0. Prologue: A Simple Machine
Alice wants to send a secret message to Bob

- Can you read the secret message $gX76W3v7K$
- But Bob can. How?
Encryption Machine

**Goal.** Design a machine to encrypt and decrypt data.

\[
\begin{array}{cccccccc}
S & E & N & D & M & O & N & E & Y \\
g & X & 7 & 6 & W & 3 & v & 7 & K \\
S & E & N & D & M & O & N & E & Y
\end{array}
\]
**Goal.** Design a machine to encrypt and decrypt data.

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>x</td>
<td>7</td>
<td>6</td>
<td>W</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>K</td>
</tr>
<tr>
<td>S</td>
<td>E</td>
<td>N</td>
<td>D</td>
<td>M</td>
<td>O</td>
<td>N</td>
<td>E</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Enigma encryption machine.**
- "Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.
Data is a sequence of bits. [bit = 0 or 1]

- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

can use decimal digits, letters, or some other system, but bits are more easily encoded physically ("on-off", "up-down", "hot-cold",...)

thousands of bits

billions of bits

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com

image courtesy of David August
A Digital World

Data is a sequence of bits. [bit = 0 or 1]
- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, …

Ex. Base64 encoding of text.
- Simple method for representing A-Z, a-z, 0-9, +, /
- 6 bits to represent each symbol (64 symbols)

<table>
<thead>
<tr>
<th>000000</th>
<th>A</th>
<th>001000</th>
<th>I</th>
<th>010000</th>
<th>Q</th>
<th>011000</th>
<th>Y</th>
<th>100000</th>
<th>g</th>
<th>101000</th>
<th>o</th>
<th>110000</th>
<th>w</th>
<th>111000</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001</td>
<td>B</td>
<td>001001</td>
<td>J</td>
<td>010001</td>
<td>R</td>
<td>011001</td>
<td>Z</td>
<td>100001</td>
<td>h</td>
<td>101001</td>
<td>p</td>
<td>110001</td>
<td>x</td>
<td>111001</td>
<td>5</td>
</tr>
<tr>
<td>000010</td>
<td>C</td>
<td>001010</td>
<td>K</td>
<td>010010</td>
<td>S</td>
<td>011010</td>
<td>a</td>
<td>100010</td>
<td>i</td>
<td>101010</td>
<td>q</td>
<td>110010</td>
<td>y</td>
<td>111010</td>
<td>6</td>
</tr>
<tr>
<td>000011</td>
<td>D</td>
<td>001011</td>
<td>L</td>
<td>010011</td>
<td>T</td>
<td>011011</td>
<td>b</td>
<td>100011</td>
<td>j</td>
<td>101011</td>
<td>r</td>
<td>110011</td>
<td>z</td>
<td>111011</td>
<td>7</td>
</tr>
<tr>
<td>000100</td>
<td>E</td>
<td>001100</td>
<td>M</td>
<td>010100</td>
<td>U</td>
<td>011100</td>
<td>c</td>
<td>100100</td>
<td>k</td>
<td>101100</td>
<td>s</td>
<td>110100</td>
<td>0</td>
<td>111100</td>
<td>8</td>
</tr>
<tr>
<td>000101</td>
<td>F</td>
<td>001101</td>
<td>N</td>
<td>010101</td>
<td>V</td>
<td>011101</td>
<td>d</td>
<td>100101</td>
<td>l</td>
<td>101101</td>
<td>t</td>
<td>110101</td>
<td>1</td>
<td>111101</td>
<td>9</td>
</tr>
<tr>
<td>000110</td>
<td>G</td>
<td>001110</td>
<td>O</td>
<td>010110</td>
<td>W</td>
<td>011110</td>
<td>e</td>
<td>100110</td>
<td>m</td>
<td>101110</td>
<td>u</td>
<td>110110</td>
<td>2</td>
<td>111110</td>
<td>+</td>
</tr>
<tr>
<td>000111</td>
<td>H</td>
<td>001111</td>
<td>P</td>
<td>010111</td>
<td>X</td>
<td>011111</td>
<td>f</td>
<td>100111</td>
<td>n</td>
<td>101111</td>
<td>v</td>
<td>110111</td>
<td>3</td>
<td>111111</td>
<td>/</td>
</tr>
</tbody>
</table>
One-Time Pad Encryption

**Encryption.**

- Convert text message to N bits. [0 or 1]

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

message

base64
One-Time Pad Encryption

Encryption.

- Convert text message to $N$ bits.
- Generate $N$ random bits (one-time pad).

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

message

| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 |

base64

| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 |

random bits
**One-Time Pad Encryption**

**Encryption.**

- Convert text message to N bits.
- Use N random bits as one-time pad.
- Take bitwise XOR of two bitstrings.

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

**XOR Truth Table**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

sum corresponding pair of bits: 1 if sum is odd, 0 if even
One-Time Pad Encryption

Encryption.

- Convert text message to N bits.
- Use N random bits as one-time pad.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>w</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>message</th>
<th>base64</th>
<th>one-time pad</th>
<th>XOR</th>
<th>encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>E</td>
<td>N</td>
<td>D</td>
<td>M</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
</tr>
<tr>
<td>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>W</td>
</tr>
</tbody>
</table>
Secure Chat

Alice wants to send a secret message to Bob

• Can you read the secret message gX76W3v7K?
• But Bob can. How?
One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
</table>

encrypted
One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>W</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

encrypted

base64
One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
  - **Key point:** Bob and Alice agreed on the one-time pad beforehand

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

encrypted base64 random bits
One-Time Pad Decryption

Decryption:

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

encrypted
base64
one-time pad
XOR

\[ 1 \oplus 1 = 0 \]
One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text.

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Base64 encoding:

```
A000000
B000001
... ...
M001100
... ...
```

For example:

```
g  X  7  6  W  3  v  7  K
100000 010111 111011 111010 010110 110111 101111 111011 001010
110010 010011 110110 111001 011010 111001 100010 111111 010010
010010 000100 001101 000011 001100 001110 001101 000100 011000
```

Encrypted message:

```
g  X  7  6  W  3  v  7  K
```

Base64:

```
100000 010111 111011 111010 010110 110111 101111 111011 001010
110010 010011 110110 111001 011010 111001 100010 111111 010010
010010 000100 001101 000011 001100 001110 001101 000100 011000
```

One-time pad:

```
g  X  7  6  W  3  v  7  K
```

XOR:

```
S  E  N  D  M  O  N  E  Y
```

Message:

```
SEND MONEY
```

Magic?
Why Does It Work?

**Crucial property.** Decrypted message = original message.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>original message bit</td>
</tr>
<tr>
<td>b</td>
<td>one-time pad bit</td>
</tr>
<tr>
<td>^</td>
<td>XOR operator</td>
</tr>
<tr>
<td>a ^ b</td>
<td>encrypted message bit</td>
</tr>
<tr>
<td>(a ^ b) ^ b</td>
<td>decrypted message bit</td>
</tr>
</tbody>
</table>

Why is crucial property true?

- Use properties of XOR.
- \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR Truth Table
One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.

```
<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>
```

encrypted base64
One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
- Use **wrong** N bits (bogus one-time pad).

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

**encrypted**

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000</td>
<td>011100</td>
<td>110101</td>
<td>101111</td>
<td>010010</td>
<td>111001</td>
<td>100101</td>
<td>101010</td>
<td>001010</td>
</tr>
</tbody>
</table>

**base64**

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

**wrong bits**
One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
- Use **wrong** N bits (bogus one-time pad).
- Take bitwise XOR of two bitstrings.
**One-Time Pad Decryption (with the wrong pad)**

**Decryption.**
- Convert encrypted message to binary.
- Use **wrong** N bits (bogus one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text: **Oops**.

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

**encrypted**

<table>
<thead>
<tr>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000</td>
</tr>
</tbody>
</table>

**wrong bits**

<table>
<thead>
<tr>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001000</td>
</tr>
</tbody>
</table>

**wrong message**

| I | L | O | V | E | O | K | R | A |
Eve's Problem (one-time pads)

**Key point:** Without the pad, Eve cannot understand the message.

But Eve has a computer. Why not try all possible pads?

One problem: it might take a long time [stay tuned].

Worse problem: she would see all possible messages!

- 54 bits
- $2^{54}$ possible messages, all different.
- $2^{54}$ possible **encoded** messages, all different.
- No way for Eve to distinguish real message from any other message.

One-time pad is “provably secure”.

| AAAAAAAA | gX76w3v7K |
| AAAAAAAAB | gX76w3v7L |
| AAAAAAAAC | gX76w3v7I |
| ... | ... |
| ocltS5lqK | ILOVEOKRA |
| ... | ... |
| qwDgbDuav | Kn4aN0Bh1 |
| ... | ... |
| TTtpWk+1E | NEWTATTOO |
| ... | ... |
| yT25a5i/S | SENDMONEY |
| ... | ... |
| //////////+ | fo7FpIQE0 |
| ////////// | fo7FpIQE1 |
Goods and Bads of One-Time Pads

**Good.**

- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are truly random. [Shannon, 1940s]

**Bad.**

- (After a short break . . .)
COS 126 Overview

What is COS 126? Broad, but technical, introduction to computer science.

Goals.

• Demystify computer systems.
• Empower you to exploit available technology.
• Build awareness of substantial intellectual underpinnings.

Topics.

• Programming in Java.
• Machine architecture.
• Theory of computation.
• Applications to science, engineering, and commercial computing.

“Computers are incredibly fast, accurate, and stupid; humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination.” — Albert Einstein
The Basics

Lectures. [D. Clark]

Precepts. [Gabai · Pritchard · Beringer · Chaney · S. Clark · Funkhouser · Israel · Kang · Lee · C. Liu · S. Liu · Nadimpalli · Pai · Przytycki · Sepin · Song]
  • Tips on assignments; worked examples.
  • Questions on lecture material.
  • Informal and interactive.

Friend 016/017 lab. [Undergrad assistants]
  • Help with systems/debugging, not with course material.
  • Full schedule on Web (usually Sun--Fri evenings, Sat. afternoons)
  • Starts this week!

Website knows all: www.princeton.edu/~cos126
Grades

Course grades. No preset curve or quota.

9 programming assignments. 40%.
2 written exams (in lecture, 3/12 and 5/7). 35%.
2 programming exams (evenings, 3/14, 5/6). 15%.
Final programming project (due Dean's date - 1). 10%.
Extra credit / staff discretion. Adjust borderline cases.

participation helps, frequent absence hurts
Course Materials

Course website.  [www.princeton.edu/~cos126]

- Submit assignments.
- Programming assignments.
- Lecture slides.

(print before lecture) annotate during lecture

Course text.
Sedgewick and Wayne.

Intro to Programming in Java: An Interdisciplinary Approach.

Recommended reading (lectures 19-20).
Harel.

Computers Ltd.: What computers really can't do.

skim before lecture; read thoroughly afterwards
Programming Assignments

Desiderata.

- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!

N-body simulation
pluck a guitar string
estimate Avogadro's number
Programming Assignments

Desiderata.
• Address an important scientific or commercial problem.
• Illustrate the importance of a fundamental CS concept.
• You solve problem from scratch!

Due. Mondays 9 PM via Web submission.

Computing equipment.
• Your laptop. [OS X, Windows, Linux, iPhone, … ]
• OIT desktop. [Friend 016 and 017 labs]

Advice.
• Start early; plan multiple sessions.
• Seek help when needed. (Our job is to help you!)
• Use the Piazza online forum for Q&A about assignments, course material.
Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

Not registered? Go to any precept now; officially register ASAP.
Change precepts? Use SCORE.

Assignment 0. [www.princeton.edu/~cos126/assignments.php]
• Due Monday 9 PM.
• Read Sections 1.1 and 1.2 in textbook.
• Install Java programming environment + a few exercises.
• Lots of help available, don't be bashful.

END OF ADMINISTRATIVE STUFF
Goods and Bads of One-Time Pads

**Good.**
- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are truly random. [Shannon, 1940s]

**Bad.**
- Easily breakable if pad is re-used.
- Pad must be as long as the message.
- Truly random bits are very hard to come by.
- Pad must be distributed securely.

"one time" means one time only

impractical for Web commerce

a Russian one-time pad

eavesdropper Eve sees only random bits
Pseudo-Random Bit Generator

Practical middle-ground.

- Make a “random” bit generator gadget.
- Alice and Bob each get identical small gadgets.
- also, matching initial values, or “seeds,” for their gadgets

Goal. Small gadget that produces a long sequence of bits.
Pseudo-Random Bit Generator

Small deterministic gadgets that produce long sequences of pseudo-random bits:

- Enigma
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- [many others have been invented]

Pseudo-random? Bits are not really random:

- Bob’s and Alice’s gadgets must produce the same bits from the same seed.
- Bits must have as many properties of random bits as possible (to foil Eve).

Ex 1. approx 1/2 0s and 1/2 1s
Ex 2. approx 1/4 each of 00, 01, 10, 11

“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”

– John von Neumann (left)
– ENIAC (right)
Shift register terminology.

- **Bit**: 0 or 1.
- **Cell**: storage element that holds one bit.
- **Register**: sequence of cells.
- **Seed**: initial sequence of bits.
- **Shift register**: when clock ticks, bits propagate one position to left.
{8, 10} linear feedback shift register.

- Shift register with 11 cells.
- Bit $b_0$ is is XOR of previous bits $b_8$ and $b_{10}$.
- Pseudo-random bit = $b_0$. 
Linear Feedback Shift Register Demo

<table>
<thead>
<tr>
<th>Time 0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Time 2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Time 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Time 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Time 6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Time 7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time 8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Q. Are these 2000 numbers random?
If not, what is the pattern?

A. No. This is output of \{8, 10\} LFSR with seed 01101000010!
LFSR Encryption

Encryption.

- Convert text message to N bits.
- Initialize LFSR with given seed.
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

**Base64 Encoding**

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>w</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Message**

```
SENDEMY
```

**Base64**

```
BASE64
```

**LFSR Bits**

```
010010 000100 011101 000011 001100 001110 001101 000100 011000
110010 010011 110110 111001 011010 111001 100010 111111 010010
100000 010111 111011 111010 010110 110111 101111 111011 001010
```

**XOR**

```
g X 7 6 W 3 v 7 K
```

**Encrypted**

```
010010 000100 011101 000011 001100 001110 001101 000100 011000
110010 010011 110110 111001 011010 111001 100010 111111 010010
100000 010111 111011 111010 010110 110111 101111 111011 001010
```
LFSR Decryption

Decryption.

- Convert encrypted message to binary.
- Initialize identical LFSR with same seed.
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert back into text.

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010001</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>001000</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>S</td>
<td>E</td>
<td>N</td>
<td>D</td>
<td>M</td>
<td>O</td>
<td>N</td>
<td>E</td>
<td>Y</td>
</tr>
</tbody>
</table>

Encrypted: S E N D M O N E Y
Base64: ...
LFSR Bits: ...
XOR: ...
Message: SEND MONEY
Key properties of LFSRs

**Property 1:** A zero fill (all 0s) produces all 0s.
- Don’t use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

**Property 2:** Bitstream must eventually cycle.
- $2^N - 1$ nonzero fills in an N-bit register.
- Future output completely determined by current fill.

**Property 3:** Cycle length in an N-bit register is at most $2^N - 1$.
- Could be smaller; cycle length depends on tap positions.
- Need higher math (theory of finite groups) to know tap positions for given N.

**Bottom line:** 11-bit register generates at most 2047 bits before cycling, so use a longer register (say, $N = 61$).

Ex: $(1, 2)$ LFSR

```
001
010
101
011
111
110
100
001
```

$2^3 - 1 = 7$
Eve’s Problem (LFSR encryption/decryption)

Key point: Without the (short) seed
Eve cannot understand the (long) message.

But Eve has a computer. Why not try all possible seeds?
• Seeds are short, messages are long.
• All seeds give a tiny fraction of all messages.
• Extremely likely that all but real seed will produce gibberish.

Bad news (for Eve): There are still too many possibilities!
• Ex: 61-bit register implies $2^{61}$ possibilities.
• If Eve could check 1 million seeds per second,
  it would take her 730 centuries to try them all!

Exponential growth dwarfs technological improvements [stay tuned].
• 1000 bits: $2^{1000}$ possibilities.
• Age of the universe in microseconds: $2^{70}$
Goods and Bads of LFSRs

**Good.**

- Easily computed with simple machine.
- Very simple encryption/decryption processes.
- Bits have many of the same properties as random bits.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.
  
  [ but need theory of finite groups to know where to put taps ]

**Bad.**

- Still need secure, independent way to distribute LFSR seed.
- The bits are not truly random.
  
  [ bits in our 11-bit LFSR cycle after \(2^{11} - 1 = 2047\) steps]
- Experts have cracked LFSR encryption.
  
  [ need more complicated machines]
Other LFSR Applications

What else can we do with a LFSR?

- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

```c
/*          efdtt.c          Author:  Charles M. Hannum <root@ihack.net>          */
/*          Usage is:  cat title-key scrambled.vob | efdtt >clear.vob          */

#define m(i)(x[i]^s[i+84])<<

unsigned char x[5]      ,y,s[2048];main(n){for( read(0,x,5     );read(0,s ,n=2048
 ); write(1   ,s,n)         )if(s
 [y=s      [13]%8+20] /16%4 ==1      ){int
 i=m(      1)17  ^256 +m(0)   8,k      =m(2)
 0,j=      m(4)   17^ m(3)   9^k*     2-k%8
 ^8,a     =0,c    =26;for   (s[y]    -=16;
 --c;j   *=2<a=  a*2^i&  1,i=i /2^j&1
 <<24;for(j=        127;     ++j<n;c=c>
 y)   c
 +=y=i^i/8^i>>4^i>>12,
 i=i>>8<y<<17,a^=a>>14,y=a^a*8^a<<6,a=a
     >>8<y<<9,k=s[j] ,k
 >>=7]+2"^cr3sfw6v;*k+>/n."[k>4]*2^k*257/
 8,s[j]=k^+(k&k+2&34)*6^c+y
 ;})

http://www.cs.cmu.edu/~dst/DeCSS/Gallery
```
LFSR and "General Purpose Computer"

Important properties.

- Built from simple components.
- Scales to handle huge problems.
- Requires a deep understanding to use effectively.

<table>
<thead>
<tr>
<th>Basic Component</th>
<th>LFSR</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>start, stop, load</td>
<td>same</td>
</tr>
<tr>
<td>clock</td>
<td>regular pulse</td>
<td>2.8 GHz pulse</td>
</tr>
<tr>
<td>memory</td>
<td>11 bits</td>
<td>1 GB</td>
</tr>
<tr>
<td>input</td>
<td>seed</td>
<td>sequence of bits</td>
</tr>
<tr>
<td>computation</td>
<td>shift, XOR</td>
<td>logic, arithmetic, ...</td>
</tr>
<tr>
<td>output</td>
<td>pseudo-random bits</td>
<td>Sequence of bits</td>
</tr>
</tbody>
</table>

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.
A Profound Idea

Programming. Can write a Java program to simulate the operations of any abstract machine.

- Basis for theoretical understanding of computation. [stay tuned]
- Basis for bootstrapping real machines into existence. [stay tuned]

Stay tuned. See Assignment 5.

```java
public class LFSR {
    private int seed[];
    private final int tap;
    private final int N;

    public LFSR(String seed, int tap) { ... }

    public int step() { ... }

    public static void main(String[] args) {
        LFSR lfsr = new LFSR("01101000010", 8);
        for (int i = 0; i < 2000; i++)
            StdOut.print(lfsr.step());
    }
}
```

```
% java LFSR
1100100100111101101110010110101
1100110001011111101001000010011
0100101111001100100111...
```