Today’s Lecture

• Threat model for distributed systems
  • Basic security requirements

• Cryptographic toolkit (Hash, Digital Signature, …)
  • Classes of attacks

• Bootstrapping trust: Certificate Authorities

• Use Case: SSL / HTTPS
Distributed Systems Threat Model

- Different nodes interact using the network
  - Opens up a host of security challenges

- Attackers may eavesdrop, modify, or drop your packets!
Internet’s Design: Insecure

- Designed for simplicity in a naïve era
- “On by default” design
- Readily available zombie machines
- Attacks look like normal traffic
- Internet’s federated operation obstructs cooperation for diagnosis/mitigation
Security Challenges

• **Availability**: Will the network deliver data?
  – Infrastructure compromise, DDoS
• **Authentication**: Who is this actor?
  – Spoofing, phishing
• **Integrity**: Do messages arrive in original form?
• **Confidentiality**: Can adversary read the data?
  – Sniffing, man-in-the-middle
• **Provenance**: Who is responsible for this data?
  – Forging responses, denying responsibility
  – Not who sent the data, but who created it
Eavesdropping - Message Interception (Attack on Confidentiality)

- Unauthorized access to information
- Packet sniffers and wiretappers
- Illicit copying of files and programs

![Diagram showing the flow from A to B with an eavesdropper]

Eavesdropper
Eavesdropping Attack: Example

• **tcpdump with promiscuous network interface**
  – On a switched network, what can you see?

• **What might the following traffic types reveal about communications?**
  – DNS lookups (and replies)
  – IP packets without payloads (headers only)
  – Payloads
Integrity Attack - Tampering

- Stop the flow of the message
- Delay and optionally modify the message
- Release the message again
Authenticity Attack - Masquerading

• Unauthorized assumption of other’s identity
• Generate and distribute objects under this identity

Masquerader: from A
Attack on Availability

• Destroy hardware (cutting fiber) or software
• Modify software in a subtle way
• Corrupt packets in transit

• Blatant *denial of service* (DoS):
  – Crashing the server
  – Overwhelm the server (use up its resource)
Basic Forms of Cryptography
Confidentiality through Cryptography

• **Cryptography:** *communication over insecure channel in the presence of adversaries*

• Studied for thousands of years

• **Central goal:** how to encode information so that an adversary can’t extract it …but a friend can

• **General premise:** a *key* is required for decoding
  – Give it to friends, keep it away from attackers

• **Two different categories of encryption**
  – Symmetric: efficient, requires key distribution
  – Asymmetric (Public Key): computationally expensive, but no key distribution problem
Symmetric Key Encryption

• **Same key for encryption and decryption**
  – Both sender and receiver know key
  – But adversary does not know key

• **For communication, problem is key distribution**
  – How do the parties (secretly) agree on the key?

• **What can you do with a huge key? One-time pad**
  – Huge key of random bits

• **To encrypt/decrypt: just XOR with the key!**
  – Provably secure! …. provided:
    • You never reuse the key … and it really is random/unpredictable
  – Spies actually use these
Using Symmetric Keys

- Both the sender and the receiver use the same secret keys.
Asymmetric Encryption (Public Key)

• Idea: use two *different* keys, one to encrypt \( e \) and one to decrypt \( d \)
  – A key pair

• Crucial property: knowing \( e \) does not give away \( d \)

• Therefore \( e \) can be public: everyone knows it!

• If Alice wants to send to Bob, she fetches Bob’s public key (say from Bob’s home page) and encrypts with it
  – Alice can’t decrypt what she’s sending to Bob …
  – … but then, *neither can anyone else* (except Bob)
Public Key / Asymmetric Encryption

• **Sender uses receiver’s public key**
  - Advertised to everyone

• **Receiver uses complementary private key**
  - Must be kept secret

```plaintext
Plaintext

Encrypt with public key

Internet

Ciphertext

Decrypt with private key

Plaintext
```
Works in Reverse Direction Too!

- Sender uses his own **private** key
- Receiver uses complementary **public** key
- Allows sender to prove he knows private key
Realizing Public Key Cryptography

• Invented in the 1970s
  – Revolutionized cryptography
  – (Was actually invented earlier by British intelligence)

• How can we construct an encryption/decryption algorithm with public/private properties?
  – Answer: Number Theory

• Most fully developed approach: RSA
  – Rivest / Shamir / Adleman, 1977; RFC 3447
  – Based on modular multiplication of very large integers
  – Very widely used (e.g., SSL/TLS for https)
• **RSA:**
  – assumes it is **difficult to factor** a large integer with two large prime factors

• **Elliptic Curve:**
  – **Difficult to calculate the discrete logarithm** of a random elliptic curve in a finite field
Symmetric V. Asymmetric Encryption

• **Symmetric**
  - Only one key
  - Fast to compute
  - hard to distribute
    - When shared everyone knows
    - Need different keys for different “friends”

• **Asymmetric**
  - Two keys
  - Slow to compute
    - 100-1000X slower than symmetric
  - easy to distribute
    - distribute public key
    - keep private safe
- public key: AES
- shared key: DES, Triple-DES, TEA, IDEA
- hash functions: MD5, SHA-1
Attacks On Encryption

• **Bruteforce attack**
  • Try every possible key
  • Exponential time ($2^N - 1$) for N bit key

• **Side channel attack (information leakage)**
  • Infer bits in key by observing time or heat used during key computations
Cryptographic Toolkit
Cryptographic Toolkit

- **Confidentiality**: Encryption
- **Integrity**: ?
- **Authentication**: ?
- **Provenance**: ?
Integrity: Cryptographic Hashes

- Sender computes a digest of message $m$, i.e., $H(m)$
  – $H()$ is a publicly known hash function

- Send $m$ in any manner

- Send digest $d = H(m)$ to receiver in a secure way:
  – Using another physical channel
  – Using encryption (why does this help?)

- Upon receiving $m$ and $d$, receiver re-computes $H(m)$ to see whether result agrees with $d$
Operation of Hashing for Integrity

Plaintext → Digest (MD5)

Internet

Digest (MD5) → digest

Digest (MD5) → digested

= NO

corrupted msg

Plaintext

Plaintext
Cryptographically Strong Hashes

• Hard to find **collisions**
  – Adversary can’t find two inputs that produce same hash
  – Someone cannot alter message without modifying digest
  – Can succinctly refer to large objects

• Hard to **invert**
  – Given hash, adversary can’t find input that produces it
  – Can refer obliquely to private objects (e.g., passwords)
    • Send hash of object rather than object itself
Effects of Cryptographic Hashing

<table>
<thead>
<tr>
<th>Input</th>
<th>Hash sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>DFCD3454 BBEA788A 751A696C 24D97009 CA992D17</td>
</tr>
<tr>
<td>The red fox <strong>run</strong>s across the ice</td>
<td>52ED879E 70F71D92 6EB69570 08E03CE4 CA6945D3</td>
</tr>
<tr>
<td>The red fox <strong>walk</strong>s across the ice</td>
<td>46042841 935C7FB0 9158585A B94AE214 26EB3CEA</td>
</tr>
</tbody>
</table>
Attacks on Digital Signatures

• Pigeon hole principle
  • Hash \( M = M' \)
  • \( M >>> M' \) —> there will be collisions
    • # of bits in \( M \) is larger than in \( M' \)

• Birthday Attack
  • 'Easy' to find two people with the same birthday
  • Only need 23 people
  • ‘Easy’ to make two documents with same hash
Google just cracked one of the building blocks of web encryption (but don’t worry)

It’s all over for SHA-1

By Russell Brandom  russellbrandom  Feb 23, 2017, 11:49am EST
Cryptographic Toolkit

- **Confidentiality**: Encryption
- **Integrity**: Cryptographic Hash
- **Authentication**: ?
- **Provenance**: ?
Public Key Authentication

- Assumption: only you know your private key

- Each side need only to know the other side’s public key
  - No secret key need be shared

- A encrypts a nonce (random number) \( x \) using B’s public key

- B proves it can recover \( x \)
  - Thus provides it has B’s private key
  - In turn providing that it’s B

- A can authenticate itself to B in the same way
Cryptographic Toolkit

- **Confidentiality**: Encryption
- **Integrity**: Cryptographic Hash
- **Authentication**: Decrypting nonce
- **Provenance**: ?
Digital Signatures

- Suppose Alice has published public key $K_E$

- If she wishes to prove who she is, she can send a message $x$ encrypted with her private key $K_D$
  - Therefore: anyone w/ public key $K_E$ can recover $x$, verify that Alice must have sent the message
  - It provides a digital signature
  - Alice can’t deny later deny it $\Rightarrow$ non-repudiation
<table>
<thead>
<tr>
<th>Key size/hash size (bits)</th>
<th>PRB optimized 90 MHz Pentium 1 (Mbytes/s)</th>
<th>Crypto++ 2.1 GHz Pentium 4 (Mbytes/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEA</td>
<td>128</td>
<td>23.801</td>
</tr>
<tr>
<td>DES</td>
<td>56</td>
<td>21.340</td>
</tr>
<tr>
<td>Triple-DES</td>
<td>112</td>
<td>9.848</td>
</tr>
<tr>
<td>IDEA</td>
<td>128</td>
<td>18.963</td>
</tr>
<tr>
<td>AES</td>
<td>128</td>
<td>61.010</td>
</tr>
<tr>
<td>AES</td>
<td>192</td>
<td>53.145</td>
</tr>
<tr>
<td>AES</td>
<td>256</td>
<td>48.229</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>216.674</td>
</tr>
<tr>
<td>SHA-1</td>
<td>160</td>
<td>67.977</td>
</tr>
</tbody>
</table>

- public key: AES
- shared key: DES, Triple-DES, TEA, IDEA
- hash functions: MD5, SHA-1
Digital Signatures

- Digital signature = encrypted hash of msg
- Faster to compute than encrypted msg
- Provides similar level of provenance
Summary of Our Crypto Toolkit

• If we can securely distribute a key, then
  – Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality

• Public key cryptography does away with problem of secure key distribution
  – But not as computationally efficient
  – Often addressed by using public key crypto to exchange a session key
  – And not guaranteed secure
    • but major result if not
Summary of Our Crypto Toolkit, con’t

• Cryptographically strong hash functions provide major building block for integrity (e.g., SHA-1)
  – As well as providing concise digests
  – And providing a way to prove you know something (e.g., passwords) without revealing it (non-invertibility)
  – But: worrisome recent results regarding their strength

• Public key also gives us signatures
  – Including sender non-repudiation
PKIs and HTTPS
Public Key Infrastructure (PKI)

- Public key crypto is very powerful …
- … but the realities of tying public keys to real world identities turn out to be quite hard

- PKI: Trust distribution mechanism
  - Authentication via Digital Certificates
- Trust doesn’t mean someone is honest, just that they are who they say they are…
Managing Trust

- The most solid level of trust is rooted in our direct personal experience
  - E.g., Alice’s trust that Bob is who they say they are
  - Clearly doesn’t scale to a global network!

- In its absence, we rely on *delegation*
  - Alice trusts Bob’s identity because Charlie attests to it
  ... and Alice trusts Charlie
Managing Trust, con’t

• Trust is not particularly transitive
  – Should Alice trust Bob because she trusts Charlie …
  – … and Charlie vouches for Donna …
  – … and Donna says Eve is trustworthy …
  – … and Eve vouches for Bob’s identity?

• Two models of delegating trust
  – Rely on your set of friends and their friends
    • “Web of trust” -- e.g., PGP
  – Rely on trusted, well-known authorities (and their minions)
    • “Trusted root” -- e.g., HTTPS
PKI Conceptual Framework

• Trusted-Root PKI:
  – Basis: well-known public key serves as root of a hierarchy
  – Managed by a Certificate Authority (CA)

• To publish a public key, ask the CA to digitally sign a statement indicating that they agree (“certify”) that it is indeed your key
  – This is a certificate for your key (certificate = bunch of bits)
    • Includes both your public key and the signed statement
  – Anyone can verify the signature

• Delegation of trust to the CA
  – They’d better not screw up (duped into signing bogus key)
  – They’d better have procedures for dealing with stolen keys
  – Note: can build up a hierarchy of signing
Putting It All Together: HTTPS

- **Steps after clicking on** [https://www.amazon.com](https://www.amazon.com)

- **https** = “Use HTTP over SSL/TLS”
  - SSL = Secure Socket Layer
  - TLS = Transport Layer Security
    - Successor to SSL, and compatible with it
    - RFC 4346

- **Provides security layer (authentication, encryption) on top of TCP**
  - Fairly transparent to the app
HTTPS Connection (SSL/TLS), con’t

- Browser (client) connects via TCP to Amazon’s HTTPS server

- Client sends over list of crypto protocols it supports

- Server picks protocols to use for this session

- Server sends over its certificate

- (all of this is in the clear)
Inside the Server’s Certificate

- Name associated with cert (e.g., Amazon)
- Amazon’s public key
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- URL to revocation center to check for revoked keys
- Name of certificate’s signatory (who signed it)
- A public-key signature of a hash (MD5) of all this
  - Constructed using the signatory’s private RSA key
Validating Amazon’s Identity

• Browser retrieves cert belonging to the signatory
  – These are hardwired into the browser

• If it can’t find the cert, then warns the user that site has not been verified
  – And may ask whether to continue
  – Note, can still proceed, just without authentication

• Browser uses public key in signatory’s cert to decrypt signature
  – Compares with its own MD5 hash of Amazon’s cert
  – So public key of certificate authority

• Assuming signature matches, now have high confidence it’s indeed Amazon …
  – … assuming signatory is trustworthy
**HTTPS Connection (SSL/TLS), con’t**

- Browser constructs a random *session key* $K$
- Browser encrypts $K$ using Amazon’s public key
- Browser sends $E(K, KA_{\text{public}})$ to server
- Browser displays
- All subsequent communication encrypted w/ symmetric cipher using key $K$
  - E.g., client can authenticate using a password

![Diagram showing the HTTPS connection process](image-url)
Denial of Service Attacks
Denial of Service Attack

• Prevent other people from using a service:
  – A server
  – A link in a network

• High level idea
  – Sent a lot of packets and ensure 100% utilization
    • No one else can use it.
Problems with DoS

• One person attacks one server/link
  – Easy to figure out who ....
  – Easy to block ....
  – Takes a while for the attack to work.....
  – To attack Google must have as much resource as Google
Enter Distributed DoS
Distributed Denial of Service Attack

• Take over a number of machines
  – Use a BotNet

• Use all machines to conduct a DoS on a server
  – Much more effective than regular DoS
  – Harder to stop and shutdown
DNS Amplification Attack

DNS Amplification attack: (\times 40 \text{ amplification})

580,000 open resolvers on Internet (Kaminsky-Shiffman’06)
DDOS

![Diagram of DDOS concept]

- **BotNet**
- **prevention of IP spoofing**
- **preventing open amplifiers**
- **disabling DNS requests**

*DDOS* stands for Distributed Denial of Service. It involves overwhelming a network with traffic to prevent legitimate users from accessing it. The diagram illustrates how a BotNet can be used to generate a flood of DNS requests, potentially leading to a denial of service for the victim's network.