Cryptography

• **Is:**
  ➤ A tremendous tool
  ➤ The basis for many security mechanisms

• **Is not:**
  ➤ The solution to all security problems
Cryptography

• Is:
  ➤ A tremendous tool
  ➤ The basis for many security mechanisms

• Is not:
  ➤ The solution to all security problems
  ➤ Reliable unless implemented and used properly
  ➤ Something you should try to invent yourself
Cryptography

• **Is:**
  ➤ A tremendous tool
  ➤ The basis for many security mechanisms

• **Is not:**
  ➤ The solution to all security problems
  ➤ Reliable unless implemented and used properly
  ➤ Something you should try to invent yourself
  ➤ Blockchain
This class: secure communication

- Authenticity: Parties cannot be impersonated
- Secrecy: No one else can read messages
- Integrity: messages cannot be modified
Attacker models

- Passive attacker: Eve only snoops on channel
- Active attacker: Eve can snoop, inject, block, tamper, etc.
In the real world (SSL/TLS)

➤ **Handshake Protocol:** Establish shared secret key using public-key cryptography

➤ **Record Layer:** Transmit data protected by symmetric-key cryptography (using negotiated key)
Outline

• Symmetric-key crypto
  ➤ Encryption
  ➤ Hash functions
  ➤ Message authentication code

• Asymmetric (public-key) crypto
  ➤ Encryption
  ➤ Digital signatures
Symmetric-key encryption

**Encryption:** (key, plaintext) $\rightarrow$ ciphertext

- $E_k(m) = c$

**Decryption:** (key, ciphertext) $\rightarrow$ plaintext

- $D_k(c) = m$
Symmetric-key encryption

- **One-time key**: used to encrypt one message
  - E.g., encrypted email, new key generate per email

- **Multi-use key**: used to encrypt multiple messages
  - E.g., SSL, same key used to encrypt many packets
Symmetric-key encryption

- **One-time key**: used to encrypt one message
  - E.g., encrypted email, new key generate per email

- **Multi-use key**: used to encrypt multiple messages
  - E.g., SSL, same key used to encrypt many packets
Symmetric-key encryption

- **One-time key**: used to encrypt one message
  - E.g., encrypted email, new key generate per email

- **Multi-use key**: used to encrypt multiple messages
  - E.g., SSL, same key used to encrypt many packets
Symmetric-key encryption

- **One-time key**: used to encrypt one message
  - E.g., encrypted email, new key generate per email

- **Multi-use key**: used to encrypt multiple messages
  - E.g., SSL, same key used to encrypt many packets

Need unique/random nonce
Encryption properties

• Encryption and decryption are inverse operations
  \[ D_k(E_k(m)) = m \]

• Secrecy: ciphertext reveals nothing about plaintext
  More formally: can’t distinguish which of two plaintexts were encrypted without key
First example: One Time Pad

Vernam (1917)

Encryption: \( c = E_k(m) = m \oplus k \)

Decryption: \( D_k(c) = \)
First example: One Time Pad

Vernam (1917)

Encryption: $c = E_k(m) = m \oplus k$

Decryption: $D_k(c) = c \oplus k$
First example: One Time Pad

Vernam (1917)

<table>
<thead>
<tr>
<th>Key</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Encryption:** $c = E_k(m) = m \oplus k$
- **Decryption:** $D_k(c) = c \oplus k = (m \oplus k) \oplus k = m$
OTP security

• Shannon (1949)
  ➤ Information theoretic security: without key, ciphertext reveals no “information” about plaintext

• Problems with OTP
  ➤ Can only use key once
  ➤ Key is as long as the message
Computational cryptography

• Want the size of the secret to be small
  ➤ If pre-arranged secret smaller than message, not all plaintexts equally probable — ciphertext reveals info about plaintext

• Modern cryptography based on idea that learning anything about plaintext from ciphertext is computationally difficult without secret
Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random key
Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random key

Examples: ChaCha, Salsa, Sosemanuk, etc.
Stream ciphers

• Problem: OTP key is as long as message

• Solution: Pseudo random key

\[ E_k(m) = \text{PRG}(k) \oplus m \]
Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random key

➤ Examples: ChaCha, Salsa, Sosemanuk, etc.

\[
E_k(m) = PRG(k) \oplus m
\]
Stream ciphers

- **Problem:** OTP key is as long as message
- **Solution:** Pseudo random key

$E_k(m) = \text{PRG}(k) \oplus m$

Examples: ChaCha, Salsa, Sosemanuk, etc.
Dangers in using stream ciphers

• Can we use a key more than once?

➤ E.g., $c_1 \leftarrow m_1 \oplus \text{PRG}(k)$

$\quad c_2 \leftarrow m_2 \oplus \text{PRG}(k)$

➤ A: yes, B: no
Dangers in using stream ciphers

• Can we use a key more than once?
  ➤ E.g., $c_1 \leftarrow m_1 \oplus \text{PRG}(k)$
    
    $c_2 \leftarrow m_2 \oplus \text{PRG}(k)$
  ➤ A: yes, B: no
  ➤ Eavesdropper does: $c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$
  ➤ Enough redundant information in English that:
    $m_1 \oplus m_2 \rightarrow m_1, m_2$
Block ciphers: crypto work horses

- Block ciphers operate on fixed-size blocks
  - E.g., 3DES: $|m| = |c| = 64$ bits, $|k| = 168$ bits
  - E.g., AES: $|m| = |c| = 128$ bits, $|k| = 128, 192, 256$

- A block cipher = permutation of fixed-size inputs
  - Each input mapped to exactly one output
How do they work?

key k

key expansion

k₁ → R(k₁, m) → R(k₂, m) → R(k₃, m) → R(kₙ, m) → c

R(k,m): round function

for 3DES (n=48), for AES-128 (n=10)
How do they work?
Challenges with block ciphers

• Block ciphers operate on single fixed-size block
Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?
  - Several modes of operation for longer messages
Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?
  - Several modes of operation for longer messages
- How do we deal with messages that are not block-aligned?
  - Must pad messages in a distinguishable way
Electronic Codebook (ECB) mode encryption

Source: wikipedia
Is ECB good? A: yes, B: no
Is ECB good? A: yes, B: no

\[ E_k(\text{Linux}) = \text{plaintext} \]

Source: wikipedia
CBC mode with random IV

Cipher Block Chaining (CBC) mode decryption

Source: wikipedia
CBC mode with random IV

Cipher Block Chaining (CBC) mode decryption

Subtle attacks that abuse padding possible!

Source: wikipedia
CTR mode with random IV

Counter (CTR) mode encryption

Source: wikipedia
CTR mode with random IV

Counter (CTR) mode encryption

Essentially use block cipher as stream cipher!

Source: wikipedia
What security do we actually get?

- All encryption breakable by brute force given enough knowledge about plaintext
- Try to decrypt ciphertext with every possible key until a valid plaintext is found
- Attack complexity proportional to size of key space
  - 64-bit key requires $2^{64}$ decryption attempts
Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication code

- Asymmetric (public-key) crypto
  - Encryption
  - Digital signatures
Hash Functions

- A (cryptographic) hash function maps arbitrary length input into a fixed-size string

\[ h = H(m) \]

- \(|m|\) is arbitrarily large
- \(|h|\) is fixed, usually 128-512 bits
Hash Function Properties

• Finding a pre-image is hard
  ➤ Given h, find m such that \( H(m) = h \)

• Finding a collision is hard
  ➤ Find \( m_1 \) and \( m_2 \) such that \( H(m_1) = H(m_2) \)
Hash Functions

• MD5: Message Digest
  ➢ Designed by Ron Rivest
  ➢ Very popular hash function
  ➢ Output: 128 bits
  ➢ Broken — do not use!
Hash Functions

• SHA-1: Secure Hash Algorithm 1
  ➢ Designed by NSA
  ➢ Output: 160 bits
  ➢ Broken — **do not use!**

• SHA-2: Secure Hash Algorithm 2
  ➢ Designed by NSA
  ➢ Output: 224, 256, 384, or 512 bits
  ➢ Recommended for use today
Hash Functions

- SHA-3: Secure Hash Algorithm 3
  - Result of NIST SHA-3 contest
  - Output: arbitrary size
  - Replacement once SHA-2 broken
Outline

• Symmetric-key crypto
  ➢ Encryption
  ➢ Hash functions
  ➢ Message authentication code

• Asymmetric (public-key) crypto
  ➢ Encryption
  ➢ Digital signatures
MAC constructions

- HMAC: MAC based on hash function

\[ MAC_k(m) = H( k\oplus opad \ || \ H( k\oplus ipad \ || \ m ) ) \]

- HMAC-SHA1: HMAC construction using SHA-1
- HMAC-SHA256: HMAC construction using SHA-256

- CMAC: MAC based on block cipher
MACs

- Validate message integrity based on shared secret

MAC: Message Authentication Code
- Keyed hash function using shared secret
- Hard compute hash without knowing key

\[ a = \text{MAC}_k(m) \]
Combining MAC with encryption

MAC then Encrypt (SSL)

➤ Integrity for plaintext not ciphertext

➤ Issue: need to decrypt before you can verify integrity

➤ Hard to get right!
Combining MAC with encryption

Encrypt and MAC (SSH)

➤ Integrity for plaintext not ciphertext
➤ Issue: need to decrypt before you can verify integrity
➤ Hard to get right!
Combining MAC with encryption

Encrypt then MAC (IPSec)

➤ Integrity for plaintext and ciphertext

➤ Always right!
AEAD construction

- Authenticated Encryption with Associated Data
  - AES-GCM
  - E.g., as used in Google’s Tink:

```java
import com.google.crypto.tink.Aead;
import com.google.crypto.tink.KeysetHandle;
import com.google.crypto.tink.aead.AeadKeyTemplates;

// 1. Generate the key material.
KeysetHandle keysetHandle = KeysetHandle.generateNew(
    AeadKeyTemplates.AES128_GCM);

// 2. Get the primitive.
Aead aead = keysetHandle.getPrimitive(Aead.class);

// 3. Use the primitive.
byte[] ciphertext = aead.encrypt(plaintext, associatedData);
```