# COS 126

General Computer Science
Princeton University
Spring 2013

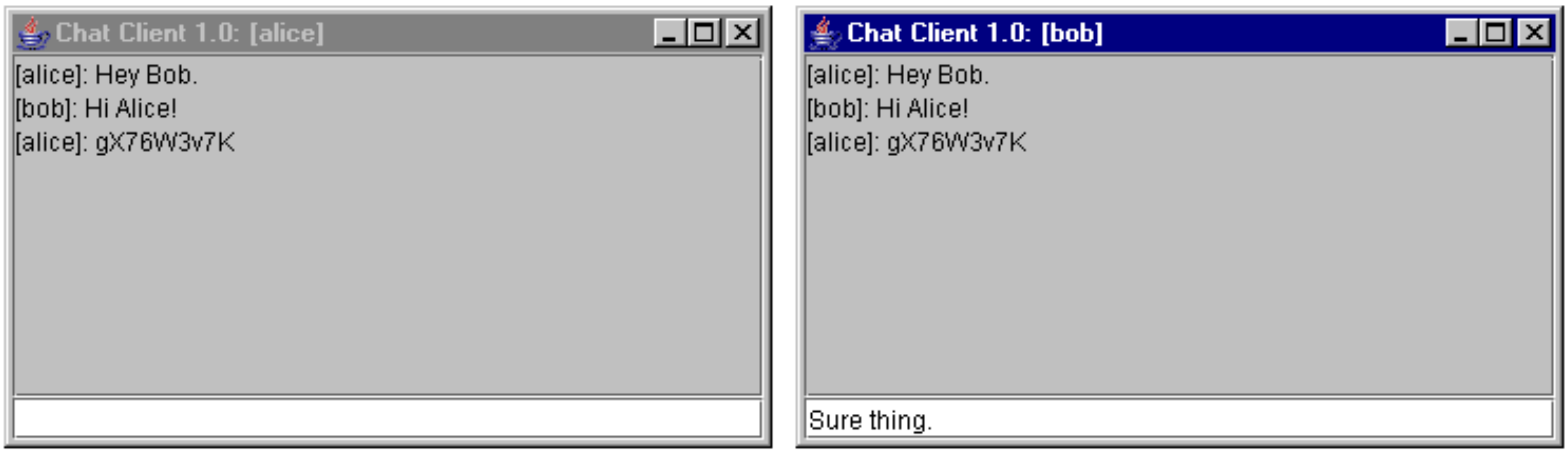
Douglas Clark

# 0. Prologue: A Simple Machine

# Secure Chat

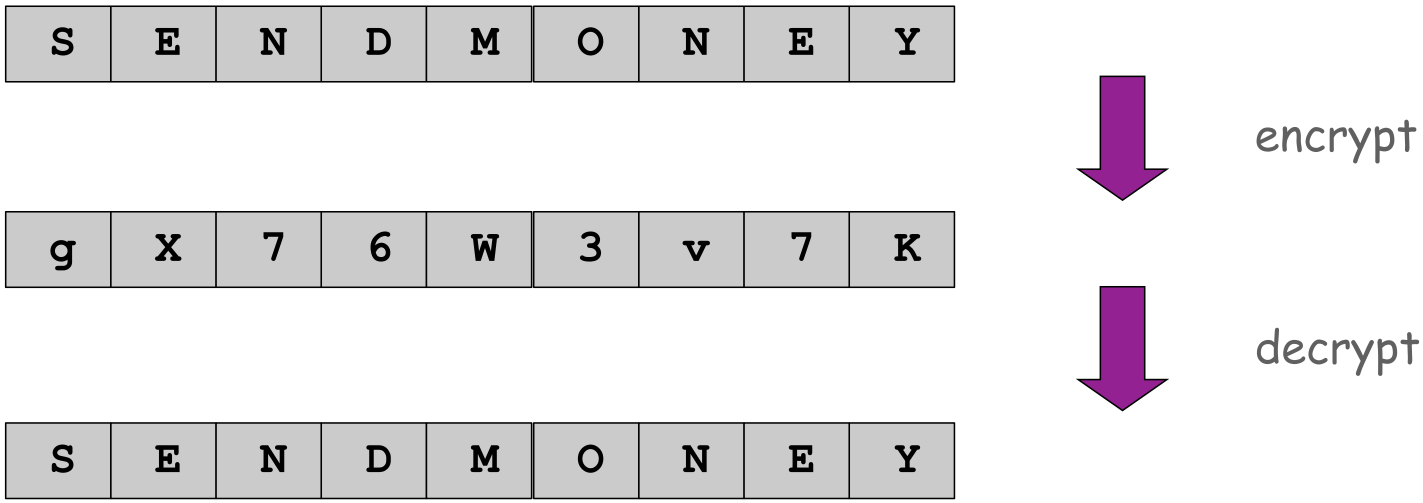
## Alice wants to send a secret message to Bob

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



# Encryption Machine

Goal. Design a machine to encrypt and decrypt data.



# Encryption Machine

Goal. Design a machine to encrypt and decrypt data.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y |

↓ encrypt

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K |

↓ decrypt

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y |

Enigma encryption machine.

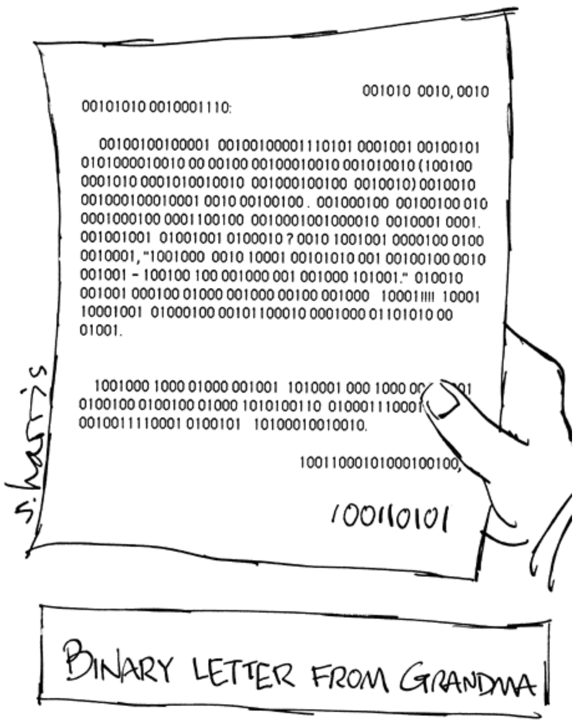
- "Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.

# A Digital World

Data is a sequence of bits. [bit = 0 or 1] ← can use decimal digits, letters, or some other system, but bits are more easily encoded physically ("on-off", "up-down", "hot-cold",...)

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

thousands of bits




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

billions of bits

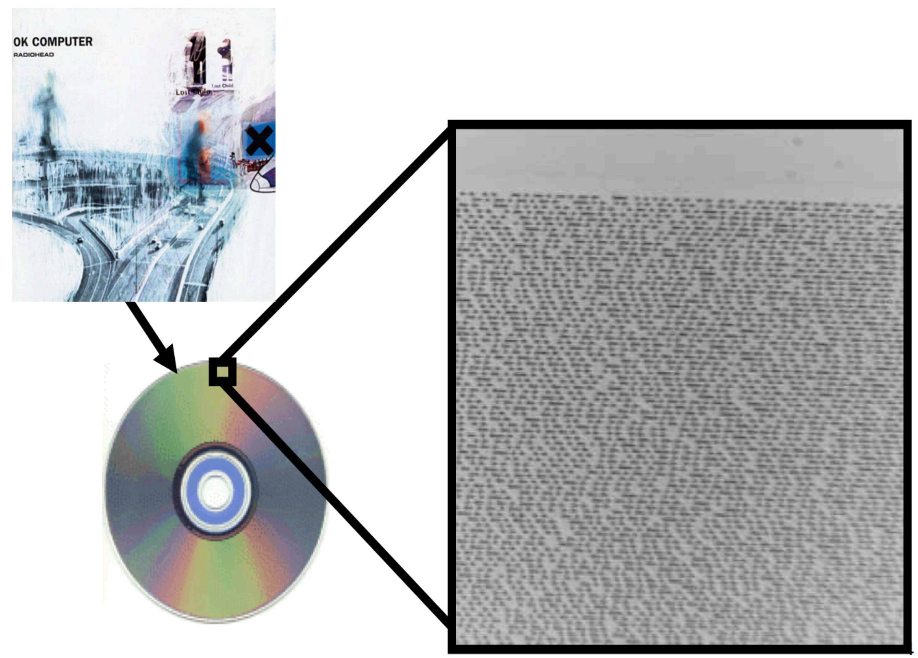



image courtesy of David August

# 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]

Base64 Encoding

| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| M | 12 | 001100 |
| … | … | … |

| | S | E | N | D | M | O | N | E | Y |
|---|---|---|---|---|---|---|---|---|---|
| message | S | E | N | D | M | O | N | E | Y |
| base64 | 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 |

# One-Time Pad Encryption

Encryption.

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y | 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.

sum corresponding pair of bits: 1 if sum is odd, 0 if even

XOR Truth Table

| x | y | x ^ y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y | message |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | one-time pad |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | XOR |

0 ^ 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.

Base64 Encoding

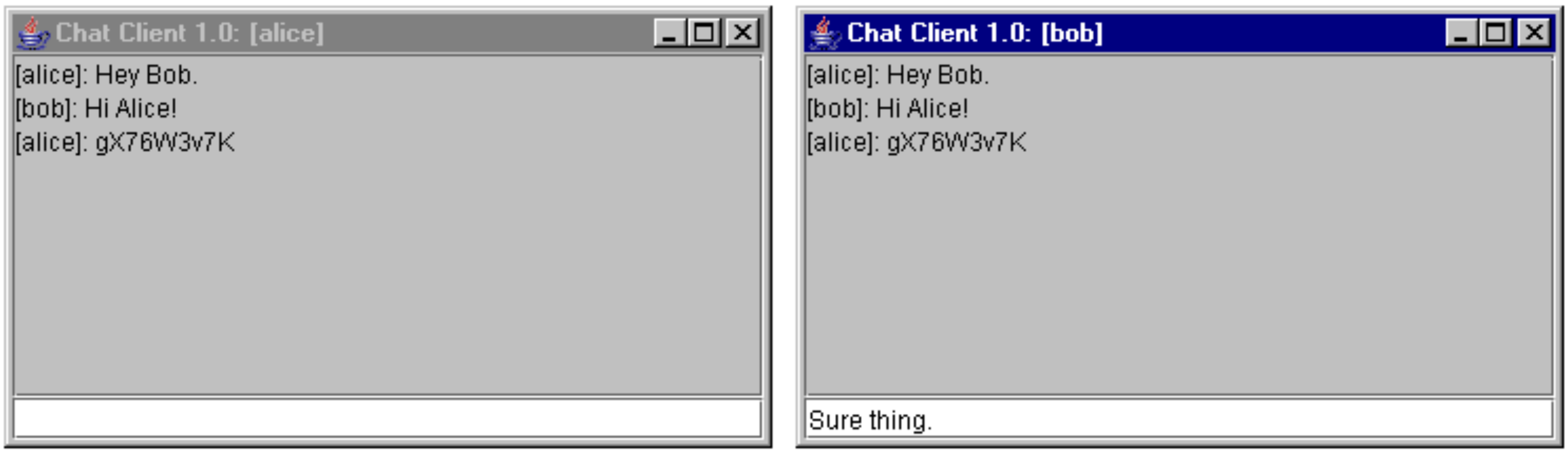
| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| W | 22 | 010110 |
| … | … | … |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y | message |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | one-time pad |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | XOR |
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |

# 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.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |

# One-Time Pad Decryption

Base64 Encoding

| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| W | 22 | 010110 |
| … | … | … |

**Decryption.**

- Convert encrypted message to binary.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | 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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | 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

| x | y | x ^ y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | one-time pad |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | XOR |

1 ^ 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

| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| M | 12 | 001100 |
| … | … | … |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | one-time pad |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | XOR |
| S | E | N | D | M | O | N | E | Y | message |

magic?

# Why Does It Work?

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

| Notation | Meaning |
|---|---|
| a | original message bit |
| b | one-time pad bit |
| ^ | XOR operator |
| a ^ b | encrypted message bit |
| (a ^ b) ^ b | decrypted message bit |

XOR Truth Table

| x | y | x ^ y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Why is crucial property true?**

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

associativity of ^

always 0

identity

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

Decryption.

- Convert encrypted message to binary.

| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
|---|---|---|---|---|---|---|---|---|---|

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

**Decryption.**

- Convert encrypted message to binary.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |

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

**Decryption.**

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 101000 | 011100 | 110101 | 101111 | 010010 | 111001 | 100101 | 101010 | 001010 | 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.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 101000 | 011100 | 110101 | 101111 | 010010 | 111001 | 100101 | 101010 | 001010 | wrong bits |
| 001000 | 001011 | 001110 | 010101 | 000100 | 001110 | 001010 | 010001 | 000000 | 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.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 101000 | 011100 | 110101 | 101111 | 010010 | 111001 | 100101 | 101010 | 001010 | wrong bits |
| 001000 | 001011 | 001110 | 010101 | 000100 | 001110 | 001010 | 010001 | 000000 | XOR |
| I | L | O | V | E | O | K | R | A | 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".

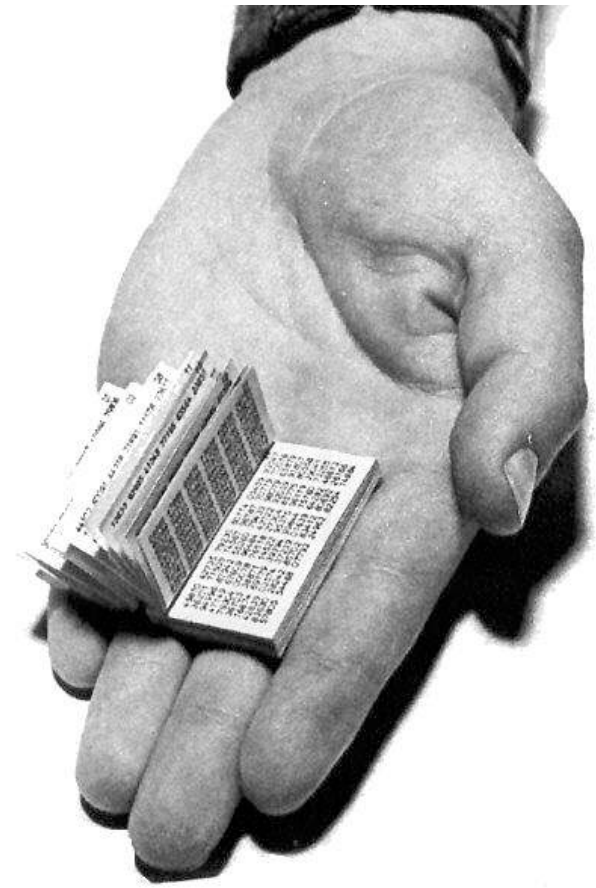
| | |
|---|---|
| AAAAAAAAA | gX76W3v7K |
| AAAAAAAAB | gX76W3v7L |
| AAAAAAAAC | gX76W3v7I |
| ... | |
| oc1tS5lqK | ILOVEOKRA |
| ... | |
| qwDgbDuav | Kn4aN0Bhl |
| ... | |
| tTtpWk+1E | NEWTATTOO |
| ... | |
| yT25a5i/S | SENDMONEY |
| ... | |
| ////////+ | fo7FpIQE0 |
| ///////// | fo7FpIQE1 |

# 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]

eavesdropper Eve sees only random bits



a Russian one-time pad

Bad.

- (After a short break . . .)

# COS 126 Overview

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

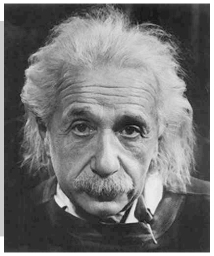
Goals.

- 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%.

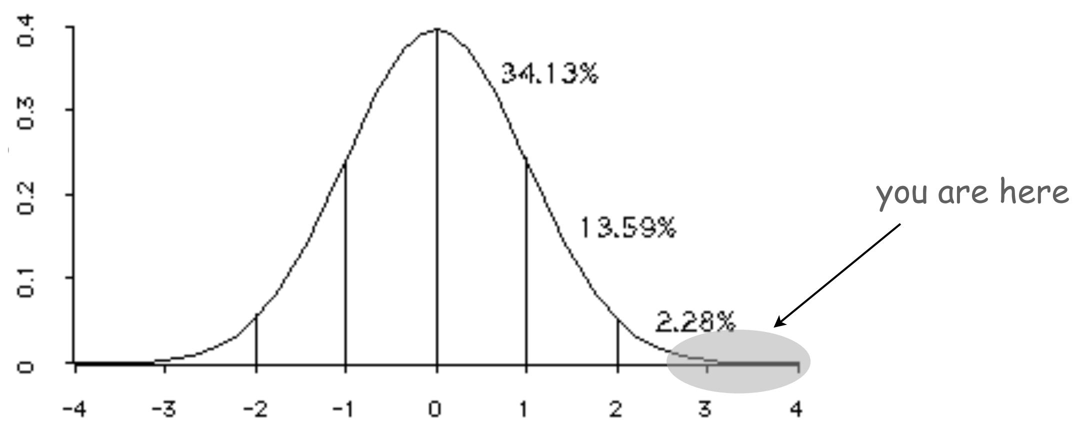
2 written exams (in lecture, 3/12 and 5/7). 35%.

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.

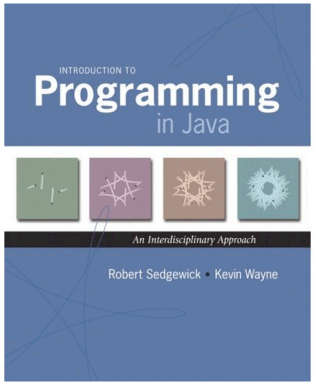
(print before lecture)
annotate during lecture

**Course text.**

Sedgewick and Wayne.

Intro to Programming in Java: *An Interdisciplinary Approach*.

skim before lecture;
read thoroughly
afterwards




**Recommended reading (lectures 19-20).**

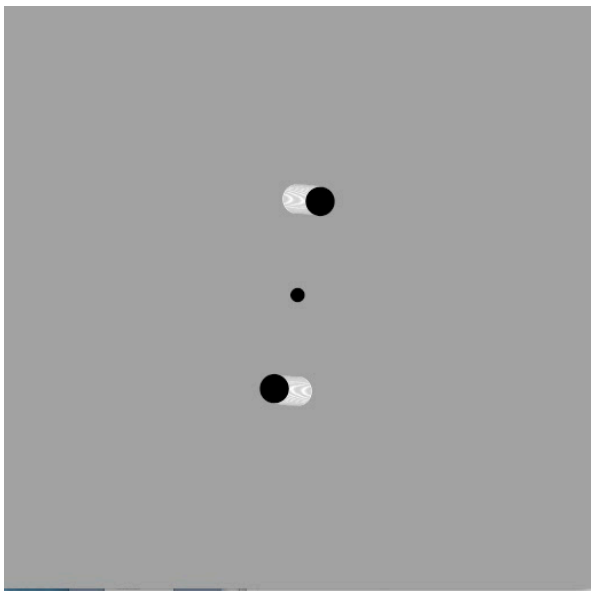
Harel.

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

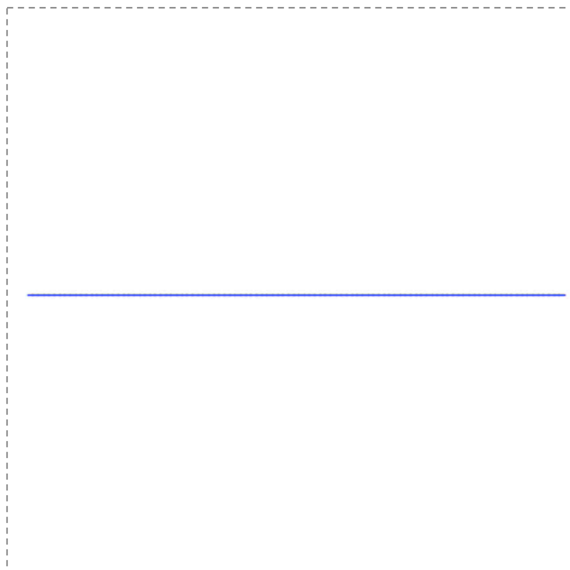
# Programming Assignments

Desiderata.

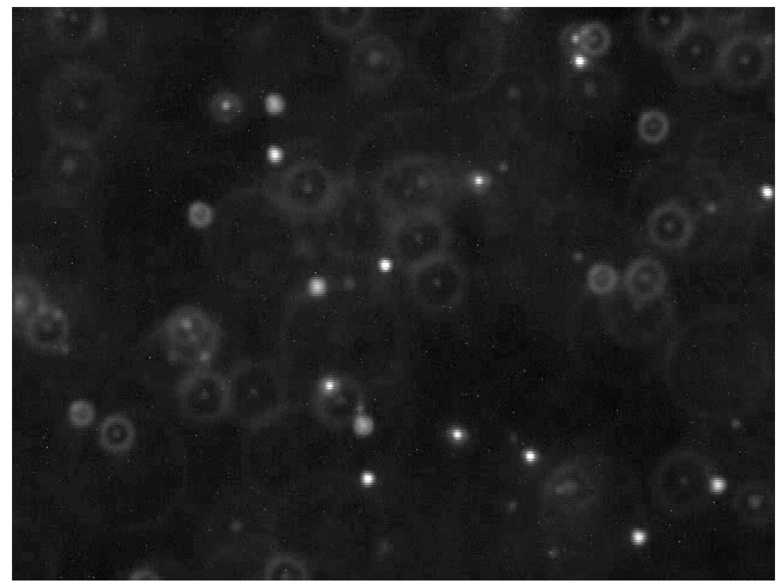
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!



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.

# What's Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

Not registered? Go to any precept now; officially register ASAP.

Change precepts? Use SCORE.

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

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]

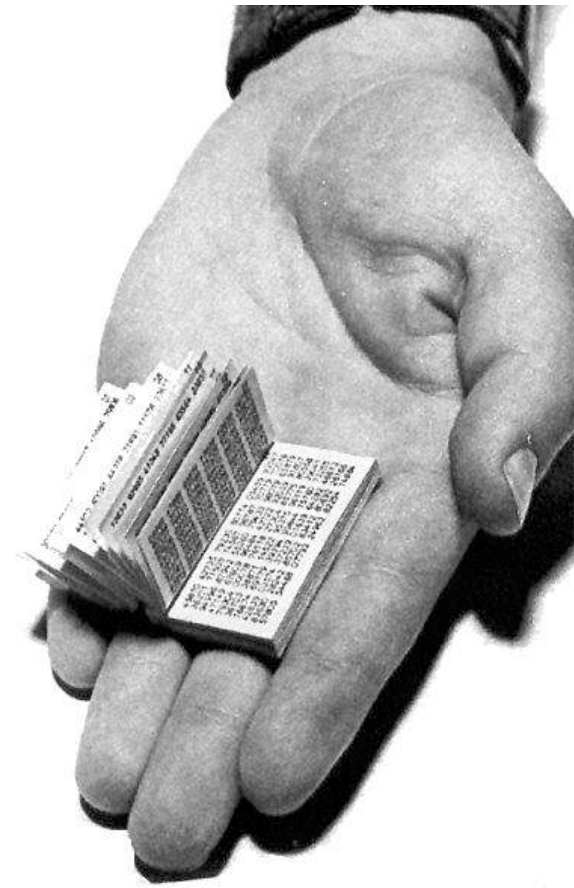
eavesdropper Eve sees only random bits

"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. ← instead of identical large one-time pads
- 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:

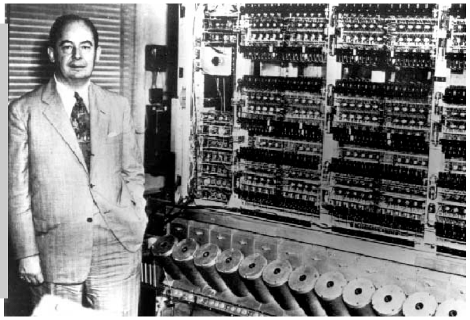
- Enigma
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- [many others have been invented]

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

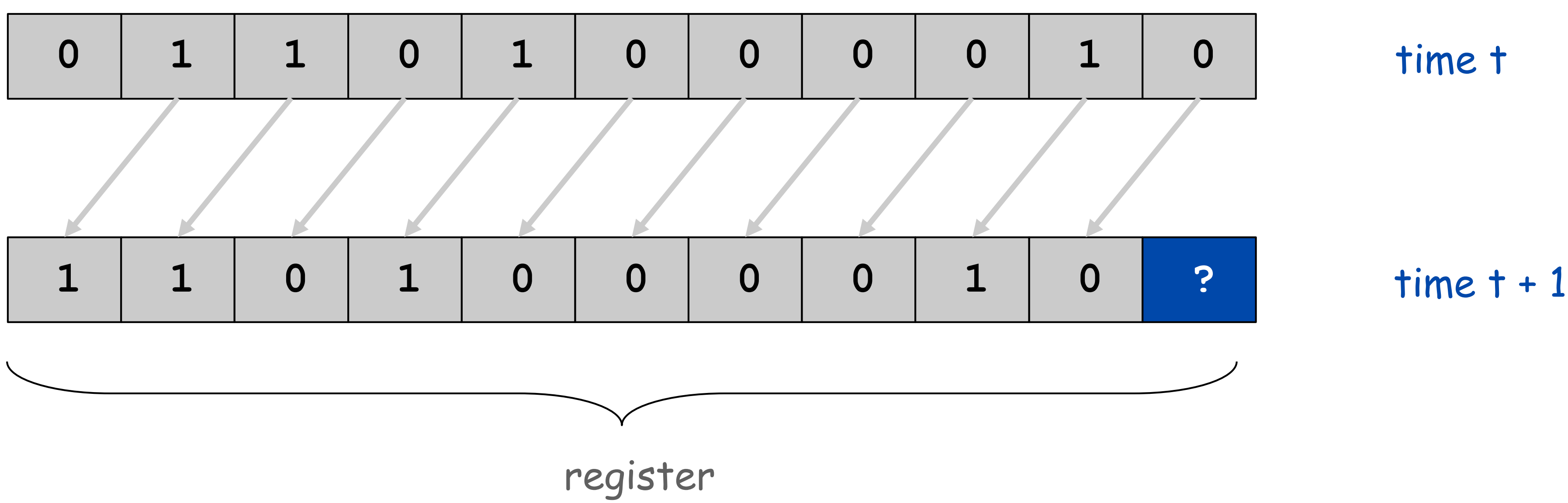
> *" Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. "*
> *– John von Neumann (left)*
> *– ENIAC (right)*



# Shift Register

Shift register terminology.

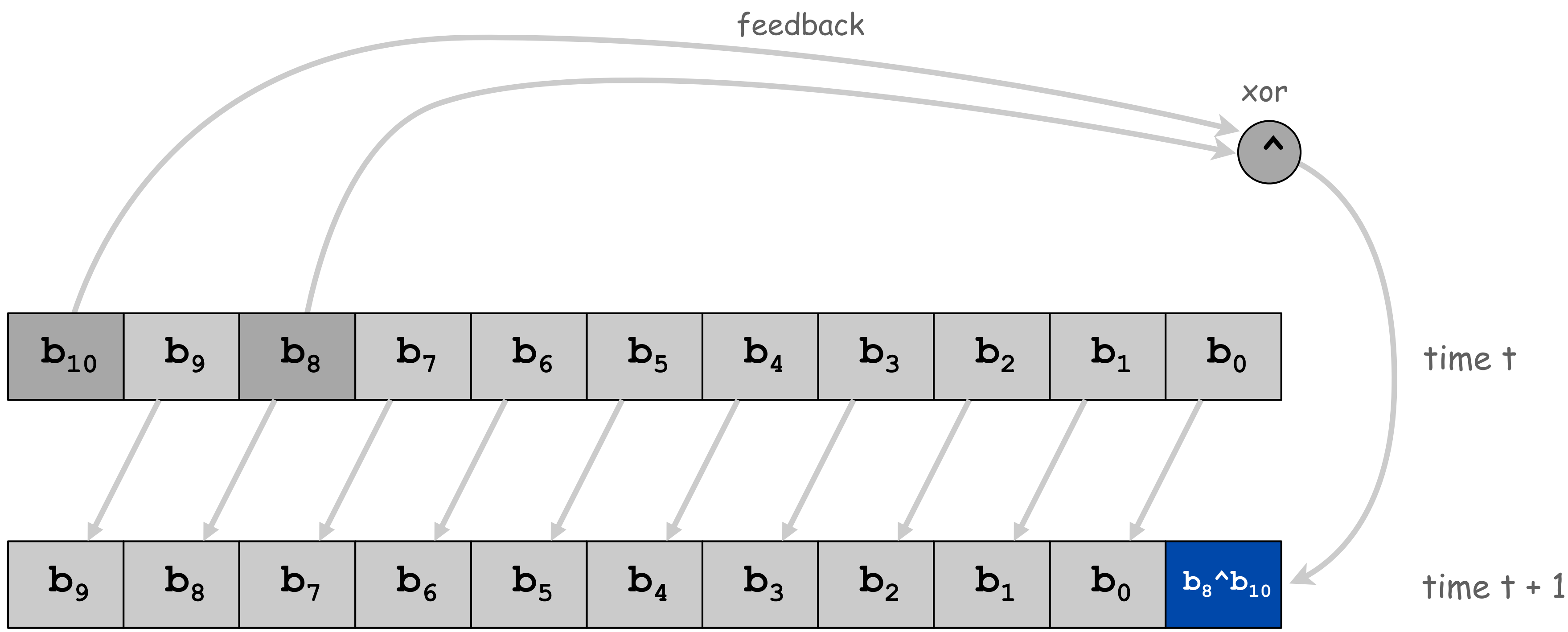
- Bit: 0 or 1.
- Cell: storage element that holds one bit.
- Register: sequence of cells.
- Seed: initial sequence of bits.
- Shift register: when clock ticks, bits propagate one position to left.



# Linear Feedback Shift Register (LFSR)

{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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | Time 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Time 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | Time 2 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | Time 3 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | Time 4 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | Time 5 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | Time 6 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | Time 7 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | Time 8 |

# Random Numbers


```
255 000s
256 001s
255 010s
256 011s
...
```

**Q.** Are these 2000 numbers random?

If not, what is the pattern?

```
1100100100111101101110010110101110011000101111110100100001001101001011110011001001111111011100
0001010110001000011101010011010000111100100110011101111111010100000100001000101001010100011000
0010111100010010011010110111100011010011011100111101011110010001001110101011101000001010010001
0001101010101110000000101100000100111000101110110100101011001100001111111001100000111111000110
0001101111001110100111101001110010011101110111010101010100000000000100000000101000000100010000 1
010101001000000001101000000111001000110111010111010100010100001010001001000101011010100001100001
0011110010111001110010111101110010010101110110000101011100100001011101001001010011011000111101
1101100101010111100000010011000010111110010010001110110101101011000110001110111101101010010110
0001100111001111110111100001010011001000111111010110000100011100101011011100001101011001110001
1111011011000101101110100110101001111000011100110011011111111101000000001001000001011010001001 1
0010101111110000100001100101001111100011100011011011011101101101010110110000011011100011101011
0110100011011001011101111001010100111000001110110001101011101110001010101101000000011001000011 1
1101001100010011111010111000100010110101010011000000111110000110001100111101111110010100001110
0010011011010111101100010010111010110010100011110001011001101001111110011100000111011001100101
1111111001000000111010000110100100111001101110111110101010001000000101010000100000100101000101
1000101001110100011101001011010011001100111111111111000000000110000000111100000110011000111111 1
1011000000101110000100101100101100111100111110011110001111001101100111110111110001010001101000
1011100101001011100011001011011111001101000111110010110001110011101101111010110100100011001101
0111111100010000011010100011100001011011001001101111011110100101001001100011011111011101000101
0100101000001100010001111010101100100000111101000110010010111110110010001011110101001001000011
0110100111011001110101111101000100010010101010110000000001110000000110110000111011100110101011 11
10000010001100010101111010
```

**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

| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| W | 22 | 010110 |
| … | … | … |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | E | N | D | M | O | N | E | Y | message |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | LFSR bits |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | XOR |
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |

# LFSR Decryption

Decryption.

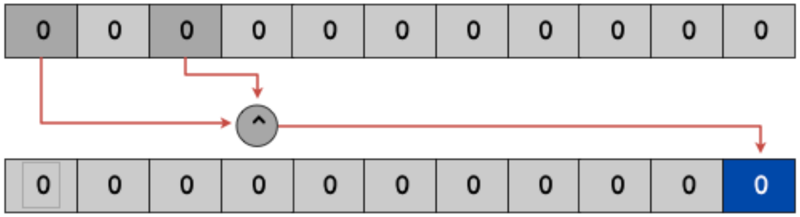
- Convert encrypted message to binary.
- Initialize identical LFSR with same seed
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert back into text.

Base64 Encoding

| char | dec | binary |
|---|---|---|
| A | 0 | 000000 |
| B | 1 | 000001 |
| … | … | … |
| M | 12 | 001100 |
| … | … | … |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g | X | 7 | 6 | W | 3 | v | 7 | K | encrypted |
| 100000 | 010111 | 111011 | 111010 | 010110 | 110111 | 101111 | 111011 | 001010 | base64 |
| 110010 | 010011 | 110110 | 111001 | 011010 | 111001 | 100010 | 111111 | 010010 | LFSR bits |
| 010010 | 000100 | 001101 | 000011 | 001100 | 001110 | 001101 | 000100 | 011000 | XOR |
| S | E | N | D | M | O | N | E | Y | message |

# 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.

Ex: (1, 2) LFSR

001
010
101
011
111
110
100
001 $2^3-1 = 7$

**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?

assume Eve has a machine (knows LFSR length and taps)

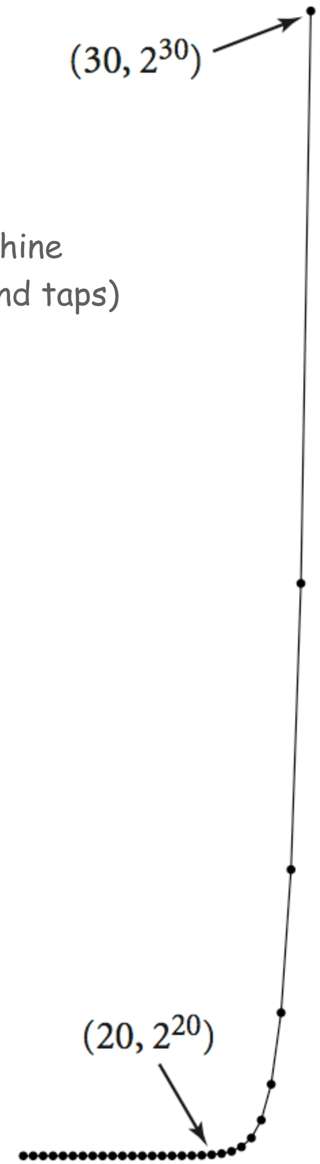
- Seeds are short, messages are long.
- All seeds give a tiny fraction of all messages.
- Extremely likely that all but real seed will produce gibberish.

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 ]


a commercially available LFSR

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]

## What else can we do with a LFSR?

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

DVD Jon
(Norwegian hacker)

```
/*     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         ="7Wo~'G_\216"[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.

| Basic Component | LFSR | Computer |
|---|---|---|
| control | start, stop, load | same |
| clock | regular pulse | 2.8 GHz pulse |
| memory | 11 bits | 1 GB |
| input | seed | sequence of bits |
| computation | shift, XOR | logic, arithmetic, … |
| output | pseudo-random bits | Sequence of bits |

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.

```
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
11001001001111011011100101101011
11001100010111111010010000100011
010010111100110010011 1...
```