Review: Kerckhoff’s Principle

- Security of a cryptographic object should depend only on the secrecy of the secret (private) key.
- Security should not depend on the secrecy of the algorithm itself.
Review: Flavors of Cryptography

• Symmetric cryptography
  – Both communicating parties have access to a shared random string $K$, called the key.

• Asymmetric cryptography
  – Each party creates a public key $pk$ and a secret key $sk$.
  – Hard concept to understand, and revolutionary! Inventors won the Turing Award 😊
Both communicating parties have access to a shared random string $K$, called the key.
Review: Asymmetric Setting

Each party creates a public key $pk$ and a secret key $sk$.
Confidentiality: Basic Problem

Given (Symmetric Crypto): both parties know the same secret.
Goal: send a message confidentially.

Ignore for now: How is this achieved in practice??
One weird bit-level trick

• XOR!
  – Just XOR with a random bit!

• Why?
  – Uniform output
  – Independent of ‘message’ bit
One-Time Pad

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key: ciphertext = plaintext \( \oplus \) key

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

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, 1949)
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 sender communicate the key to receiver?
Problems with the One-Time Pad?

- What potential security problems do you see with the one-time pad?
- Canvas -> Quizzes -> January 25
Dangers of Reuse

Learn relationship between plaintexts

\[ C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = (P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2 \]
Problems with One-Time Pad

• (1) Key must be as long as the plaintext
  – Impractical in most realistic scenarios
  – Still used for diplomatic and intelligence traffic

• (2) Insecure if keys are reused
  – Attacker can obtain XOR of plaintexts
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:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

\[ 00110010... \oplus 00110010... = 10111101... \]

\[ 10001111... \oplus 00110010... = 10111101... \]

\[ 00110010... = 00110010... \]

\[ 01011101... \]
Problems with One-Time Pad

• (1) Key must be as long as the plaintext
  – Impractical in most realistic scenarios
  – Still used for diplomatic and intelligence traffic

• (2) Insecure if keys are reused
  – Attacker can obtain XOR of plaintexts

• (3) Does not guarantee integrity
  – One-time pad only guarantees confidentiality
  – Attacker cannot recover plaintext, but can easily change it to something else
Reducing Key Size

• What to do when it is infeasible to pre-share huge random keys?
  – When one-time pad is unrealistic...

• Use special cryptographic primitives: block ciphers, stream ciphers
  – Single key can be re-used (with some restrictions)
  – Not as theoretically secure as one-time pad
Block Ciphers

• Operates on a single chunk ("block") of plaintext
  – For example, 64 bits for DES, 128 bits for AES
  – Each key defines a different permutation
  – Same key is reused for each block (can use short keys)
### Keyed Permutation

#### Possible Permutations

For **N**-bit input:

\[ 2^N! \text{ possible permutations} \]

For **K**-bit key:

\[ 2^K \text{ possible keys} \]

<table>
<thead>
<tr>
<th>input</th>
<th>possible output</th>
<th>possible output</th>
<th>etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>010</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>111</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>101</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>110</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>000</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>
Keyed Permutation

• Not just shuffling of input bits!
  – Suppose plaintext = “111”.
  – Then “111” is not the only possible ciphertext!

• Instead:
  – Permutation of possible outputs
  – Use secret key to pick a permutation
Block Cipher Security

• Result should “look like” a random permutation on the inputs

• 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)

Block of plaintext

Add some secret key bits to provide confusion

Each S-box transforms its input bits in a “random-looking” way to provide diffusion (spread plaintext bits throughout ciphertext)

repeat for several rounds

Procedure must be reversible (for decryption)

Block of ciphertext
Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity
**DES and 56 bit keys**

- 56 bit keys are quite short

<table>
<thead>
<tr>
<th>Key Size (bits)</th>
<th>Number of Alternative Keys</th>
<th>Time required at 1 encryption/μs</th>
<th>Time required at 10^6 encryptions/μs</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2^32 = 4.3 \times 10^9</td>
<td>2^{31} μs = 35.8 minutes</td>
<td>2.15 milliseconds</td>
</tr>
<tr>
<td>56</td>
<td>2^56 = 7.2 \times 10^{16}</td>
<td>2^{55} μs = 1142 years</td>
<td>10.01 hours</td>
</tr>
<tr>
<td>128</td>
<td>2^{128} = 3.4 \times 10^{38}</td>
<td>2^{127} μs = 5.4 \times 10^{24} years</td>
<td>5.4 \times 10^{18} years</td>
</tr>
<tr>
<td>168</td>
<td>2^{168} = 3.7 \times 10^{50}</td>
<td>2^{167} μs = 5.9 \times 10^{16} years</td>
<td>5.9 \times 10^{30} years</td>
</tr>
<tr>
<td>26 characters (permutation)</td>
<td>26! = 4 \times 10^{26}</td>
<td>2 \times 10^{26} μs = 6.4 \times 10^{12} years</td>
<td>6.4 \times 10^{6} years</td>
</tr>
</tbody>
</table>

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

• Two-key 3DES increases security of DES by doubling the key length

• Why not 2DES?
  – Minimal gain in security compared to DES due to meet-in-the-middle attack (more in section on this)
Standard Block Ciphers

• DES: Data Encryption Standard
  – Feistel structure: builds invertible function using non-invertible ones
  – Invented by IBM, issued as federal standard in 1977
  – 64-bit blocks, 56-bit key + 8 bits for parity

• AES: Advanced Encryption Standard
  – New federal standard as of 2001
    • NIST: National Institute of Standards & Technology
  – Based on the Rijndael algorithm
    • Selected via an open process
  – 128-bit blocks, keys can be 128, 192 or 256 bits
Encrypting a Large Message

• So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size

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

128-bit ciphertext

• What should we do?
Electronic Code Book (ECB) Mode

Canvas “quiz” time!
Electronic Code Book (ECB) Mode

- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks
Information Leakage in ECB Mode

Encrypt in ECB mode

[Wikipedia]
Oops

Move Fast and Roll Your Own Crypto
A Quick Look at the Confidentiality of Zoom Meetings

By Bill Marczak and John Scott-Railton  April 3, 2020

- Zoom documentation claims that the app uses “AES-256” encryption for meetings where possible. However, we find that in each Zoom meeting, a single AES-128 key is used in ECB mode by all participants to encrypt and decrypt audio and video. The use of ECB mode is not recommended because patterns present in the plaintext are preserved during encryption.

Cipher Block Chaining (CBC) Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
- Still does not guarantee integrity
CBC Mode: Decryption

plaintext

 Initialization vector

key

decrypt

ciphertext
ECB vs. CBC

AES in ECB mode

Similar plaintext blocks produce similar ciphertext blocks (not good!)

AES in CBC mode

[Picture due to Bart Preneel]
Initialization Vector Dangers

Found in the source code for Diebold voting machines:

```c
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data, totalSize, DESKEY, NULL, DES_ENCRYPT)
```
Counter Mode (CTR): Encryption

Initial ctr (random) →

- ctr
  - Key
  - block cipher
  - pt
  - \(\oplus\)
  - ciphertext

- ctr+1
  - Key
  - block cipher
  - pt
  - \(\oplus\)

- ctr+2
  - Key
  - block cipher
  - pt
  - \(\oplus\)

- ctr+3
  - Key
  - block cipher
  - pt
  - \(\oplus\)

- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity; Fragile if ctr repeats
Counter Mode (CTR): Decryption

Initial ctr

<table>
<thead>
<tr>
<th>ctr</th>
<th>ctr+1</th>
<th>ctr+2</th>
<th>ctr+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>block cipher</td>
<td>block cipher</td>
<td>block cipher</td>
<td>block cipher</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Key</td>
</tr>
</tbody>
</table>

\[
\text{Key} \oplus \text{ct} \rightarrow \text{block cipher} \rightarrow \text{ct} \rightarrow \text{pt}
\]
When is an Encryption Scheme “Secure”?

• Hard to recover the key?
  – What if attacker can learn plaintext without learning the key?

• Hard to recover plaintext from ciphertext?
  – What if attacker learns some bits or some function of bits?
How Can a Cipher Be Attacked?

- Attackers knows ciphertext and encryption algorithm
  - What else does the attacker know? Depends on the application in which the cipher is used!

- Ciphertext-only attack

- KPA: Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs

- CPA: Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of his choice

- CCA: Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext except the target
Chosen Plaintext Attack

Attacker #1 changes their PIN to a number of their choice

PIN is encrypted and transmitted to bank

cipher(key, PIN)

Attacker #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

... repeat for any PIN value
Very Informal Intuition

• Security against chosen-plaintext attack (CPA)
  – Ciphertext leaks no information about the plaintext
  – Even if the attacker correctly guesses the plaintext, they cannot verify their guess
  – Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
    • Implication: encryption must be randomized or stateful

• Security against chosen-ciphertext attack (CCA)
  – Integrity protection – it is not possible to change the plaintext by modifying the ciphertext