Week 3: Lecture B
Block Ciphers

Thursday, September 7, 2023
Announcements

- **Project 1: Crypto** released
  - **Deadline:** Thursday, September 21st by 11:59PM
Project Tips

- Projects are challenging—**you’re performing real-world attacks!**
  - Build off of lecture concepts
  - Make sure you understand the lectures
  - Prepare you to defend **in the real world**

- **Suggested strategy:** _get high-level idea down, then start implementing_
  1. Go through assignment and start sketching-out your approach
  2. **Come to Office Hours and ask if you’re on the right track!**
  3. Then start building your program

- Don’t get discouraged—**we are here to help!**
  - Most issues are cleared up in a few minutes of white-boarding
Announcements

See Discord for meeting info!

www.utahsec.com
Questions?
Last time on CS 4440...

Pseudo-random Keys
One-time Pads
Transposition Ciphers
Cipher Metrics
Generating Random Keys

- **Physical randomness:**
  - ???
Generating Random Keys

- **Physical randomness:**
  - Coin flips
  - Atomic decay
  - Thermal noise
  - Electromagnetic noise
  - Physical variation
    - Clock drift
    - DRAM decay
    - Image sensor errors
    - SRAM startup-state
  - Lava Lamps
Generating Random Keys

- Physical randomness:
  - Coin flips
  - Atomic decay
  - Thermal noise
  - Electromagnetic noise
  - Physical variation:
    - Clock drift
    - DRAM decay
    - Image sensor errors
    - SRAM startup-state
  - Lava Lamps
Generating Random Keys

Physical randomness:
- Coin flips
- Atomic decay
- Thermal noise
- Electromagnetic noise
- Physical variation
  - Clock drift
  - DRAM decay
  - Image sensor errors
  - SRAM startup-state
- Lava Lamps

Highest guarantees of **security**

**Difficult** to use, or **rate-limited**
Pseudo-random Key Generators

- What is **true randomness**?
  - ???
What is true randomness?
- Physical process that’s inherently random
- Secure yet impractical
  - Scarce, hard to use
  - Rate-limited

Pseudo-random generator (PRG)
- Input: ???
- Output: ???
What is **true randomness**?
- **Physical** process that’s inherently random
- **Secure** yet **impractical**
  - Scarce, hard to use
  - Rate-limited

**Pseudo-random generator (PRG)**
- **Input**: a small **seed** that is **truly random**
- **Output**: long sequence that **appears random**
Pseudo-random Generators (PRGs)

- We say a **PRG** is **secure** if Mallory can’t do better than random guessing.

**Problem:** How much **true randomness** is **enough**?
- **Example:** one coin flip = Mallory needs very few tries to guess.

**Problem:** Is our “true randomness” **truly random**?
- **Example:** coin flip output = one in two. Lava lamps have way more!

**Solutions:**
- ???
We say a **PRG** is **secure** if Mallory can’t do better than random guessing.

**Problem:** How much **true randomness** is **enough**?
- **Example:** one coin flip = Mallory needs very few tries to guess.

**Problem:** Is our “true randomness” **truly random**?
- **Example:** coin flip output = one in two. Lava lamps have way more!

**Solutions:**
- Generate a bunch of true randomness over a long time from a high entropy source.
- Run through a **PRF** to get an easy-to-work-with, fixed-length randomness (e.g., 256 bits).
Where do you get true randomness?

Modern OSes typically collect randomness.

They give you API calls to capture it.

e.g., Linux:
- `/dev/random` is a device that gives random bits; it blocks until available
- `/dev/urandom` gives output of a PRG; nonblocking; seeded from `/dev/random` eventually
One-time Pads

- Alice and Bob generate ???
One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad \( k \)
  - Encryption: \( c_i := p_i \ XOR \ k_i \)
  - Decryption: \( p_i := c_i \ XOR \ k_i \)

- Are they practical?
  - ???

- Are they secure?
  - ???

```
a \ XOR \ b \ XOR \ b = a
a \ XOR \ b \ XOR \ a = b
```
One-time Pads

- Alice and Bob generate a plaintext-length string of random bits: the one-time pad $k$
  - Encryption: $c_i := p_i \text{ XOR } k_i$
  - Decryption: $p_i := c_i \text{ XOR } k_i$
- Are they practical?
  - ???
- Are they secure?
  - ???

**Provably Secure** (if key is random + not reused)

**Highly Impractical**
Attacking OTPs

- What happens if the key isn’t truly random?
Attacking OTPs

- What happens if the key isn’t truly random?
  - If Mallory correctly guesses some key bits, she can recover parts of the plaintext

\[
(a \text{ XOR } k) \text{ XOR } g
\]
Attacking OTPs

- What if Mallory intercepts multiple messages that reuse the same key?
  - Mallory can XOR them together to recover partial plaintext information!

\[(a \text{ XOR } k) \oplus \text{SEND} \oplus \text{CASH} = (b \text{ XOR } k)\]
Attacking OTPs

- What if Mallory intercepts multiple messages that **reuse** the same key?
  - Mallory can **XOR them together** to recover partial plaintext information!

\[(a \text{ XOR } k) \text{ XOR } (b \text{ XOR } k) = a \text{ XOR } b\]
Stream Cipher

- **Idea:** Use a Pseudo-random Generator instead of a truly random pad

- **Recall:** a secure PRG inputs a true-random seed, outputs a stream that’s indistinguishable from true randomness (unless attacker knows seed)

1. Start with a shared secret truly random seed (from a lava lamp, mouse clicks, etc.)
2. Alice & Bob each use this seed to seed their PRG and generate \( k \) bits of PRG output
3. To encrypt and decrypt, perform the same operations as the One-time Pad:
   - Encryption: \( c_i := p_i \text{ XOR } k_i \)
   - Decryption: \( p_i := c_i \text{ XOR } k_i \)
Stream Cipher

- **Idea:** Use a pseudorandom generator instead of a truly random pad.

- **Recall:** Secure PRG inputs a seed $k$, outputs a stream practically indistinguishable from true randomness (unless attacker knows $k$).

1. Start with shared secret key truly random number $k$.
2. Alice & Bob each use $k$ to seed the PRG.
3. To encrypt, Alice XORs next bit of her generator’s output with next bit of plaintext.
4. To decrypt, Bob XORs next bit of his generator’s output with next bit of ciphertext.

**What is the tradeoff between an **OTP** and **Stream Cipher**?**
Stream Cipher

- **Idea:** Use a pseudorandom generator instead of a truly random pad.
- **Recall:** Secure PRG inputs a seed $k$, outputs a stream practically indistinguishable from true randomness (unless attacker knows $k$).

1. Start with shared secret key truly random number $k$
2. Alice & Bob each use $k$ to seed the PRG
3. To encrypt, Alice XORs next bit of her generator’s output with next bit of plaintext
4. To decrypt, Bob XORs next bit of his generator’s output with next bit of ciphertext

**Attack potential?**
Stream Cipher

- **Idea:** Use a pseudorandom generator instead of a truly random pad.

- **Recall:** Secure PRG inputs a seed $k$, outputs a stream practically indistinguishable from true randomness (unless attacker knows $k$).

1. Start with shared secret key.
2. Alice & Bob each use $k$ to seed the PRG.
3. To encrypt, Alice XORs next bit of her generator's output with next bit of plaintext.
4. To decrypt, Bob XORs next bit of his generator's output with next bit of ciphertext.

*Attack potential?*

*Seed or key reuse* helps Mallory recover plaintext!
Substitution vs Transposition Ciphers

- **Substitution**: replace plaintext symbols with others
  - Examples: ???
Substitution vs Transposition Ciphers

- **Substitution**: replace plaintext symbols with others
  - **Examples**: simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness**: ???
Substitution vs Transposition Ciphers

- **Substitution**: replace plaintext symbols with others
  - **Examples**: simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness**: although letters changed, *frequencies upheld*

- **Transposition**: plaintext symbols are rearranged
  - **Examples**: ???

Stefan Nagy
Substitution vs Transposition Ciphers

- **Substitution**: replace plaintext symbols with others
  - **Examples**: simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness**: although letters changed, *frequencies upheld*

- **Transposition**: plaintext symbols are rearranged
  - **Examples**: columnar, rail fence / zig zag / scytale, grids
  - **Key weakness**: ???
Substitution vs Transposition Ciphers

- **Substitution**: replace plaintext symbols with others
  - **Examples**: simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness**: although letters changed, *frequencies upheld*

- **Transposition**: plaintext symbols are rearranged
  - **Examples**: columnar, rail fence / zig zag / scytale, grids
  - **Key weakness**: plaintext letters in ciphertext; *anagram attacks*
Columnar Transposition

- **Rearrange plaintext symbols to create ciphertext**
  - Create a table with \(|k|\) columns and \(|p|/|k|\) rows (\(k\) is the keyword)
  - Place plaintext symbols in columns (left to right), cycling around to next row of the first column when current row of last column is filled
  - Create the ciphertext by writing entire columns (as a serial stream) to the output, where the keyword determines the column order

- **Example:**
  - \(k = \text{“ZEBRAS”} \ (632415)\)
  - \(p = \text{“We are discovered flee at once”}\)
  - \(c = \text{EVLNX ACDTQ ESEAM ROFOP DEECWD WIREE}\)
  - Replace **null** with nonsense symbol

<table>
<thead>
<tr>
<th>6</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>E</td>
<td>A</td>
<td>R</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>I</td>
<td>S</td>
<td>C</td>
<td>O</td>
<td>V</td>
<td>E</td>
</tr>
<tr>
<td>R</td>
<td>E</td>
<td>D</td>
<td>F</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>T</td>
<td>O</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Columnar Transposition

- **How does Bob decrypt** Alice’s columnar-transposition-encrypted message?

  \[ k = "\text{ZEBRAS}" (632415) \]

  \[ p = \text{"We are discovered flee at once"} \]

  \[ c = \begin{array}{ccccccc}
  \text{EVLNX} & \text{ACDTQ} & \text{ESEAM} \\
  \text{ROFOP} & \text{DEEC} & \text{D WIREE}
  \end{array} \]
Columnar Transposition

- **How does Bob decrypt** Alice’s columnar-transposition-encrypted message?

- **k = “ZEBRAS” (632415)**
- **p = “We are discovered flee at once”**
- **c = EVLNX ACDTQ ESEAM ROFOP DEECD WIREE**

<table>
<thead>
<tr>
<th>E</th>
<th>A</th>
<th>E</th>
<th>R</th>
<th>D</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>C</td>
<td>S</td>
<td>O</td>
<td>E</td>
<td>I</td>
</tr>
<tr>
<td>L</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>E</td>
<td>R</td>
</tr>
<tr>
<td>N</td>
<td>T</td>
<td>A</td>
<td>O</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>X</td>
<td>Q</td>
<td>M</td>
<td>P</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>
**Columnar Transposition**

- How does Bob decrypt Alice’s columnar-transposition-encrypted message?

\[ k = \text{“ZEBRAS” (632415)} \]

\[ p = \text{“We are discovered flee at once”} \]

\[ c = \text{EVLNX ACDTQ ESEAM ROFOP DEECD WIREE} \]
How does Bob decrypt Alice’s columnar-transposition-encrypted message?

k = “ZEBRAS” (632415)

p = “We are discovered flee at once”

c = EVLNX ACDTQ ESEAM
ROFOP DEECD WIREE
Can you decrypt the ciphertext?

\[ c = \text{SAKSECROYN shaving to YOUL} \]

\[ k = \text{“TEAMS”} \]
Columnar Transposition

- Can you decrypt the ciphertext?

\[ c = \text{SAKSECROYNSBOWOLYUOL} \]

\[ k = \text{“TEAMS” (52134)} \]
Columnar Transposition

- Can you decrypt the ciphertext?

**c = SAKSECROYNBOWOLYUOL**

**k = “TEAMS” (52134)**

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>E</th>
<th>A</th>
<th>M</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>E</td>
<td>S</td>
<td>Y</td>
<td>O</td>
</tr>
<tr>
<td>1</td>
<td>U</td>
<td>C</td>
<td>A</td>
<td>N</td>
<td>W</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>R</td>
<td>K</td>
<td>S</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
<td>O</td>
<td>S</td>
<td>B</td>
<td>L</td>
</tr>
</tbody>
</table>
Columnar Transposition

- Can you decrypt the ciphertext?

\[ c = \text{SAKSECROYNSBOWOLYUOL} \]

\[ k = "\text{TEAMS}" (52134) \]

<table>
<thead>
<tr>
<th>T</th>
<th>E</th>
<th>A</th>
<th>M</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Y</td>
<td>E</td>
<td>S</td>
<td>Y</td>
<td>O</td>
</tr>
<tr>
<td>U</td>
<td>C</td>
<td>A</td>
<td>N</td>
<td>W</td>
</tr>
<tr>
<td>O</td>
<td>R</td>
<td>K</td>
<td>S</td>
<td>O</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Columnar Transposition

- Can you decrypt the ciphertext?

  \[ c = \text{SAKSECROYNBOWOLYUOL} \]
  \[ k = \text{“TEAMS” (52134)} \]

- “Yes, you can work solo” (on projects)
  - Though we don’t recommend it! 😊

A crummy reminder of course policy?
Can we make transposition stronger?
Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!

- $k_1 = \text{"ZEBRAS" (632415)}$
  - $c_1 = \text{EVLNX ACDTQ ESEAM ROFOP DEECD WIREE}$

- $k_2 = \text{"STRIPE" (632415)}$
  - $c_2 = \text{CAEIX NSOIN AEDRX LEFWS EDREE VTOCG}$

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>V</td>
<td>L</td>
<td>N</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
<td>E</td>
<td>S</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>R</td>
<td>O</td>
<td>F</td>
<td>O</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>C</td>
<td>W</td>
<td>I</td>
<td>R</td>
<td>I</td>
</tr>
<tr>
<td>E</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!
  - $k_1 = \text{"ZEBRAS"(632415)}$
  - $c_1 = \text{EVLNX ACDTQ ESEAM ROFOP DEECD WIREE}$
  - $k_2 = \text{"STRIPE"(632415)}$
  - $c_2 = \text{CAEIX NSOIN AEDRX LEFWS EDREE VTOCG}$

- **Apply Fractionation:**
  - Eliminate anagrams!
  - “We’re discovered flee at once!”

```
0101011101100101...
```

```
E V L N A C
D T E S E A
R O F O D E
E C W I R I
E null null null null null
```
Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!
  
  \[
  k_1 = \text{"ZEBRAS" (632415)}
  \]
  \[
  c_1 = \begin{array}{c}
  E \\
  V \\
  L \\
  N \\
  A \\
  C \\
  \end{array}
  \]
  \[
  k_2 = \text{"STRIPE" (632415)}
  \]
  \[
  c_2 = \begin{array}{c}
  E \\
  C \\
  W \\
  I \\
  R \\
  I \\
  \end{array}
  \]

- **Apply Fractionation:**
  - Eliminate anagrams!

- **Apply Substitution:**
  - Increase entropy + eliminate anagrams!

\[
01010111101100101\ldots
\]

\[
k = \begin{array}{c}
  A \\
  B \\
  C \\
  A \\
  B \\
  C \\
  \end{array}
\]

\[
c_2 = \begin{array}{c}
  E \\
  W \\
  N \\
  N \\
  C \\
  C \\
  \end{array}
\]

\[
\text{\textit{We’re discovered flee at once!}}
\]
Cipher Metrics

- How we “weigh” a cipher’s resilience to cryptanalysis

- “Confusion”
  - ???

- “Diffusion”
  - ???
Cipher Metrics

- How we “weigh” a cipher’s resilience to cryptanalysis

  - “Confusion”
    - Every bit of the ciphertext should depend on several parts of the plaintext
    - Maintains that the ciphertext is statistically independent of the plaintext

  - “Diffusion”
    - A change to one plaintext bit should change 50% of the ciphertext bits
    - A change to one ciphertext should change 50% of the plaintext bits
    - Plaintext features spread throughout the entire ciphertext
## Exercise: Cipher Metrics

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Relies on?</th>
<th>Strength?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caesar</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
### Exercise: Cipher Metrics

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Relies on?</th>
<th>Strength?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caesar</td>
<td>Confusion</td>
<td>Weak</td>
<td>Frequencies unchanged</td>
</tr>
<tr>
<td>Vigenere</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>One-time Pad, Stream Cipher</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Transposition</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Fractionation</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
## Exercise: Cipher Metrics

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Relies on?</th>
<th>Strength?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caesar</td>
<td>Confusion</td>
<td>Weak</td>
<td>Frequencies unchanged</td>
</tr>
<tr>
<td>Vigenere</td>
<td>Confusion</td>
<td>Weak</td>
<td>Frequencies unchanged</td>
</tr>
<tr>
<td>One-time Pad, Stream Cipher</td>
<td>Confusion</td>
<td>Strong</td>
<td>Key change = relationship cannot be determined</td>
</tr>
<tr>
<td>Transposition</td>
<td>Diffusion</td>
<td>Weak</td>
<td>Symbols unchanged</td>
</tr>
<tr>
<td>Fractionation</td>
<td>Both!</td>
<td>Strong</td>
<td>Symbols changed, spread</td>
</tr>
</tbody>
</table>
Food for thought...

- **Question:** do we care about confusion and diffusion in cryptographic hashes?
Food for thought...

**Question:** do we care about confusion and diffusion in cryptographic hashes?
- Absolutely we do!

**Implications of low confusion/diffusion:**
- Tampering, forgery, collisions
- Pre-image attacks
Questions?
This time on CS 4440...

Block Ciphers
DES and AES
Block Cipher Modes
Building a Secure Channel
Confidentiality: ensure that only trusted parties can read the message

Terminology:

- **p**: plaintext: original, readable message
- **c**: ciphertext: transmitted, unreadable message
- **k**: secret key: known only to Alice and Bob; facilitates $p \rightarrow c$ and $c \rightarrow p$
- **$E$**: encryption function: $E(p, k) \rightarrow c$
- **$D$**: decryption function: $D(c, k) \rightarrow p$
Message Confidentiality

- **Confidentiality**: ensure that only trusted parties can read the message
- Terminology:
  - **p** plaintext: original, readable message
  - **c** ciphertext: transmitted, unreadable message
  - **k** secret key: known only to Alice and Bob; facilitates \( p \rightarrow c \) and \( c \rightarrow p \)
  - **E** encryption function: \( E(p, k) \rightarrow c \)
  - **D** decryption function: \( D(c, k) \rightarrow p \)
Key-based Encryption Schemes

“Symmetric” Key
- Encryption and decryption relies on the same key.
- Communicating parties must share key in advance.
- Examples: ???
Key-based Encryption Schemes

“Symmetric” Key
- Encryption and decryption relies on the same key
- Communicating parties must share key in advance
- Examples:
  - Caesar, Vigènere
  - One-time Pad, Stream
  - Transposition ciphers
Stream cipher: operates on **individual bits** (or bytes); **one at a time**
- Generates pseudo-random key bits that are XOR'd to plaintext bits

**Encryption**
- plaintext = HELLO
- key = AXHJB
- ciphertext = KMIVE

**Decryption**
- ciphertext = KMIVE
- key = AXHJB
- plaintext = HELLO
Stream cipher: operates on individual bits (or bytes); one at a time
- Generates pseudo-random key bits that are XOR'd to plaintext bits

Confusion and diffusion?

No diffusion—symbols are not spread around!
Block Ciphers
Functions that encrypts fixed-size blocks with a reusable key

Inverse function decrypts when used with same key

The most commonly used encryption approach for confidentiality.
Block Ciphers vs. Hashes

- Hash functions:
  - ???
Hash functions:
- **Must not** have collisions
- **Must not** be reversible
- **Goal:** integrity
  - Detect message tampering
Block Ciphers vs. Hashes

- **Hash functions:**
  - **Must not** have collisions
  - **Must not** be reversible
  - **Goal:** integrity
    - Detect message tampering

- **Block Ciphers:**
  - **Must not** have collisions
  - **Must be** reversible
  - **Goal:** confidentiality
    - Keep secret message secret
Block Ciphers vs. Hashes

- **Hash functions:**
  - **Must not** have collisions
  - **Must not** be reversible
  - **Goal:** integrity
    - Detect message tampering

- **Block Ciphers:**
  - **Must not** have collisions
  - **Must** be reversible
  - **Goal:** confidentiality
    - Keep secret message secret

A block cipher is not a pseudo-random function
Block Ciphers vs. Hashes

- **Hash functions:**
  - Must **not** have collisions
  - Must **not** be reversible
  - **Goal:** integrity
    - Detect message tampering

- **Block Ciphers:**
  - Must **not** have collisions
  - Must **be** reversible
  - **Goal:** confidentiality
    - Keep secret message secret

A block cipher is **not a** pseudo-random function

A block cipher is **a** pseudo-random permutation
Pseudo-random Permutation (PRP)

- Defined similarly to a PRF:
  - Practically indistinguishable from a random permutation without secret $k$

- **Main challenge**: design a function that’s **invertible**... but only with the key

- **Minimal properties of a good block cipher:**
  - Highly nonlinear (“confusion”)
  - Mixes input bits together (“diffusion”)
  - Dependent on the key
What we want at a high-level:

- Function from $n$-bit input to $n$-bit output
- Ideally, one bit flip of the input results in $50\%$ of output bits flipping
- Distinct inputs yield distinct outputs
- Thus, an invertible bijection
**Block cipher:** operates on **fixed-length groups** of bits called **blocks**
- Processes blocks using a **reversible, non-colliding** function

### Encryption
- plaintext = $B_1$
- plaintext = $k \Rightarrow Enc$
- ciphertext = $C_1$

### Decryption
- ciphertext = $C_1$
- ciphertext = $k \Rightarrow Dec$
- plaintext = $B_1$
Block vs. Stream Ciphers

- **Major categories of SKE**
  - **Stream cipher**: operates on *individual bits* (or bytes); **one at a time**
  - **Block cipher**: operates on *fixed-length groups* of bits called **blocks**

- **Only a few symmetric methods are used today**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Year approved</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Encryption Standard (<strong>DES</strong>)</td>
<td>1977</td>
<td>1998: EFF's Deep Crack breaks a DES key in 56 hrs</td>
</tr>
<tr>
<td>DES-Cipher Block Chaining (<strong>DES-CBC</strong>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triple DES – (<strong>TDES</strong> or <strong>3DES</strong>)</td>
<td>1999</td>
<td></td>
</tr>
<tr>
<td>Advanced Encryption Standard (<strong>AES</strong>)</td>
<td>2001</td>
<td>Among the most used today</td>
</tr>
</tbody>
</table>

**Other symmetric encryption methods**
- IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish
Questions?
Data Encryption Standard (DES)
Breaking up long messages into “blocks”

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size...
  - But message might not be *multiples* of block size
Breaking up long messages into “blocks”

- **Challenge**: How to encrypt longer messages?
  - Can only encrypt in units of cipher block size...
  - But message might not be multiples of block size

- **Solution**: Append **padding** to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add \( n \) bytes that have value \( n \)
Breaking up long messages into “blocks”

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size...
  - But message might not be multiples of block size

- **Solution:** Append **padding** to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add \( n \) bytes that have value \( n \)

- **Challenge:** What if message **terminates** a block?
  - End of message might be misread as padding!
Breaking up long messages into “blocks”

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size...
  - But message might not be multiples of block size

- **Solution:** Append padding to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add $n$ bytes that have value $n$

- **Challenge:** What if message terminates a block?
  - End of message might be misread as padding!

- **Solution:** Append an entire new block of padding
Data Encryption Standard (DES)

- DES is a **block, symmetric encryption** scheme
  - Uses a **64-bit** key
  - Plaintext divided and encrypted as fixed-size, 64-bit blocks
  - Different **modes** of encryption—each with different security implications

<table>
<thead>
<tr>
<th>Methods</th>
<th>Year approved</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Encryption Standard (DES)</td>
<td>1977</td>
<td>1998: EFF's Deep Crack breaks a DES key in 56 hrs</td>
</tr>
<tr>
<td>DES-Cipher Block Chaining (DES-CBC)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triple DES – (TDES or 3DES)</td>
<td>1999</td>
<td></td>
</tr>
<tr>
<td>Advanced Encryption Standard (AES)</td>
<td>2001</td>
<td>Among the most used today</td>
</tr>
</tbody>
</table>

**Other symmetric encryption methods**
IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish
A variety of “block cipher modes” exist today

- As time went on, researchers found issues with them and proposed better ones
- We'll talk about a few of these: **Electronic Codebook** and **Cipher Block Chaining**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Year approved</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Encryption Standard (DES)</td>
<td>1977</td>
<td>1998: EFF's Deep Crack breaks a DES key in 56 hrs</td>
</tr>
<tr>
<td>DES-Cipher Block Chaining (DES-CBC)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triple DES – (TDES or 3DES)</td>
<td>1999</td>
<td></td>
</tr>
<tr>
<td>Advanced Encryption Standard (AES)</td>
<td>2001</td>
<td>Among the most used today</td>
</tr>
<tr>
<td>Other symmetric encryption methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DES Modes: Electronic Codebook
Mode #1: Electronic Codebook (ECB)

- Electronic Codebook (ECB)
  - Message divided into code blocks
  - Each block encrypted separately

plaintext = B₁ ↓ k Enc C₁
           ↓    ↓    ↓
key Enc k Enc C₁
           ↓    ↓    ↓
ciphertext = C₁ C₂ C₃
Mode #1: Electronic Codebook (ECB)

- Electronic Codebook (ECB)
  - Message divided into code blocks
  - Each block encrypted \textit{separately}; decrypted separately too

\[
\text{ciphertext} = \begin{array}{c} C_1 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ B_1 \end{array} \quad \begin{array}{c} C_2 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ B_2 \end{array} \quad \begin{array}{c} C_3 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ B_3 \end{array}
\]

\[
\text{key} \Rightarrow \begin{array}{c} \text{Dec} \\ \downarrow \end{array} \quad \text{key} \Rightarrow \begin{array}{c} \text{Dec} \\ \downarrow \end{array} \quad \text{key} \Rightarrow \begin{array}{c} \text{Dec} \\ \downarrow \end{array}
\]

\[
\text{plaintext} = \begin{array}{c} B_1 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ C_1 \end{array} \quad \begin{array}{c} B_2 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ C_2 \end{array} \quad \begin{array}{c} B_3 \\ \downarrow \\ \text{Dec} \\ \downarrow \\ C_3 \end{array}
\]
Mode #1: Electronic Codebook (ECB)

- **ECB Strengths:**
  - Construction is **un-chained**
    - Message can be ???

plaintext = \[
B_1 \downarrow \quad B_2 \downarrow \quad B_3 \downarrow
\]

key $\Rightarrow$ \[
\downarrow \quad \downarrow \quad \downarrow
\]

k $\Rightarrow$ \[
\downarrow \quad \downarrow \quad \downarrow
\]

ciphertext = \[
C_1 \quad C_2 \quad C_3
\]
**ECB Strengths:**
- Construction is **un-chained**
  - Message can be split up and processed in parallel—**fast!**
  - No need to wait on previous block’s encryption

plaintext = $B_1$ \[\downarrow\] Enc \[\downarrow\] ciphertext = $C_1$

key $\Rightarrow$ $k$ $\Rightarrow$ $k$ $\Rightarrow$ Enc

thread1

thread2

thread1

plaintext = $B_1$ \[\downarrow\] Enc \[\downarrow\] ciphertext = $C_1$

plaintext = $B_2$ \[\downarrow\] Enc \[\downarrow\] ciphertext = $C_2$

plaintext = $B_3$ \[\downarrow\] Enc \[\downarrow\] ciphertext = $C_3$
Mode #1: Electronic Codebook (ECB)

- **ECB Drawbacks:**
  - Identical plaintext blocks produce same ciphertext
  - This results in **low uniqueness**

\[
\text{plaintext} = B_1 \rightarrow \text{Enc} \rightarrow C_1 \\
B_2 \rightarrow \text{Enc} \rightarrow C_2 \\
B_3 \rightarrow \text{Enc} \rightarrow C_3
\]
**Mode #1: Electronic Codebook (ECB)**

- **ECB Drawbacks:**
  - Identical plaintext blocks produce same ciphertext
  - This results in **low diffusion**

plaintext = \[B_1 \quad B_2 \quad B_3\]

key ⇒ \[\text{Enc}\] \[k ⇒ \text{Enc}\] \[k ⇒ \text{Enc}\]

ciphertext = \[C_1 \quad C_2 \quad C_3\]

enabled
Mode #1: Electronic Codebook (ECB)

**ECB Drawbacks:**
- Do larger block sizes increase diffusion?
  - Yes—but at what cost???
ECB Drawbacks:
- Do larger block sizes increase diffusion?
  - Yes—but at what cost
    - Much more impractical
    - E.g., higher memory footprint

(a) Plaintext image, 2000 by 1400 pixels, 24 bit color depth.
How can we increase diffusion?
DES Modes: Cipher Block Chaining
**Mode #2: Cipher Block Chaining (CBC)**

- **Key idea:** seed current block with ciphertext from the previous block

```
plaintext = B_1  
  ↓  ⊕  ↓  
  key => Enc  
  ↓  
  ciphertext = C_1
```

```
plaintext = B_2  
  ↓  ⊕  ↓  
  k => Enc  
  ↓  
  ciphertext = C_2
```

```
plaintext = B_3  
  ↓  ⊕  ↓  
  k => Enc  
  ↓  
  ciphertext = C_3
```
Mode #2: Cipher Block Chaining (CBC)

- **Key idea:** seed current block with **ciphertext from the previous block**
  - Since first block has no “previous” cipher, seed it with a 64-bit initialization vector (I.V.)
    - A random or pseudo-random block that’s unpredictable
Mode #2: Cipher Block Chaining (CBC)

- **Decryption** operates similarly:

  ciphertext = $C_1 \oplus \text{key} \Rightarrow \text{Dec} \Rightarrow B_1$

  $\downarrow$

  ciphertext = $C_2 \oplus k \Rightarrow \text{Dec} \Rightarrow B_2$

  $\downarrow$

  ciphertext = $C_3 \oplus k \Rightarrow \text{Dec} \Rightarrow B_3$
Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - **More diffusion!**
    - Negates ECB’s need for super-large blocks
Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - More diffusion!
    - Negates ECB’s need for super-large blocks

- **CBC Drawbacks:**
  - Completely sequential
    - ???

---

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

96

Stefan Nagy
Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - More diffusion!
    - Negates ECB’s need for super-large blocks

- **CBC Drawbacks:**
  - Completely sequential
    - Cannot be parallelized!
    - No leveraging advances in multi-threading etc.
Questions?
Advanced Encryption Standard (AES)
Advanced Encryption Standard (AES)

- Today’s most common block cipher
  - Designed by NIST competition, with a very long public discussion
  - Widely believed to be secure... but we don’t know how to prove it

- Variable **key size:**
  - 128-bit fairly common; also 192-bit and 256-bit versions

- Input message is split into **128-bit blocks**

- Ten **rounds:**
  - Split $k$ into ten subkeys (key scheduling)
  - Performs set of identical operations ten times (each with different subkey)
AES Cliff Notes

- Systematically designed through a read/blue team competition by NIST
  - Layered design to remove flaws of individual components
  - Prevent statistical leakage
    - Letter frequency of substitution ciphers
    - Anagrams of transposition ciphers

- Many fancier “modes” with ordering counters, etc.
  - Efficient software and hardware implementations

- Exposes security performance tradeoff to user
  - 128-bit key: 10 rounds
  - 192-bit key: 12 rounds
  - 256-bit key: 14 rounds

Disclaimer: details are hairy—don’t worry about them.
Secure Channels
Building a Secure Channel

- What if you want confidentiality and integrity at the same time?
  - Which would you perform first: encrypting or hashing? And why?
<table>
<thead>
<tr>
<th>Option</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt (Confidentiality) first</td>
<td>0%</td>
</tr>
<tr>
<td>Hash (Integrity) first</td>
<td>0%</td>
</tr>
</tbody>
</table>
What if you want **confidentiality** and **integrity** at the same time?
- Which would you perform **first**: encrypting or hashing? And why?
Limitations of Symmetric Crypto

- Complex mathematics
  - Hardware and software efficiency is key
  - A huge study of modern cryptography research

- Requires pre-shared keys
  - The keys need to stay secret always
Limitations of Symmetric Crypto

- Complex mathematics
  - Hardware and software efficiency is key
  - A huge study of modern cryptography research

- Requires pre-shared keys
  - The keys need to stay secret always

**Amazing fact:** Alice and Bob can have a **public** conversation to derive a shared **secret** key
Next time on CS 4440...

Public-key Encryption, Signatures