Keeping communications secret

- **Encryption guarantees secrecy**

- **Symmetric encryption**
  - Encryption algorithm comprises two functions $E$ and $D$
  - To communicate secretly, parties share secret key $K$
  - Given message $M$, $E(K, M) \rightarrow C$, $D(K, C) \rightarrow M$
  - $M$ is plaintext, $C$ is ciphertext
  - Attacker cannot derive $M$ from $C$ without $K$

- **Most common algorithm type: Block cipher**
  - AES from Lab 4 is a block cipher
  - Operates on fixed-size blocks (e.g., 64 or 128 bits)
  - Maps plaintext blocks to same size ciphertext blocks
Example block cipher (blowfish)

- Derive $F$ and 18 subkeys from Key—$P_1 \ldots P_{18}$
- Divide plaintext block into two halves, $L_0$ and $R_0$
  - $R_i = L_{i-1} \oplus P_i$
  - $L_i = R_{i-1} \oplus F(R_i)$
  - $R_{17} = L_{16} \oplus P_{17}$
  - $L_{17} = R_{16} \oplus P_{18}$
- Output $L_{17}R_{17}$.

(Note: This is just to give an idea; it's not a complete description)
Problem: Integrity

- Attacker can tamper with messages
  - E.g., corrupt a block to flip a bit in next

- What if you delete original file after transfer?
  - Might have nothing but garbage at recipient

- Encryption does not guarantee integrity
  - A system that uses encryption alone (no integrity check) is often incorrectly designed.
  - Exception: Cryptographic storage like lab 4 (just protects against stolen or copied data)
Message authentication codes

- **Message authentication codes (MACs)**
  - Sender & receiver share secret key $K$
  - On message $m$, $\text{MAC}(K, m) \rightarrow v$
  - Attacker cannot produce valid $\langle m, v \rangle$ without $K$

- **To send message securely, append MAC**
  - Send $\{m, \text{MAC}(K, m)\}$, or encrypt $\{m, \text{MAC}(K, m)\}_{K'}$
  - Receiver of $\{m, v\}$ checks $v \overset{?}{=} \text{MAC}(K, m)$

- **Careful of Replay – don’t believe previous $\{m, v\}$**
Cryptographic hashes

- **Hash arbitrary-length input to fixed-size output**
  - Typical output size 128 or 160 bits
  - Cheap to compute on large input (faster than network)

- **Collision-resistant: Computationally infeasible to find** \( x \neq y, \ H(x) = H(y) \)
  - Many such collisions exist
  - No one has been able to find one, even after analyzing the alrogithm

- **Several hashes in common use** (SHA-1, MD5)
Applications of cryptographic hashes

• Small hash uniquely specifies large data
  - Hash a file, remember the hash value
  - Recompute hash later, if same value no tampering
  - Hashes often published for software distribution

• \( \text{HMAC}(K, m) = H(K \oplus \text{opad}, H(K \oplus \text{ipad}, m)) \)
  - \(H\) is a cryptographic hash like SHA-1
  - ipad is 0x36 repeated 64 times, opad 0x5c repeated 64 times
Public key encryption

- Three randomized algorithms:
  - Generate – $G(1^k) \rightarrow K, K^{-1}$
  - Encrypt – $E(K, m) \rightarrow \{m\}_K$
  - Decrypt – $D(K^{-1}, \{m\}_K) \rightarrow m$

- Provides secrecy, like conventional encryption
  - Can’t derive $m$ from $\{m\}_K$ without knowing $K^{-1}$

- Encryption key $K$ can be made public
  - Can’t derive $K^{-1}$ from $K$
  - Everyone can use the same public key to encrypt messages for one recipient.
Digital signatures

- Three (randomized) algorithms:
  - Generate – $G(1^k) \rightarrow K, K^{-1}$
  - Sign – $S(K^{-1}, m) \rightarrow \{m\}_{K^{-1}}$
  - Verify – $V(K, \{m\}_{K^{-1}}, m) \rightarrow \{\text{true, false}\}$

- Provides integrity, like a MAC
  - Cannot produce valid $\langle m, \{m\}_{K^{-1}} \rangle$ pair without $K^{-1}$

- Many keys support both signing & encryption
  - But Encrypt/Decrypt and Sign/Verify different algorithms!
  - Common error: Sign by “encrypting” with private key
Cost of cryptographic operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>msec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>1.11</td>
</tr>
<tr>
<td>Decrypt</td>
<td>39.62</td>
</tr>
<tr>
<td>Sign</td>
<td>40.56</td>
</tr>
<tr>
<td>Verify</td>
<td>0.10</td>
</tr>
</tbody>
</table>

[1,280-bit Rabin-Williams keys on 550 MHz K6]

- **Cost of public key algorithms significant**
  - Encryption only on small messages (< size of key)
  - Signature cost relatively insensitive to message size

- **In contrast, symmetric algorithms much cheaper**
  - Symmetric can encrypt+MAC faster than 100Mbit/sec LAN
Hybrid schemes

- **Use public key to encrypt symmetric key**
  - Send message symmetrically encrypted:
    \[ K_S \leftarrow \{0, 1\}^{128}, \{msg\}_{K_S}, \{K_S\}_{K_{Pub}} \]

- **Use PK to negotiate secret session key**
  - E.g., Client sends server \( \{K_1, K_2, K_3, K_4\}_{K_P} \)
  - Client sends server: \( \{m_1\}_{K_1}, \text{MAC } (K_2, \{m_1\}_{K_1}) \)
  - Server sends client: \( \{m_2\}_{K_3}, \text{MAC } (K_4, \{m_2\}_{K_3}) \)
  - Note: Better to MAC encrypted data than vice versa

- **Often want mutual authentication (client & server)**
  - Or more complex, user(s), client, & server
Server authentication

- **An approach:** Use public key cryptography
  - Give client public key of server
  - Lets client authenticate secure channel to server

- **Problem:** Key management problem
  - How to get server’s public key?
  - How to know the key is really server’s?
Otherwise: Attacker impersonates server

- **Man-in-the-middle attack:**
  - Attacker emulates server when talking to client
  - Attacker emulates client when talking to server
  - Attacker passes most messages through unmodified
  - Attacker substitutes own public key for client’s & server’s
  - Attacker records secret data, or tampers to cause damage
Key management

• Put public keys in the phone book
  - How do you know you have the real phone book?
  - How is a program supposed to use phone book www.phonebook.com? (are you talking to real web server)

• Exchange keys with people in person

• “Web of trust” – get keys from friends you trust
Certification authorities

- Everybody trusts some certification authority
- Everybody knows authority’s public key
  - E.g., built into web browser

1. PubKey, $$$ to Certification Authority
2. Certificate from Certification Authority to Server
3. Connection request from Client to Server
4. PubKey, Certificate from Server to Client