Goals for Today

• Symmetric Key Encryption

• Public Key Encryption

• Certificate Authorities

• Secure Sockets Layer
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

  Encryption key: mapping from set of 26 letters to set of 26 letters
Stream and Block Ciphers

- n substitution ciphers, $M_1, M_2, \ldots, M_n$
- cycling pattern:
  - e.g., $n=4$: $M_1, M_3, M_4, M_3, M_2; \ M_1, M_3, M_4, M_3, M_2; \ldots$
  - random initialization
- 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
AES: Advanced Encryption Standard

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

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

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

plaintext message, $m$

$K^+(m)$

Bob’s public key

$K^-(m)$

Bob’s private key

$ciphertext$

$K_B^+(m)$

plaintext message

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

requirements:

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

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

**RSA**: Rivest, Shamir, Adelson algorithm
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

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

\[ m = (m^e \mod n)^d \mod n \]
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>00001000</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></td>
<td>481968572106750915091411825223071697</td>
</tr>
</tbody>
</table>
RSA: an important property

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

use public key first, followed by private key

use private key first, followed by public key

result is the same!
Why 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
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
**Digital signatures**

**simple digital signature for message m:**

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

In practice, this is done more efficiently on message digests
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

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 $x = H(m)$
TCP 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!
Widely used hash functions

- MD5 (RFC 1321) has known vulnerabilities
  - computes 128-bit message digest in 4-step process
- SHA-1 is widely used but is deprecated
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
  - Collision attack with 1000 GPUs in a month
- SHA-2 and SHA-3 are now available
  - Also standardized by NIST
  - More secure, but slower (in software)
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 showing the process of certifying a public key](image-url)
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

\[ K_B^+ \text{ digital signature (decrypt) } K_B^+ \]
Secure Sockets Layer

- SSL provides application programming interface (API) to applications
**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**
- **MIC**

Message Integrity Code (MIC) is a cryptographic hash. Data and MIC use **symmetric** encryption.
SSL cipher suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MIC 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
SSL overview

- **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
SSL: Setup ("handshake")

1. Server authentication
   - client sends list of algorithms it supports, along with client nonce (a random number, used only once)
   - server chooses algorithms from list; sends back: choice + certificate + server nonce

2. Crypto negotiation
   - client verifies certificate, extracts server’s public key
   - generates pre_master_secret, encrypts with server’s public key, sends to server

3. Establish keys
   - Client and server independently compute encryption and MIC keys from pre_master_secret and nonces

4. Authentication
   - client sends a MIC of all the handshake messages
   - server sends a MIC of all the handshake messages
SSL: handshake authentication

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
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”
- key block is then sliced and diced:
  - client MIC key
  - server MIC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)
SSL connection

handshake: ClientHello
handshake: ServerHello
handshake: Certificate
handshake: ServerHelloDone
handshake: ClientKeyExchange
ChangeCipherSpec
handshake: Finished
ChangeCipherSpec
handshake: Finished
application_data
application_data
Alert: warning, close_notify

everything henceforth is encrypted

TCP FIN follows