Network Security Overview

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

- What is network security?
- Principles of cryptography
- Message integrity
What is network security?
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
- Cell phones, RFID tags, NFC device, sensors
- other examples?
There are bad guys (and girls) out there!

Q: What can a “bad guy” do?
There are bad guys (and girls) out there!

Q: What can a “bad guy” do?
A: A lot!

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

- What is network security?
- Principles of cryptography
- Message integrity
The language of cryptography

m plaintext message
$K_A(m)$ ciphertext, encrypted with key $K_A$
$m = K_B(K_A(m))$
Simple encryption scheme

substitution cipher: substituting one thing for another
  - monoalphabetic 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
Polyalphabetic encryption

- n monoalphabetic ciphers, $M_1, M_2, \ldots, M_n$
- Cycling pattern:
  - e.g., n=4, $M_1, M_3, M_4, M_3, M_2$; $M_1, M_3, M_4, M_3, M_2$
- 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
Breaking an encryption scheme

- **Cipher-text only attack:** Trudy has ciphertext that she can analyze
  - Two approaches:
    - Search through all keys: must be able to differentiate resulting plaintext from gibberish
    - Statistical analysis

- **Known-plaintext attack:** Trudy has some plaintext corresponding to some ciphertext
  - eg, in monoalphabetic cipher, trudy determines pairings for a,l,i,c,e,b,o,

- **Chosen-plaintext attack:** Trudy can get the ciphertext for some chosen plaintext
Types of Cryptography

- Crypto often uses keys:
  - Algorithm is known to everyone
  - Only “keys” are secret

- Symmetric key cryptography
  - Involves the use of one key

- Public key cryptography
  - Involves the use of two keys

- Hash functions
  - Involves the use of no keys
  - Nothing secret: How can this be useful?
Symmetric key cryptography

**symmetric key crypto:** Bob and Alice share same (symmetric) key: $K$

- 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 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
- \( m(i) = \text{ith bit of message} \)
- \( ks(i) = \text{ith bit of keystream} \)
- \( c(i) = \text{ith bit of ciphertext} \)
- \( c(i) = ks(i) \oplus m(i) \quad (\oplus = \text{exclusive or}) \)
- \( m(i) = ks(i) \oplus c(i) \)
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
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>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>110</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>001</td>
<td>111</td>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>010</td>
<td>101</td>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>111</td>
<td>001</td>
</tr>
</tbody>
</table>

What is the ciphertext for 010110001111 ?
Block ciphers

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

- 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
Prototype function

From Kaufman et al
Why rounds in prototype?

- If only a single round, then one bit of input affects at most 8 bits of output.
- In 2\textsuperscript{nd} 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
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 cyphertext.

- 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 \( c(i), r(i), i=1,2,\ldots \)
  - 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

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

- **cipher block chaining**: XOR ith input block, m(i), with previous block of cipher text, c(i-1)
  - c(0) transmitted to receiver in clear
  - what happens in “HTTP/1.1” scenario from above?
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

- 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
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
Types of Cryptography

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

$K_B^+(m)$

Bob's public key

Bob's private key

$m = K_B^-(K_B^+(m))$
Public key encryption algorithms

Requirements:

① need $K_B^+(\cdot) \text{ and } K_B^-(\cdot)$ 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

- 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 cyphertext).
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
   \[
   m = c^d \mod n
   \]

   Magic happens! \[
   m = (m^e \mod n)^d \mod n
   \]
RSA example:


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

Encrypting 8-bit messages.

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

<table>
<thead>
<tr>
<th>decrypt:</th>
<th>$c$</th>
<th>$c^d$</th>
<th>$m = c^d \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>17</td>
<td>481968572106750915091411825223071697</td>
<td>12</td>
</tr>
</tbody>
</table>

Encrypting 8-bit messages.
Why does RSA work?

- Must show that \( c^d \mod n = m \)
  where \( c = m^e \mod n \)

- Fact: for any \( x \) and \( y \): \( x^y \mod n = x^{(y \mod z)} \mod n \)
  - where \( n = pq \) and \( z = (p-1)(q-1) \)

- Thus,
  \[
  c^d \mod n = (m^e \mod n)^d \mod n
  = m^{ed} \mod n
  = m^{(ed \mod z)} \mod n
  = m^1 \mod n
  = m
  \]
RSA: another important property

The following property will be very useful later:

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

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.

Generating RSA keys

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

- Asymmetric crypto is expensive but simplify key exchange
- Symmetric crypto is cheap but tricky to securely establish the key
Session keys

- Exponentiation is computationally intensive
- DES is at least 100 times faster than RSA

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
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 signature"

- 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>
</tbody>
</table>

\[\text{B2 C1 D2 AC} \quad \text{different messages} \quad \text{but identical checksums!}\]
Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
Message Authentication Code (MAC)

- \( s = \text{shared secret} \)

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

- Popular MAC standard
- 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.
End-point authentication

- Want to be sure of the originator of the message - *end-point authentication*.

- Assuming Alice and Bob have a shared secret, will MAC provide end-point authentication.
  - We do know that Alice created the message.
  - But did she send it?
Playback attack

$\text{MAC} = f(\text{msg}, s)$

Transfer $\$1\text{M}$ from Bill to Trudy

Transfer $\$1\text{M}$ from Bill to Trudy
Defending against playback attack: nonce

MAC = f(msg, s, R)

Transfer $1M from Bill to Susan

MAC
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 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

$K_B^-(m)$

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

Bob sends digitally signed message:

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

Alice verifies signature and integrity of digitally signed message:

- Bob's public key $K_B^+$
- Large message $m$
- Hash function $H$: $H(m)$
- Encrypted msg digest $K_B^-(H(m))$
- Digital signature (decrypt) $K_B^+(H(m))$
- $H(m)$?
- Equal?
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.
Does digital signature guarantee authenticity?
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”

- Bob’s public key $K_B^+$
- Bob’s identifying information
- CA private key $K_{CA}^-$
- digital signature (encrypt)
- certificate for Bob’s public key, signed by CA
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
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”