Reminder

• Checkpoint for lab #1 due Monday @ 8pm
  – Submit md5 hashes to Catalyst dropbox
Recap: 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 $sk$. 
Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.

Message = M
Ciphertext = C

Alice
K

Encrypt
K

Decrypt
K

Bob
K

Adversary
Achieving Privacy (Asymmetric)

Encryption schemes: A tool for protecting privacy.

\[ \text{Alice} \quad \text{pk}_A, \text{sk}_A \quad \begin{align*} \text{pk}_B & \to \text{Encrypt} \quad \text{C} \\ \text{pk}_B & \to \text{Decrypt} \quad \text{sk}_B \end{align*} \quad \text{Bob} \quad \text{pk}_B, \text{sk}_B \]

Message = M
Ciphertext = C

Adversary
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**: $\text{Ciphertext}(\text{Key}, \text{Message}) = \text{Message} \oplus \text{Key}$
  - 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
  - No efficient algorithm can tell this sequence from truly random

• $\text{Ciphertext}(\text{Key}, \text{Msg}) = \text{Msg} \oplus \text{PRNG}(\text{Key})$
  - Message processed bit by bit (unlike block cipher)
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)
For N-bit input, $2^N!$ possible permutations

Idea for how to use a **keyed** permutation: split plaintext into blocks; for each block use secret key to pick a permutation

- Without the key, permutation should "look random"
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)

Procedure must be reversible (for decryption)

Block of ciphertext

repeat for several rounds
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>$2^{31} \mu s = 35.8$ minutes</td>
<td>2.15 milliseconds</td>
</tr>
<tr>
<td>56</td>
<td>$2^{56} = 7.2 \times 10^{16}$</td>
<td>$2^{55} \mu s = 1142$ years</td>
<td>10.01 hours</td>
</tr>
<tr>
<td>128</td>
<td>$2^{128} = 3.4 \times 10^{38}$</td>
<td>$2^{127} \mu s = 5.4 \times 10^{24}$ 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>$2^{167} \mu s = 5.9 \times 10^{36}$ 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>$2 \times 10^{26} \mu s = 6.4 \times 10^{12}$ 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 ---> 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

```
  □□□□□
  □□□□□
  □□□□□ (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

Encrypt in ECB mode

[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

\[ \oplus \]

key

\[ \oplus \]

ciphertext

Initialization vector

decrypt

decrypt

decrypt

decrypt

\[ \oplus \]
ECB vs. CBC

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

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 → block cipher → pt
- ctr+1 → block cipher → pt
- ctr+2 → block cipher → pt
- ctr+3 → block cipher → pt

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

Initial ctr

\[
\begin{align*}
\text{ctr} & \quad \text{ctr+1} & \quad \text{ctr+2} & \quad \text{ctr+3} \\
\text{block cipher} & \quad \text{block cipher} & \quad \text{block cipher} & \quad \text{block cipher} \\
\text{ct} & \quad \text{ct} & \quad \text{ct} & \quad \text{ct} \\
\text{pt} & \quad \text{pt} & \quad \text{pt} & \quad \text{pt}
\end{align*}
\]

Key

\[
\begin{align*}
\text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus
\end{align*}
\]
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?

- Fixed mapping from plaintexts to ciphertexts?
  - What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?
  - Implication: encryption must be randomized or stateful
How Can a Cipher Be Attacked?

• Attackers knows ciphertext and encryption algthm
  – 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

Crook #1 changes his PIN to a number of his choice

PIN is encrypted and transmitted to bank

cipher(key,PIN)

Crook #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, he cannot verify his guess
  – Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts

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

- Leaking even a little bit of information about the plaintext can be disastrous
- Electronic voting
  - 2 candidates on the ballot (1 bit to encode the vote)
  - If ciphertext leaks the parity bit of the encrypted plaintext, eavesdropper learns the entire vote
- Also, want a strong definition, that implies other definitions (like not being able to obtain key)