Security in Computer Networks

- **Goal:** we want to enable secure communication between two parties in a distributed system.

  ![Security Diagram]

- This requires implementing three related security functions:
  - **Authentication:**
    1. ensuring that a message is genuine and came from the stated source
    2. verifying the identity of an individual, such as a person at a remote terminal or the sender of a message.
  - **Data integrity:**
    The property that data has not been altered or destroyed in an unauthorized manner.
  - **Confidentiality (secrecy):**
    The property that information is not been made available or disclosed to unauthorized individuals, entities, or processes.

This generally requires the use of **cryptographic protocols**

Cryptographic functions

- **Secret key:**
  Use a single key to encrypt the plaintext and decrypt the ciphertext
  - requires that sender and receiver share the secret key

- **Public key:**
  Use different keys for encryption and decryption, of which one is private, and the other public

- **Hashing:**
  Just use a hash function on the plaintext and send it off
  - there’s no decryption at all, just verification

One-Time Pads

- **Simple idea:** Choose a random bit string, as long as the plaintext, and simply XOR it to get the ciphertext.

  - It can never be broken because the ciphertext has no information in it at all.

  - **Example:**
    1. Alice wants to send message \( m = 101010 \)
    2. Alice and Bob share a key \( k = 011010 \)
    3. Alice sends (and Trudy intercepts) \( m' = m \oplus k = 110000 \)

  - **What can Trudy learn from** \( m' = 110000 \)? **Nothing!**
    - either \( m_0 = 1 \), if \( k_0 = 0 \)
    - or \( m_0 = 0 \), if \( k_0 = 1 \)

  - **Downsides:**
    - They cannot be memorized
    - The length of the transmitted data is limited by the key length
    - Requires strict synchronization between sender and receiver: a single missed bit will screw up everything.
Symmetric-Key Algorithms

DES: Data Encryption Standard

- Symmetric-key algorithms use the same key for encryption and decryption
- One of the former and most widely used (albeit insecure nowadays) was the Data Encryption Standard (DES)

- Each iteration $i$ uses a different key $K_i$. The complexity lies in the mangler function $f$.
- The keys $K_i$ are derived from the initial 56-bit key.
- The real problem is the 56-bit key: it's too easy to break.
- Many variants exist: Triple DES, AES, ...

Public-Key Algorithms

RSA: Rivest, Shamit, Adleman

- The idea is to use a different key to encrypt and decrypt
  1. Choose two large primes $p$ and $q$ (usually $p \cdot q$ should be 1,024 bit)
  2. Compute $n = p \times q$ and $z = (p - 1) \times (q - 1)$
  3. Choose a number $e$ relatively prime to $z$
  4. Find $d$ such that $ed - 1$ is divisible by $24$
- The Bob's public key $K_B^+$ is the pair of number $(n, e)$; his private key $K_B^-$ is the pair of number $(n, d)$

   **Example**
   - Bob chooses $p = 5$ and $q = 7$ $\Rightarrow$ $n = 35$ and $z = 24$
   - Bob chooses $e = 5$ since 24 and 5 have no common factors
   - Finally Bob chooses $d = 29$ since $5 \cdot 29 - 1$ is divisible by $24$

- Why Does RSA work?
  - It can be shown that if $\gcd(a, n) = 1$, then $a^{\phi(n)-1} \mod n = 1$

Digital Signatures

- What we often really need is to authenticate a message, and assure its integrity:
  1. Receiver can verify the claimed identity of the sender
  2. The sender can later not deny that he/she sent the message
  3. The receiver can not tamper the message itself.
- The solution is to digitally sign the message.
- This means:
  - have the sender put a signature that can be verified
  - be sure that the signature cannot be faked, i.e. it should be uniquely associated with the message.
Symmetric-Key Signatures

- **Basic idea**: just use a Big Brother who passes the message, but signed, to the destination:
  1. Alice sends \([A, K_A(B, R_A, t, P)]\) to Big Brother.
  2. Big Brother signs \([A, t, P]\) and sends it along with the original message, encrypted with Bob’s secret key:\n     \([K_B(A, R_A, t, P, K_{BB}(A, t, P))]\).
- Using \(R_A\) and timestamps helps against replays.

**Question**

Why is signing by Big Brother necessary? This way, Alice cannot repudiate the message.

Public-Key Signatures

1. Alice encrypts her message \(P\) with her private key \(D_A\):
   \(P_A = D_A(P)\)
2. She then encrypts \(P_A\) with Bob’s public key \(E_A\): \(E_A(P_A)\), and sends it off.
3. Bob decrypts the incoming message with his private key \(D_B\). We know for sure that no one else has been able to read \(P_A\) during its transmission.
4. Bob decrypts the message with Alice’s public key \(E_A\), now knowing that it came from Alice.

**Message Digests**

- **Idea**: take an arbitrary length message, and compute a unique, fixed-length number from it.
- **Properties**:
  - Computing the hash \(h(m)\) for any message \(m\) is relatively easy.
  - Given a hash value \(h(m)\), the only way of getting \(m\) is to enumerate over all possible messages. In other words, \(h^{-1}\) is almost impossible to find.
  - It is computationally infeasible to find two messages \(m_1\) and \(m_2\) such that \(h(m_1) = h(m_2)\).
- **Used for**:
  - **password hashing** (store hash values for comparison instead of cleartext passwords)
  - **message fingerprinting** (add a message digest to the message to safeguard against changes)
  - **signatures** (sign the message digest instead of the entire message).

**Message Digests: Signatures**

- **Problem**: Don’t mix authentication and secrecy.
  - Instead, it should also be possible to send a message in the clear, but have it signed as well.
- **Solution**: take a message digest, and sign that:

  ![Diagram of message digestion and signature]
Public-Key Management

Problem: If two parties don't know each other, how can they get a hold of each other's public key and be certain that it's the right key?

Solution: Introduce a trusted third party that signs public keys by means of a certificate.

The standard for certificate is called X.509

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Which version of X.509</td>
</tr>
<tr>
<td>Serial number</td>
<td>This number plus the CA's name uniquely identifies the certificate</td>
</tr>
<tr>
<td>Signature algorithm</td>
<td>The algorithm used to sign the certificate</td>
</tr>
<tr>
<td>Issuer</td>
<td>X.500 name of the CA</td>
</tr>
<tr>
<td>Validity period</td>
<td>The starting and ending times of the validity period</td>
</tr>
<tr>
<td>Subject name</td>
<td>The entity whose key is being certified</td>
</tr>
<tr>
<td>Public key</td>
<td>The subject's public key and the ID of the algorithm using it</td>
</tr>
<tr>
<td>Issuer ID</td>
<td>An optional ID uniquely identifying the certificate's issuer</td>
</tr>
<tr>
<td>Subject ID</td>
<td>An optional ID uniquely identifying the certificate's subject</td>
</tr>
<tr>
<td>Extensions</td>
<td>Many extensions have been defined</td>
</tr>
<tr>
<td>Signature</td>
<td>The certificate's signature (signed by the CA's private key)</td>
</tr>
</tbody>
</table>

Public-Key Infrastructures (PKI)

Issue: We can't have just a single CA; we probably want several to distribute the work.

The solution is simple: build a hierarchy (and cache certificates):

This implies building a chain of trust

users accept certificate which have been signed by CA which in turn has been signed by upper-level CA

signed certificates are transmitted together so the check can be done locally

there are 100 root CA, whose keys are hardcoded in web browsers

certificates have an expiration time but can be also revoked

Firewalls

Essence: Sometimes it's better to select service requests at the lowest level: network packets.

packets that do not fit certain requirements are simply removed

e.g., source IP address, destination TCP port, …

Solution: Protect your company by a firewall: it implements access control

Virtual Private Networks

Issue: Build your own private network that can span several different locations, for example, building IPSec tunnels between firewalls:

IPSec is an enhancement to IP to send packets securely over the Internet
Wireless Security

- **802.11 (WEP): Wired Equivalent (?) Privacy**
  - provides authentication and data encryption between a host and a wireless access point using a symmetric key approach
  - unfortunately, the protocol is heavily flawed; the encryption algorithm has been broken
  - It took only two hours for two students to build the software to eavesdrop on an industry 802.11 network

- **WPA: Wi-Fi Protected Access**
  - created in response to several serious weaknesses found in WEP
  - It supports authentication, key management and stronger encryption (AES)
  - not broken... yet

- **Bluetooth:**
  - applied to different layers of the protocol stack
  - frequency hopping
  - passkey to establish connection (keys may be hardcoded)
  - data encryption
  - vulnerable to worm

### Authentication

The whole business of security is that we can ensure authorized access to resources.

- in practice, this means that we pay a lot of attention to authentication first

#### Question

- What's the difference between authentication and authorization? The former tells who you are, the latter what you can do.

- A stronger version of authentication is **nonrepudiation**: it is not possible for someone to deny that they sent a message.

#### Question

- How can we safeguard against repudiation? Just use digital signatures and insist on their usage.

### Authentication versus Integrity

- Authentication and data integrity rely on each other:
  - consider an active attack by an enemy X on the communication from A to B.
  - **Authentication without integrity:**
    - A's message is authenticated, and intercepted by X, who tampers with its content, but leaves the authentication part as is.
    - B will conclude the message came from A → it came from X, so authentication fails.
  - **Integrity without authentication:**
    - X intercepts a message from A, and then makes B believe that the content was really sent by X
    - the data has now been “changed” in an unauthorized manner, so integrity is violated.
    → integrity is meaningless if you don’t know the source of information.

#### Question

- What can we say about confidentiality versus authentication and integrity? No relationship: confidentiality is orthogonal to the other two

### Authentication Protocols

#### Secret Keys

1. Alice sends ID to Bob
2. Bob sends challenge $R_B$ (i.e. a random number) to Alice
3. Alice encrypts $R_B$ with shared key $K_{AB}$. Now Bob knows he's talking to Alice
4. Alice send challenge $R_A$ to Bob
5. Bob encrypts $R_A$ with $K_{AB}$. Now Alice knows she's talking to Bob

#### Question

- That's so inefficient...let's combine steps 1 & 4, and 2 & 5 Really?
1. Trudy claims she is Alice, and sends challenge $R_T$
2. Bob sends back a challenge $R_B$ and the encrypted $R_T$
3. Trudy starts a second session, claiming she is Alice, but uses challenge $R_B$
4. Bob sends back a challenge, plus $\{R_B\}_{K_{AB}}$
5. Trudy sends back $\{R_B\}_{K_{AB}}$ for the first session to prove she is Alice

Establishing a Shared Key: Diffie-Hellman

- Alice and Bob have to agree on two large prime numbers, $n$ and $g$
- both numbers may be public.
- Alice chooses large number $x$, and keeps it to herself
- Bob does the same, say $y$.

1. Alice sends $(n, g^x \mod n)$ to Bob
2. Bob sends $(g^y \mod n)$ to Alice
3. Alice computes $K_{AB} = (g^y \mod n)^x = g^{xy} \mod n$
4. Bob computes $K_{AB} = (g^x \mod n)^y = g^{xy} \mod n$

Bucket-Brigade Attack

- **Problem**: Diffie Hellman works fine, but there is no way that Bob knows for sure he’s getting information from Alice.
- Here comes Trudy again:

- Also called man-in-the-middle attack

Authentication Protocols

Public Key

1. Alice sends a challenge $R_A$ to Bob, encrypted with Bob’s public key $E_B$
2. Bob decrypts the message, proves he’s Bob (by sending $R_A$ back), and sends a challenge $R_B$ to Alice, along with a session key $K_S$.
   Everything’s encrypted with Alice’s public key $E_A$.
3. Alice proves she’s Alice by sending back the decrypted challenge, but now encrypted with the session key $K_S$. 

Notes
Pretty Good Privacy

PGP (Pretty Good Privacy) is essentially the brainchild of one person, Phil Zimmermann

- If privacy is outlawed, only outlaws will have privacy
- He was sued by the U.S. Government to have violated the export of munitions law because he put on the Internet

1. Calculate hash (MD5) of message, and encrypt that hash with Alice’s private key
   → you’ve got Alice’s signature.
2. Append signature to text, and compress it to P1.Z.
3. Encrypt P1.Z with IDEA, and send along key $K_M$, after encrypting it with Bob’s public key
   → Bob can get $K_M$ for decryption.

Pretty Good Privacy

Observations

- Expensive RSA is used only to encrypt two 128-bit messages. IDEA, which is much more efficient, is used for the hard stuff.
- Public keys are stored locally and can be retrieved in different ways.
  - for that reason, there is a value indicating the strength of the trust the holder has in that key
  - don’t use low-trusted keys for high-security messages.
- A user can maintain several private-public key pairs
  - this allows easy switching to another key pair when one is suspected to have been compromised.

Secure Sockets Layer

Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) are cryptographic protocols that provide secure end-to-end communications

- they are the basis of the https protocol
- They sit on top of the transport layer

(a) TCP session established
(b) Alice (server) sends a signed certificate to Bob (client) containing Alice’s public key
(c) Bob create a session key, encrypts with Alice’s public key and send its to Alice

- MAC are used for integrity checks
- sequence number are used to avoid replay attacks