General Computer Science
Princeton University
Spring 2014

Douglas Clark
0. Prologue: A Simple Machine
Secure Chat

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.

<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, ...

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)

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

Encryption.

• Convert text message to N bits. [0 or 1]
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>
<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>
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.

sum corresponding pair of bits: 1 if sum is odd, 0 if even
alternatively: 1 if bits are different, 0 if the same

<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>

message
\[x\]

base64
\[y\]

one-time pad
\[x \oplus y\]

XOR
\[0 \oplus 1 = 1\]
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.
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>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>

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>
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.

### 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>

```
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

base64

one-time pad

XOR

\[ 1 \ ^
\[ 1 = 0 \]
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>M</td>
<td>12</td>
<td>001100</td>
</tr>
</tbody>
</table>

### One-Time Pad Decryption

<table>
<thead>
<tr>
<th>Message</th>
<th>XOR</th>
<th>Base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>E</td>
<td>N</td>
</tr>
<tr>
<td>D</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>N</td>
<td>E</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encrypted</th>
<th>One-time pad</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>110010</td>
<td>g</td>
</tr>
<tr>
<td>010111</td>
<td>010011</td>
<td>X</td>
</tr>
<tr>
<td>111011</td>
<td>110110</td>
<td>7</td>
</tr>
<tr>
<td>111010</td>
<td>010110</td>
<td>6</td>
</tr>
<tr>
<td>110111</td>
<td>110111</td>
<td>W</td>
</tr>
<tr>
<td>101111</td>
<td>100111</td>
<td>3</td>
</tr>
<tr>
<td>111011</td>
<td>001100</td>
<td>v</td>
</tr>
<tr>
<td>001100</td>
<td>011000</td>
<td>7</td>
</tr>
<tr>
<td>001101</td>
<td>111001</td>
<td>K</td>
</tr>
<tr>
<td>001000</td>
<td>110010</td>
<td>encrypt</td>
</tr>
<tr>
<td>001101</td>
<td>111010</td>
<td>base64</td>
</tr>
<tr>
<td>001100</td>
<td>111010</td>
<td>one-time</td>
</tr>
<tr>
<td>001110</td>
<td>100010</td>
<td>pad</td>
</tr>
<tr>
<td>001110</td>
<td>111111</td>
<td>XOR</td>
</tr>
<tr>
<td>001101</td>
<td>010010</td>
<td>message</td>
</tr>
<tr>
<td>001010</td>
<td>111011</td>
<td>magic?</td>
</tr>
</tbody>
</table>
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.
• \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

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>
One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.

```
g  x  7  6  W  3  v  7  K
```
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>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>
<td></td>
</tr>
</tbody>
</table>

encrypted

base64

wrong bits
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)

<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>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>
<tr>
<td>001000</td>
<td>001011</td>
<td>001110</td>
<td>010101</td>
<td>000100</td>
<td>001110</td>
<td>001010</td>
<td>010001</td>
<td>000000</td>
</tr>
</tbody>
</table>

**encrypted**

**base64**

**wrong bits**

**XOR**
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>
<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>
<tr>
<td>001000</td>
<td>001011</td>
<td>001110</td>
<td>010101</td>
<td>000100</td>
<td>001110</td>
<td>001010</td>
<td>010001</td>
<td>000000</td>
</tr>
</tbody>
</table>

**encrypted**

**base64**

**wrong bits**

**XOR**

**wrong message**
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”.

IF pad is random and used only once

<table>
<thead>
<tr>
<th>Pad</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAA</td>
<td>gX76W3v7K</td>
</tr>
<tr>
<td>AAAAAAAB</td>
<td>gX76W3v7L</td>
</tr>
<tr>
<td>AAAAAAAC</td>
<td>gX76W3v7I</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ocltS51qK</td>
<td>ILOVEOKRA</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>qwDgbDuav</td>
<td>Kn4aN0Bhl</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>tTtpWk+1E</td>
<td>NEWTATTOO</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>yT25a5i/S</td>
<td>SENDMONEY</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>//////////+</td>
<td>fo7FpIQE0</td>
</tr>
<tr>
<td>//////////</td>
<td>fo7FpIQE1</td>
</tr>
</tbody>
</table>
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 . . .)
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.** [Clark]

**Precepts.** [Pritchard · Ararat · Boyko · Chen · Fan · Gabai · Ghasemi · Ginsburg · Hristov · Israel · Kang · Lee · Shi · Song · Vithanage · Wang · Yang · Zhao]

- Tips on assignments, worked examples, clarify lecture material.
- Informal and interactive.

**Friend 016/017 lab.**

- Undergraduate lab assistants.
- Help with systems and debugging.

**Piazza.** [online discussion]

- Best chance for quick response to a question.
- Post to class or via private post to staff.

**Website knows all:** [www.princeton.edu/~cos126](http://www.princeton.edu/~cos126)
Grades

Course grades. No preset “curve” or quota.

9 programming assignments. 40%.
2 written exams (in lecture, midterm week & last week). 35%.
2 programming exams (evenings, same weeks). 15%.
Final programming project (due Dean’s date - 1). 10%.
Course Materials

Course website. [www.princeton.edu/~cos126]
• Submit assignments.
• Programming assignments.
• Lecture slides
• “Booksite”.
  • Summary of course content.
  • Code, exercises, examples.
  • Supplementary material.
  • NOT the same as Text
  • for use while online

Course text. [Sedgewick and Wayne]
• Full introduction to course material
• Developed for this course
• For use while learning and studying

Recommended reading (lectures 19-20). [Harel]
Desiderata.

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

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

Due. Mondays midnight 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.
What's Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

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

Assignment 0.
• Due Monday midnight.
• Read Sections 1.1 and 1.2 in textbook.
• Install Java programming environment (find directions in Assignment 0)
• Lots of help available, don't be bashful.

see Colleen Kenny-McGinley in CS 210 only
if the only precept time you can attend is closed

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]

  - eavesdropper Eve sees only random bits

  "one time" means one time only!

**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.

  - impractical for Web commerce

  a Russian one-time pad
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. approximately 1/2 0s and 1/2 1s
Ex 2. approximately 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.

![Shift Register Diagram]
Linear Feedback Shift Register (LFSR)

\{8, 10\} linear feedback shift register.

- Shift register with 11 cells.
- Bit \(b_0\) is XOR of previous bits \(b_8\) and \(b_{10}\).
- Pseudo-random bit = (new) \(b_0\).
### 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>0</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>
Random Numbers

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

<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>

### base64

<table>
<thead>
<tr>
<th>LFSR bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>110010</td>
</tr>
<tr>
<td>010011</td>
</tr>
<tr>
<td>110110</td>
</tr>
<tr>
<td>111001</td>
</tr>
<tr>
<td>011010</td>
</tr>
<tr>
<td>111001</td>
</tr>
<tr>
<td>100010</td>
</tr>
<tr>
<td>111111</td>
</tr>
<tr>
<td>010010</td>
</tr>
</tbody>
</table>

### XOR

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

S E N D M O N E Y
Key properties of LFSRs

**Property 1:** A zero fill (all 0s) produces all 0s.
- So 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).

challenge for the bored: what tap positions?
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): Alice and Bob can use a much larger LFSR.

- For instance: 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] /164  ==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=<<8^y<17,a^a>>14,y=a^a*8^a<<6,a=a
 >>8^y<<9,k=s[j],k
 ="7Wo~'G_\216"[k
 &7]+2"cr3sfw6v;*k++/n."[k>>4]*2^k257/
 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...

`