CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography
[Symmetric Encryption]

Spring 2019

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
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-Time Pad

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

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

- **One-time pad**: $\text{Ciphertext}(\text{Key, Message}) = \text{Message} \oplus \text{Key}
  - Key must be a random bit sequence as long as message
- **Idea**: replace “random” with “pseudo-random”
  - Use a pseudo-random number generator (PRNG)
  - PRNG takes a short, truly random secret seed and expands it into a long “random-looking” sequence
    - E.g., 128-bit seed into a $10^6$-bit pseudo-random sequence
- $\text{Ciphertext}(\text{Key, Msg}) = \text{Msg} \oplus \text{PRNG}(\text{Key})$
  - Message processed bit by bit (like 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

- 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
### Keyed Permutation

For $N$-bit input, $2^N!$ possible permutations
For $K$-bit key, $2^K$ 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>

Key = 00
Key = 01
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

Key

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

Block of ciphertext

Procedure must be reversible (for decryption)
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 × 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 × 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 × 10^{38}</td>
<td>2^{127} μs = 5.4 × 10^{24} years</td>
<td>5.4 × 10^{18} years</td>
</tr>
<tr>
<td>168</td>
<td>2^{168} = 3.7 × 10^{50}</td>
<td>2^{167} μs = 5.9 × 10^{36} years</td>
<td>5.9 × 10^{30} years</td>
</tr>
<tr>
<td>26 characters</td>
<td>26! = 4 × 10^{26}</td>
<td>2 × 10^{26} μs = 6.4 × 10^{12} years</td>
<td>6.4 × 10^{6} years</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)
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

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

[Encrypted image]

[Chanellings from Wikipedia]
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
<table>
<thead>
<tr>
<th>decrypt</th>
<th>decrypt</th>
<th>decrypt</th>
<th>decrypt</th>
</tr>
</thead>
</table>
| ciphertext

Initialization vector

4/19/19
CSE 484 / CSE M 584 - Spring 2019
AES in ECB mode

AES in CBC mode

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

[Picture due to Bart Preneel]
CBC and Electronic Voting

Initialization vector (supposed to be random)

plaintext

DES

DES

DES

DES

ciphertext

Found in the source code for Diebold voting machines:

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
- ctr+1
  - Key
  - block cipher
- ctr+2
  - Key
  - block cipher
- ctr+3
  - Key
  - block cipher
- pt
  - ⊕
- ciphertext

- 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>ct</td>
<td>ct</td>
<td>ct</td>
<td>ct</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>pt</td>
<td>pt</td>
<td>pt</td>
<td>pt</td>
</tr>
</tbody>
</table>