➤ is many-to-one

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

<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>
</tbody>
</table>

B2 C1 D2 AC  different messages  but identical checksums!  B2 C1 D2 AC
Hash function algorithms

- **MD5 hash function widely used (RFC 1321)**
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string $x$, appears difficult to construct msg $m$ whose MD5 hash is equal to $x$

- **SHA-1 is also used**
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
Digital signature = signed message digest

Bob sends digitally signed message:

- Large message $m$
- Hash function $H$:
  - $H(m)$
- Digital signature (encrypt) using Bob's private key $K_B^-$:
  - $K_B^-(H(m))$
- Encrypted msg digest

Alice verifies signature, integrity of digitally signed message:

- Large message $m$
- Hash function $H$:
  - $H(m)$
- Digital signature (decrypt) using Bob's public key $K_B^+$:
  - $K_B^+(H(m))$
- Encrypted msg digest

Digital signature = signed message digest
Recall: ap5.0 security hole

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

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

Trudy gets
\[ m = K_T^-(K_T^+(m)) \]
sends \( m \) to Alice
encrypted with Alice’s public key

\[ K_T^+(m) \rightarrow K_T^- \]

Send me your public key
\[ K_A^+ \rightarrow K_A^- \]

Send me your public key
\[ K_T^+ \rightarrow K_T^- \]
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 pepperoni pizzas to Bob
  - Bob doesn’t even like pepperoni
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”
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”
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

Alice:
- generates random symmetric private session key, $K_S$
- encrypts message with $K_S$ (for efficiency)
- also encrypts $K_S$ with Bob’s public key
- sends both $K_S(m)$ and $K_B^+(K_S)$ to Bob
Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

**Bob:**
- uses his private key to decrypt and recover $K_S$
- uses $K_S$ to decrypt $K_S(m)$ to recover m
Secure e-mail (continued)

- Alice wants to provide sender authentication and message integrity.

Alice digitally signs the message and sends both the message (in the clear) and the digital signature.

Internet

m ----> H(·) ----> K_A^- (H(m))

K_A^- (·) ----> + ----> Internet

K_A^+ (·) ----> - ----> H(m)

Network Security 8-68
Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.

Alice uses three keys: her private key, Bob’s public key, newly created symmetric key.
Secure E-mail: PGP

- De-factor standard for email encryption
- On installation PGP creates public, private key pair
  - Public key posted on user’s webpage or placed in a public key server
  - Private key protected by password
- Option to digitally sign the message, encrypt the message or both
- MD5 or SHA for message digest
- CAST, triple-DES or DEA for symmetric key encryption
- RSA for public key encryption
Secure E-mail: PGP

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yhHJRHHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----

Figure 8.22 ♦ A PGP signed message

-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=1KhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfghbnmKlok8iy6gThlp
-----END PGP MESSAGE

Figure 8.23 ♦ A secret PGP message
Quiz

- Suppose a CA creates Bob’s certificate, which binds Bob’s public key to Bob. This certificate is signed with
  - Bob’s private key
  - Bob’s public key
  - The CA’s private key
  - The CA’s public key

- Suppose Bob wants to send Alice a digital signature for the message $m$. To create the digital signature
  - Bob applies a hash function to $m$ and encrypts the result with his private key
  - Bob applies a hash function to $m$ and encrypts the result with Alice’s public key
  - Bob encrypts $m$ with his private key and then applies a hash function to the result
  - Bob applies a hash function to $m$ and encrypts the result with his public key
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
Firewalls

*firewall*

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

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

Should arriving packet be allowed in? Departing packet let out?
Stateless packet filtering: example

- **example 1**: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or destination 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.
### Stateless packet filtering: more examples

<table>
<thead>
<tr>
<th>Policy</th>
<th>Firewall Setting</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 source 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>all</td>
</tr>
</tbody>
</table>

Network Security 8-81
Application gateways

- filters 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.
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
Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions

- **IDS: intrusion detection system**
  - *deep packet inspection:* look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning
    - network mapping
    - DoS attack
Intrusion detection systems

- multiple IDSs: different types of checking at different locations
Network Security: roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and digital signatures
8.4 End-point Authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL (TIME PERMITTING)
8.7 Network layer security: Ipsec (NOT COVERED)
8.8 Securing wireless LANs (NOT COVERED)
8.9 Operational security: firewalls and IDS
SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
    - https
    - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation - TLS: transport layer security, RFC 2246
- provides
  - confidentiality
  - integrity
  - authentication
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface
## SSL and TCP/IP

<table>
<thead>
<tr>
<th>Application</th>
<th>TCP</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal application</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>SSL</th>
<th>TCP</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>application with SSL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available
Could do something like PGP:

but want to send byte streams & interactive data

want set of secret keys for entire connection

want certificate exchange as part of protocol: handshake phase
Toy SSL: a simple secure channel

- **handshake**: Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- **key derivation**: Alice and Bob use shared secret to derive set of keys
- **data transfer**: data to be transferred is broken up into series of records
- **connection closure**: special messages to securely close connection
Toy: a simple handshake

MS: master secret
EMS: encrypted master secret
Toy: key derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption

- four keys:
  - $K_c = \text{encryption key for data sent from client to server}$
  - $M_c = \text{MAC key for data sent from client to server}$
  - $K_s = \text{encryption key for data sent from server to client}$
  - $M_s = \text{MAC key for data sent from server to client}$

- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data and creates the keys
**Toy: data records**

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
  - want to use variable-length records

| length | data | MAC |
Toy: sequence numbers

- **problem**: attacker can capture and replay record or re-order records
- **solution**: put sequence number into MAC:
  - $\text{MAC} = \text{MAC}(M_x, \text{sequence}||\text{data})$
  - note: no sequence number field

- **problem**: attacker could replay all records
- **solution**: use nonce
Toy: control information

- **problem**: truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is.

- **solution**: record types, with one type for closure
  - type 0 for data; type 1 for closure

- $$\text{MAC} = \text{MAC}(M_x, \text{sequence}||\text{type}||\text{data})$$
Toy SSL: summary

- hello
- certificate, nonce
- $K_B^+(MS) = EMS$
- type 0, seq 1, data
- type 0, seq 2, data
- type 0, seq 1, data
- type 0, seq 3, data
- type 1, seq 4, close
- type 1, seq 2, close

encrypted

bob.com
Toy SSL isn’t complete

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer
SSL cipher suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
  - client offers choice
  - server picks one

common SSL symmetric ciphers
- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption
- RSA
Real SSL: handshake (1)

Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)
Real SSL: handshake (2)

1. Client sends list of algorithms it supports, along with client nonce
2. Server chooses algorithms from list; sends back: choice + certificate + server nonce
3. Client verifies certificate, extracts server’s public key, generates pre_master_secret, encrypts with server’s public key, sends to server
4. Client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. Client sends a MAC of all the handshake messages
6. Server sends a MAC of all the handshake messages
Real SSL: handshaking (3)

last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the-middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted
Real SSL: handshaking (4)

- why two random nonces?
- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy’s messages will fail Bob’s integrity check
SSL record protocol

- **data**
  - *data fragment*:
    - encrypted data and MAC
  - *MAC*
  - *record header*:
    - content type; version; length

- **MAC**: includes sequence number, MAC key $M_x$

- **fragment**: each SSL fragment $2^{14}$ bytes (~16 Kbytes)
SSL record format

<table>
<thead>
<tr>
<th>1 byte</th>
<th>2 bytes</th>
<th>3 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>content type</td>
<td>SSL version</td>
<td>length</td>
</tr>
</tbody>
</table>

data

MAC

data and MAC encrypted (symmetric algorithm)
Real SSL connection

everything henceforth is encrypted

TCP FIN follows
Key derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - produces master secret
- master secret and new nonces input into another random-number generator: “key block”
  - because of resumption: TBD
- key block sliced and diced:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)
Network Security (summary)

basic techniques……

- cryptography (symmetric and public)
- message integrity
- end-point authentication

…. used in many different security scenarios

- secure email
- secure transport (SSL) – TIME PERMITTING
- IP sec (NOT COVERED)
- 802.11 (NOT COVERED)

operational security: firewalls and IDS