Chapter 8
Network Security

Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Some of the slides from the text books authors site
Why Network Security?

Malicious people share your network
  People who want to snoop
  People who want to destroy
  People who want to corrupt
  People who want to pretend
  People who want to steal

Problem made more severe as Internet becomes more commercialized

Active and passive attacks
Dealing with Network Security

Network security can be broken into several related areas:

**Authentication**: Proving that someone is who she says she is
- Identity verification (use something you know, you have, you are)

**Confidentiality**: hiding data from prying eyes
- Keep data secret from unauthorized users

**Integrity**: Proving that a message has not been intentionally corrupted by a third party
- No tampering by someone in the middle

**Non repudiation**: Once someone sends a message, or conducts a transaction, she can’t later deny the contents of that message
- Undeniability
Available Security Tools

Cryptography/Encryption:
- Encode a message in a way that only the communicating parties can interpret it
- Used for secrecy and authentication

Signatures
- Allow for the authentication of a message’s sender and the message’s integrity
- Used for authentication, nonrepudiation and integrity control
Cryptography

Encoding a message in a way that only the communicating parties can interpret it

Notation:

Encryption: $C = E_K(P)$  
Decryption: $P = D_K(P)$
Rules for Encryption

Encryption requires both an encryption algorithm and an encryption “key”

Key is a string which controls how the algorithm encrypts

Algorithm:

Should be public and known to all
  Inspires trust that the algorithm works

Keys:

Should be long enough to prevent easy breaking of the encryption
Should be short enough to keep algorithm efficient

Typical key lengths: 56-bit, 128-bit, 256-bit, 512-bit
Substitution Ciphers

Every letter (or group of letters) is replaced by another letter (or group of letters)

Example:

Caesar cipher:
- a/D, b/E, c/F, d/G, …, z/C
- Each letter is replaced by a shift: Succ(1), Succ (2), pred(1), pred(2)

Monoalphabetic cipher:
- a/Q, b/W, c/E, …
- Need a table to provide the mapping

Easy to break by analyzing statistical properties of written language
Polyalphabetic encryption

Use a set of monoalphabetic substitutions in a cycle $S_1, S_2, S_3$

N=3, LADYGAGA

L is replaced by I according to $S_1$
A is replaced by D according to $S_2$
D is replaced by I according to $S_3$
Y is replaced by O according to $S_1$
G is replaced by T according to $S_2$ .. And So on

Key is the 3 different monoalphabetic siphers
Types of Encryption

Transposition Ciphers

Instead of substituting letters in the plaintext, we change their order

\[
\begin{array}{c|ccccccc}
A & N & D & R & E & W \\
1 & 4 & 2 & 5 & 3 & 6 \\
T & h & i & s & i & s & a & m & e & s & s & a & g & e & i & w & o & u \\
& l & d & l & i & k & e \\
& t & o & e & n & c & r & y & p & t & n & o & w \\
\end{array}
\]

Key = ANDREW
Plaintext = thisisamessageiwouldliketoencryptnow
Ciphertext = tiihssaesmsagioewulIkdietecdnrytopnw

- Also easy to break by analyzing structure of language
Types of Encryption (cont’d)

Transposition Ciphers
  Transpose rows and columns

Key = ANDREW
Plaintext = thisisamessageiwould liketoencryptnow
Ciphertext = tagltyieiletisokco hmedopsswinnsauerw

● Also easy to break by analyzing structure of language
Most actual encryption algorithms use a complex combination of substitution and transposition

Examples:

Data Encryption Standard (DES)
  Multiple iterations of substitution and transposition using a 56-bit key
designed by IBM with input from the NSA

DES chaining
  Multiple stages of DES coding, in which the input of each stage is the output of previous stages

International Data Encryption Algorithm (IDEA)
  uses a 128-bit key
Symmetric key cryptography

plaintext → encryption algorithm → ciphertext → decryption algorithm → plaintext

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?
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

\[ m(i) = \text{ith bit of message} \]
\[ ks(i) = \text{ith bit of keystream} \]
\[ c(i) = \text{ith bit of ciphertext} \]

\[ c(i) = ks(i) \oplus m(i) \quad (\oplus = \text{exclusive or}) \]
\[ m(i) = ks(i) \oplus c(i) \]

keystream generator

key

keystream

pseudo random
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 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?
Block ciphers

How many possible mappings are there for $k=3$?

How many 3-bit inputs?
How many permutations of the 3-bit inputs? $8!$
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
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
    (actually encrypt, decrypt, encrypt)
Triple DES

Use three keys and three executions of the DES algorithm (encrypt-decrypt-encrypt)

\[
C = \text{ciphertext} \\
P = \text{Plaintext} \\
E \text{ is encryption, } D \text{ is decryption} \\
K_1 \text{ is 56 bit key, } K_2 \text{ is 56 bit key, } K_3 \text{ is 56 bit key} \\
\text{Effective key length of 168 bits}
\]

\[
C = E_{K_3}(D_{K_2}(E_{K_1}(P)))
\]

\[
P = D_{K_1}(E_{K_2}(D_{K_3}(P)))
\]
Example

K1: SUCC (1)
K2: SUCC(2)
K3: SUCC(3)

P=GAGA
C1=EK1(GAGA) →
C2=DK2(          ) →
C3=EK3(           ) →

C=
P1=DK3(           ) →
P2=EK2 (              ) →
P3=DK1 (              ) →
P=
Symmetric key crypto: DES

**DES operation**

- Initial permutation
- 16 identical "rounds" of function application, each using different 48 bits of key
- Final permutation
Types of Encryption

Problem with all the cryptography algorithms used so far:
   if the key is stolen, any message can be decrypted
Is there a way to do cryptography without worrying about this concern?

Yes, public key cryptography
Public Key Cryptography

Uses two different keys: an encryption key and a decryption key

Each user holds:
- a private decryption key to decrypt messages sent to her
- a public encryption key that everyone else should use to encrypt messages that are sent to her

Important property of public key cryptography algorithms:
- Private decryption key cannot be easily determined by knowing the public encryption key
Public key cryptography

plaintext message, m

encryption algorithm

plaintext message, m

Bob's public key, \( K_B^+ \)

Bob's private key, \( K_B^- \)

ciphertext \( K_B^+ (m) \)

decryption algorithm

plaintext message \( m = K_B (K_B^+ (m)) \)

plaintext message, m
RSA: Encryption, decryption

Given \((n, e)\) and \((n, d)\)

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

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

An Example

Two keys:
\( K_{\text{pub,sally}} \)
\( K_{\text{priv,sally}} \)

Two keys:
\( K_{\text{pub,jeff}} \)
\( K_{\text{priv,jeff}} \)
Public Key Cryptography

An Example

Sally and Jeff exchange *public* keys

Two keys:

- $K_{\text{pub}, \text{sally}}$
- $K_{\text{priv}, \text{sally}}$

Two keys:

- $K_{\text{pub}, \text{jeff}}$
- $K_{\text{priv}, \text{jeff}}$

Sally and Jeff exchange *public* keys
Public Key Cryptography

An Example

If Jeff wants to send an encrypted plaintext message $P$ to Sally, he uses Sally’s public key to encrypt the message into $C$.

Mathematically, this is represented as:

$$C = E_{K_{pub,sally}}(P)$$

Where $K_{pub,sally}$ is Sally's public key and $K_{priv,sally}$ is Sally's private key.
Public Key Cryptography

An Example

Sally uses her private key to decrypt the message \( C \) from Jeff. Only Sally can decrypt messages that are encrypted using her public key. A message to Sally cannot be decrypted using Sally’s public key.
Authentication: the problem of proving that a user is who he says he is

Two approaches:
- based on secret key cryptography
- based on public key cryptography

Mike sends a message to Sally pretending to be Jeff
Secret Key Authentication

Sally and Jeff share a secret key $K$
Sally and Jeff issue authentication challenges

- Sally sends $\text{Identity}=\text{Sally}$
- Jeff sends $\text{Challenge } R_{Jeff}$
- Sally sends $E_K(R_{Jeff})$
- Jeff sends $\text{Challenge } R_{Sally}$
- Sally sends $E_K(R_{Sally})$
Public Key Authentication

Sally and Jeff issue authentication challenges using public keys
Efficiency issue with Public Key Cryptography

Public key encryption methods tend to be slower than secret key encryption, because they rely on more expensive computations. So, it is common in practice to use public key cryptography only to encode a private secret key ($K_S$), which is then used to encrypt all other data shared between the two communicating parties.
Public Key Cryptography

(cont’d)

Why public key cryptography is so powerful:
   No secret keys need to be distributed
   Only the receiver of encrypted information holds the secret key

Examples of public key algorithms:
   Merkle-Helman knapsack
   Rivest-Shamir-Adleman (RSA)
   Pretty Good Privacy (PGP)
Digital Signatures

A digital signature...

- allows the receiver to authenticate the identity of the sender
- prevents the sender from later claiming she sent a different (or no) message
- prevents the receiver from constructing a message that appears as if it came from the sender
Public Key Signatures

Let us use a public key cryptography system with the following properties:

\[ D_{pu}(E_{pr}(M)) = M \]

\[ D_{pr}(E_{pu}(M)) = M \]
Jeff applies his private key to decrypt the message. Then he applies Sally’s public key to “encrypt” the Sally-decrypted message. If a meaningful message (P) is obtained, Jeff can be sure that Sally sent it.
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 MAC, except now use public-key cryptography
verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
Digital Signatures

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

Bob’s message, m, signed (encrypted) with his private key $K_B(m)$
Digital signature = signed message digest

Bob sends digitally signed message:

- Large message $m$
- Hash function $H$: $H(m)$
- Digital signature (encrypt) $K_B^{-1}(H(m))$
- Encrypted msg digest $K_B^{-1}(H(m))$

Alice verifies signature and integrity of digitally signed message:

- Large message $m$
- Hash function $H$: $H(m)$
- Encrypted msg digest $K_B^{-1}(H(m))$
- Digital signature (decrypt) $K_B^+(H(m))$
- $H(m)$

Digital signature = signed message digest
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.
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
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

$K_B^+$
digital signature (decrypt)

$K_{CA}^+$
Bob’s public key

Network Security
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, certification authorities	en often considered “heavy”
Message Integrity

Original Message content Has not been changed/tampered with
Use Message Digest
One-way hash functions
F(M)→ D
Given D, it is impossible to find M
Also, it is not possible to find another M’ such that
F(M’)=D
Even a single bit change to M will result in a different digest
Two well-known schemes
   MD5 and SHA-1
Message Integrity

MD5: Message Digest uses a 128-bit message digest
Break the message into 512 bit blocks
   From an initial 128 bit (four 32-bit words) key compute the hash for the first block, then use this hash as the seed for the next block and so on until the final value which is the digest

SHA-1: Secure Hash uses a 160-bit digest
Same idea as MD5 but uses 160-bit (five 32-bit words)
Still a function iteratively applied to successive 512 bit blocks of the message
Output hash value is the final buffer value
Secure Socket Layer SSL

SSL: used in authentication of client and server
A server sends a certificate
Client verifies the certificate and uses public key to send encrypted data to the server
Https is the protocol that enables secure communication between client and server
SSL Architecture

- SSL Handshake Protocol
- SSL Change Cipher Spec Protocol
- SSL Alert Protocol
- HTTP

SSL Record Protocol

TCP

IP
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

$K_B^+(MS) = EMS$
SSL Protocol

Client initiates a connection

Client verifies the server’s ID by using servers public key. If requested by the server, the client sends its ID.

Client verifies the server’s ID by using servers public key. If requested by the server, the client sends its ID.

When the authentication is complete, the client sends the server a session key encrypted using the server’s public key.

Once a session key is established, secure communications commence between client and server.

Server responds by sending the client its certificate. Identity signed by its private key. The server may also, optionally, request the client’s ID for client authentication.

Server

Client
Figure 1: SSL handshake and data
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 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

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 and MAC encrypted (symmetric algorithm)**
Real Connection

TCP Fin follow

Everything henceforth is encrypted

Network
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)
Public –key certificate

Certificate consists of a public key and a user ID of the owner
Signed by the private key of certificate authority
Receiver of certificate can verify owner by using the signature of the CA
SSL
What’s in a Certificate

Legal owner of the certificate
May be a chain of certificates leading to a well-known provider
The physical location of owner (city, state, country, etc)
Valid dates for the certificate
Server’s public key
HTTPS

HTTPS (HTTP over SSL)

combination of HTTP & SSL/TLS to secure communications between browser & server
documented in RFC2818
no fundamental change using either SSL or TLS

use https:// URL rather than http://
and port 443 rather than 80

encrypts

URL, document contents, form data, cookies, HTTP headers