Goals for Today

- Cryptography Background
- Symmetric (Shared-Key Foundations)

Basic Problem

- Basic Internet model: Communications through untrusted intermediaries.
- I know M (attack privacy)
- I can change M (attack integrity)
- Important for: Secure remote logins, file transfers, web access, ...

Symmetric Setting

- Solution: Encapsulate and decapsulate messages in some secure way.
- Symmetric setting: Both parties share some secret information, called a key.
Achieving Privacy

Encryption schemes

<table>
<thead>
<tr>
<th>Encrypt</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td>Ciphered Message</td>
<td>Ciphered Message</td>
</tr>
</tbody>
</table>

Adversary

History

- Substitution Ciphers
  - Caesar Cipher
- Transposition Ciphers
- Codebooks
- Machines

Recommended Reading: The Codebreakers by David Kahn.
- Military uses
- Rumrunners
- ...

Achieving Integrity

Message authentication schemes or message authentication codes or MACs

<table>
<thead>
<tr>
<th>MAC</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td>Tag</td>
<td>Tag</td>
</tr>
</tbody>
</table>

Adversary

Achieving Both Privacy and Integrity

Authenticated encryption schemes

(Authenticated encryption notion is “new” (around 2000), so many books and protocols don’t discuss this. Can be subtle?)

<table>
<thead>
<tr>
<th>Encrypt</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td>Ciphered Message</td>
<td>Ciphered Message</td>
</tr>
</tbody>
</table>

Adversary
How this is achieved

- Layered approach:
  - Cryptographic primitives, like block ciphers, stream ciphers, and hash functions
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication

- Today:
  - Start on the above. Basic concepts. Basic pitfalls.

Asymmetric Setting (NOT today)

Alice
PK_A, SK_A
PK_B

Bob
PK_B, SK_B
PK_A

Adversary

One-Time Pad

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{plaintext} = (\text{ciphertext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

Cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon)

Advantages of One-Time Pad

- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - ...as long as the key sequence is truly random
  - True randomness is expensive to obtain in large quantities
  - ...as long as each key is same length as plaintext
    - But how does the sender communicate the key to receiver?
Disadvantages

Disadvantage #1: Keys as long as messages.
Impractical in most scenarios
Still used by intelligence communities

Disadvantage #2: No integrity protection

Disadvantage #3: Keys cannot be reused

Learn relationship between plaintexts:
\[ C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = (P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2 \]

Reducing Keysize

- What do we do when we can’t pre-share huge keys?
  - When OTP is unrealistic
  - We use special cryptographic primitives
    - Single key can be reused (with some restrictions)
      - But no longer provable secure (in the sense of the OTP)
  - Examples: Block ciphers, stream ciphers
Background: Permutation

- For N-bit input, N! possible permutations
- Idea: split plaintext into blocks, for each block use secret key to pick a permutation, rinse and repeat
  - Without the key, permutation should "look random"

Block Ciphers

- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Same key is reused for each block (can use short keys)

Block Cipher Security

- Result should look like a random permutation
  - "As if" plaintext bits were randomly shuffled
- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption), then can only break by brute-force, try every possible key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

Block Cipher Operation (Simplified)

- Procedure must be reversible (for decryption)
Feistel Structure (Stallings Fig 2.2)

DES
- Feistel structure
  - “Ladder” structure: split input in half, put one half through the round and XOR with the other half
  - After 3 random rounds, ciphertext indistinguishable from a random permutation (Luby & Rackoff)
- DES: Data Encryption Standard
  - Feistel structure
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

DES and 56 bit keys (Stallings Tab 2.2)
- 56 bit keys are quite short

<table>
<thead>
<tr>
<th>Key size (bits)</th>
<th>Number of iterations</th>
<th>Time required at 1 encryption</th>
<th>Time required at 1000 encryptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>$2^{24} \times 10^{3}$</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>64</td>
<td>$2^{44} \times 10^{3}$</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>128</td>
<td>$2^{64} \times 10^{3}$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>256</td>
<td>$2^{128} \times 10^{3}$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1999: EFF DES Crack + distributed machines
- < 24 hours to find DES key
- DES ---> 3DES
- 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

Advanced Encryption Standard (AES)
- New federal standard as of 2001
- Based on the Rijndael algorithm
- 128-bit blocks, keys can be 128, 192 or 256 bits
- Unlike DES, does not use Feistel structure
  - The entire block is processed during each round
- Design uses some very nice mathematics
Basic Structure of Rijndael

- 128-bit plaintext (arranged as 4x4 array of 8-bit bytes)
- 128-bit key

- Byte substitution
- Shift rows: 1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3
- Mix columns: mix 4 bytes in each column (each new byte depends on all bytes in old column)
- Add key for this round

Encrypted a Large Message

- So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size
- Electronic Code Book (ECB) mode
  - Split plaintext into blocks, encrypt each one separately using the block cipher
- Cipher Block Chaining (CBC) mode
  - Split plaintext into blocks, XOR each block with the result of encrypting previous blocks
- Counter (CTR) mode
  - Use block cipher to generate keystream, like a stream cipher
- ...