Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Public Key Crypto: Basic Problem

**Given:** Everybody knows Bob’s **public key**
Only Bob knows the corresponding **private key**

**Goals:**
1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself
Last Week

• Public key crypto protocols
  – Based on underlying assumptions about hard problems
  – Diffie Hellman and RSA
  – Not in this course: elliptic curves

• Last time: confidentiality (no integrity or authentication)
Digital Signatures: Basic Idea

**Given:** Everybody knows Bob’s public key
Only Bob knows the corresponding private key

**Goal:** Bob sends a “digitally signed” message
1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed
RSA Signatures

• Public key is \((n,e)\), private key is \((n,d)\)

• To sign message \(m\): \(s = m^d \mod n\)
  
  – Signing & decryption are same underlying operation in RSA
  
  – It’s infeasible to compute \(s\) on \(m\) if you don’t know \(d\)

• To verify signature \(s\) on message \(m\):
  
  verify that \(s^e \mod n = (m^d)^e \mod n = m\)

  – Just like encryption (for RSA primitive)
  
  – Anyone who knows \(n\) and \(e\) (public key) can verify signatures produced with \(d\) (private key)

• In practice, also need padding & hashing
  
  – Standard padding/hashing schemes exist for RSA signatures
DSS Signatures

• Digital Signature Standard (DSS)
• Public key: \((p, q, g, y=g^x \mod p)\), private key: \(x\)
• Security of DSS requires hardness of discrete log
  – If could solve discrete logarithm problem, would extract \(x\) (private key) from \(g^x \mod p\) (public key)
Cryptography Summary

• Goal: Privacy
  – Symmetric keys:
    • One-time pad, Stream ciphers
    • Block ciphers (e.g., DES, AES) \(\rightarrow\) modes: EBC, CBC, CTR
  – Public key crypto (e.g., Diffie-Hellman, RSA)

• Goal: Integrity
  – MACs, often using hash functions (e.g., MD5, SHA-256)

• Goal: Privacy and Integrity
  – Encrypt-then-MAC

• Goal: Authenticity
  – Digital signatures (e.g., RSA, DSS)
Problem: How does Alice know that the public key she received is really Bob’s public key?
Threat: Man-In-The-Middle (MITM)
Distribution of Public Keys

• Public announcement or public directory
  – Risks: forgery and tampering

• Public-key certificate
  – Signed statement specifying the key and identity
    • $\text{sig}_{\text{CA}}(\text{“Bob”}, \text{PK}_B)$

• Common approach: certificate authority (CA)
  – Single agency responsible for certifying public keys
  – After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  – Every computer is pre-configured with CA’s public key
Trusted(?) Certificate Authorities

![Keychain Access](image)

- Keychain Access: Click to unlock the System Roots keychain.
- Keychains:
  - login
  - Local Items
  - System
  - System Roots
- Category:
  - All Items
  - Passwords
  - Secure Notes
  - My Certificates
  - Keys
  - Certificates
- Apple Root CA
  - Root certificate authority
  - Expires: Friday, February 9, 2035 at 1:40:36 PM Pacific Standard Time
  - This certificate is valid
- Table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminCA-CD-T01</td>
<td>certificate</td>
<td>Jan 25, 2016, 4:36:19 AM</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:06:06 AM</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
</tr>
<tr>
<td>America Online...cation Authority 1</td>
<td>certificate</td>
<td>Nov 19, 2037, 12:43:00 PM</td>
</tr>
<tr>
<td>America Online...cation Authority 2</td>
<td>certificate</td>
<td>Sep 29, 2037, 7:08:00 AM</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>Feb 9, 2035, 1:40:36 PM</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:10:09 AM</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:19:06 AM</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>Feb 9, 2025, 4:18:14 AM</td>
</tr>
<tr>
<td>Application CA G2</td>
<td>certificate</td>
<td>Mar 31, 2016, 7:59:59 AM</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
</tr>
</tbody>
</table>
Hierarchical Approach

• Single CA certifying every public key is impractical

• Instead, use a trusted root authority (e.g., Verisign)
  – Everybody must know the root’s public key
  – Instead of single cert, use a certificate chain
    • \( \text{sig}_{\text{Verisign}}(\text{“AnotherCA”}, \text{PK}_{\text{AnotherCA}}), \text{sig}_{\text{AnotherCA}}(\text{“Alice”}, \text{PK}_A) \)

  – What happens if root authority is ever compromised?
You encounter this every day...

SSL/TLS: Encryption & authentication for connections
Example of a Certificate

<table>
<thead>
<tr>
<th>GeoTrust Global CA</th>
<th>Google Internet Authority G2</th>
<th>*.google.com</th>
</tr>
</thead>
</table>

***.google.com**

- **Issued by:** Google Internet Authority G2
- **Expires:** Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time
- **This certificate is valid**

**Details**

<table>
<thead>
<tr>
<th>Subject Name</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>California</td>
</tr>
<tr>
<td>State/Province</td>
<td>Mountain View</td>
</tr>
<tr>
<td>Locality</td>
<td>Google Inc</td>
</tr>
<tr>
<td>Organization</td>
<td>*.google.com</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issuer Name</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>Google Inc</td>
</tr>
<tr>
<td>Organization</td>
<td>Google Internet Authority G2</td>
</tr>
</tbody>
</table>

**Signature Algorithm**
- SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)

**Parameters**
- none

**Not Valid Before**
- Wednesday, April 8, 2015 at 6:40:10 AM Pacific Daylight Time

**Not Valid After**
- Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time

**Public Key Info**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Elliptic Curve Public Key (1.2.840.10045.2.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)</td>
</tr>
<tr>
<td>Public Key</td>
<td>65 bytes: 04 CB DD C1 CE AC D6 20 ...</td>
</tr>
<tr>
<td>Key Size</td>
<td>256 bits</td>
</tr>
<tr>
<td>Key Usage</td>
<td>Encrypt, Verify, Derive</td>
</tr>
<tr>
<td>Signature</td>
<td>256 bytes: 34 8B 7D 64 5A 64 08 5B ...</td>
</tr>
</tbody>
</table>
X.509 Certificate
Many Challenges...

• Hash collisions
• Weak security at CAs
  – Allows attackers to issue rogue certificates
• Users don’t notice when attacks happen
  – We’ll talk more about this later in the course
• Etc…
# Colliding Certificates

## Set by the CA

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Validity period</th>
<th>Real cert domain name</th>
<th>RSA key</th>
<th>X.509 extensions</th>
<th>Signature</th>
</tr>
</thead>
</table>

## Rogue Cert

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Validity period</th>
<th>Rogue cert domain name</th>
<th>RSA key</th>
<th>X.509 extensions</th>
<th>Signature</th>
</tr>
</thead>
</table>

- **Hash to the same MD5 value!**
  - **Valid for both certificates!**
  - **Identical bytes** (copied from real cert)
  - **Collision bits** (computed)

---

[Sotirov et al. “Rogue Certificates”]
Attacking CAs

Security of DigiNotar servers:

- All core certificate servers controlled by a single admin password (Prod@dm1n)
- Software on public-facing servers out of date, unpatched
- No anti-virus (could have detected attack)
Consequences

• Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then…
  – For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
• … “authenticate” as the real site
• … decrypt all data sent by users
  – Email, phone conversations, Web browsing
More Rogue CerTs

• In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  – TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  – Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network
• This rogue *.google.com certificate was trusted by every browser in the world
Certificate Revocation

• Revocation is very important
• Many valid reasons to revoke a certificate
  – Private key corresponding to the certified public key has been compromised
  – User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  – CA’s private key has been compromised!
• Expiration is a form of revocation, too
  – Many deployed systems don’t bother with revocation
  – Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

• Certificate revocation list (CRL)
  – CA periodically issues a signed list of revoked certificates
    • Credit card companies used to issue thick books of canceled credit card numbers
  – Can issue a “delta CRL” containing only updates

• Online revocation service
  – When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    • Like a merchant dialing up the credit card processor
Attempt to Fix CA Problems: Certificate Pinning

- **Trust on first access**: tells browser how to act on subsequent connections
- **HPKP – HTTP Public Key Pinning**
  - Use these keys!
  - HTTP response header field "Public-Key-Pins"
- **HSTS – HTTP Strict Transport Security**
  - Only access server via HTTPS
  - HTTP response header field "Strict-Transport-Security"
Attempt to Fix CA Problems:

Certificate Transparency

- **Problem**: browsers will think nothing is wrong with a rogue certificate

- **Goal**: make it impossible for a CA to issue a bad certificate for a domain *without the owner of that domain knowing*
  - (Then what?)

- **Approach**: auditable certificate logs

www.certificate-transparency.org
Keys for People: Keybase

• Basic idea:
  – Rely on existing trust of a person’s ownership of other accounts (e.g., Twitter, GitHub, website)
  – Each user publishes signed proofs to their linked account

Franzi Roesner
@franziroesner

Verifying myself: I am franziroesner on Keybase.io. 5YGG83pd-i4zvxxl2dDUHDMrOouRG386Q_tZ / keybase.io/franziroesner/…

https://keybase.io/