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.

G X 7 6 W 3 v 7 K

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]

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

Thousand of bits

Billions of bits

![Image of paper and CD]

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

One-Time Pad Encryption

Encryption.
- Convert text message to N bits. [0 or 1]
- Generate N random bits (one-time pad).

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>
</tbody>
</table>

message

SENDMONYE

010010 000100 001101 000011 001110 001101 000100 011000

base64

010010 000100 001101 000011 001110 001101 000100 011000

base64

011001 010111 111011 011010 111001 011011 011011 011011 011011 011011

random bits

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

XOR Truth Table
\[
\begin{array}{ccc}
  x & y & x \oplus y \\
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]

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>001001</td>
<td>001110</td>
<td>001101</td>
<td>001100</td>
<td>000100</td>
<td>011001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110011</td>
<td>110011</td>
<td>111001</td>
<td>111010</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>011101</td>
<td>110111</td>
<td>001010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ x \oplus y = 1 \]

Secure Chat

Alice wants to send a secret message to Bob
• Can you read the secret message \texttt{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>
<tbody>
<tr>
<td>010010</td>
<td>001001</td>
<td>001110</td>
<td>001101</td>
<td>001100</td>
<td>000100</td>
<td>011001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110011</td>
<td>110011</td>
<td>111001</td>
<td>111010</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>011101</td>
<td>110111</td>
<td>001010</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>base64 Encoding</th>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>22</td>
<td>010110</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>encrypted</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000 010111 111011 111010 010110 110111 110111 110111 001010</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR Truth Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>encrypted</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000 010111 111011 111010 010110 110111 110111 110111 001010</td>
<td></td>
</tr>
<tr>
<td>110010 010011 110110 111001 011010 110011 110011 100010 111111 010010</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR</th>
<th>one-time pad</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>000000000000</td>
</tr>
<tr>
<td>Y</td>
<td>000001111111</td>
</tr>
<tr>
<td>XOR</td>
<td>001010101010</td>
</tr>
</tbody>
</table>

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

| Base64 Encoding |
|-----------------|------|-----|--------|
| A               | 0    | 000000 |
| B               | 1    | 000001 |
| M               | 12   | 001100 |

<table>
<thead>
<tr>
<th>encrypted</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000 010111 111011 111010 010110 110111 110111 110111 001010</td>
<td></td>
</tr>
<tr>
<td>110010 010011 110110 111001 011010 110011 110011 100010 111111 010010</td>
<td></td>
</tr>
<tr>
<td>010010 001000 001101 000011 001100 001110 001101 000100 011000</td>
<td></td>
</tr>
<tr>
<td>magic?</td>
<td></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.
- \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

One-Time Pad Decryption (with the wrong pad)

Decryption.
- Convert encrypted message to binary.

\[
\begin{array}{cccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\
\hline
100000 & 010111 & 111011 & 111010 & 010110 & 110111 & 101111 & 111011 & 001010 \\
\end{array}
\]

One-Time Pad Decryption (with the wrong pad)

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

\[
\begin{array}{cccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\
\hline
100000 & 010111 & 111011 & 111010 & 010110 & 110111 & 101111 & 111011 & 001010 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\
\hline
101000 & 011100 & 110101 & 101111 & 010010 & 111001 & 100101 & 101010 & 001010 \\
\end{array}
\]
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.

<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>111011</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>101000</td>
<td>011100</td>
<td>111010</td>
<td>101111</td>
<td>100101</td>
<td>101010</td>
<td>001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001000</td>
<td>001011</td>
<td>001110</td>
<td>010101</td>
<td>001010</td>
<td>001110</td>
<td>001010</td>
<td>010001</td>
<td>000000</td>
</tr>
</tbody>
</table>

encrypted

base64

100000 010111 111011 111010 010110 110111 111011 111011 001010
101000 011100 111010 101111 100101 101010 001010
001000 001011 001110 010101 001010 001110 001010 010001 000000

wrong bits

XOR

I L O V E O K R A

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.

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

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.** [Clark]

**Precepts and grading.** [Gabai · Ginsburg · Leyzberg · Ararat · Cook · Frankle · Goldfeder · Gossels · Grover · Hasdemir · Jones · Kaplan · Li · Petersen · Song · Ulus · Yang]
- 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.

---

**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). 15%.
- Final programming project (due Dean’s date - 1). 10%.

---

**Course Materials**

**Course website.** [www.princeton.edu/~cos126]
- Submit assignments.
- Programming assignments.
- Lecture slides → (print before lecture) annotate during lecture
  - "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 18-19).** [Harel]
Programming Assignments

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.

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.

N-body simulation

pluck a guitar string

estimate Avogadro’s number

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 TigerHub.

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.

END OF ADMINISTRATIVE STUFF
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.

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.

<table>
<thead>
<tr>
<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>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>?</td>
</tr>
</tbody>
</table>

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) $b_0$.

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>
Linear Feedback Shift Register Demo

"seed" = initial contents

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>
Linear Feedback Shift Register Demo

0 1 1 0 1 0 0 0 0 1 0  

Time 0

0 1 1 0 1 1 1  

Time 1

0 1 1 0 1 1  

Time 2

0 0 0 0 1 0 1 1 0 0  

Time 3

0 0 0 0 1 1 0 0 1 0  

Time 4

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>

45
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.

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.

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]

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></td>
<td></td>
<td>sequence of bits</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.

What else can we do with a LFSR?
- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

/*       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, y = m(4) 17 * m(3) 9 * k ^ 2-k
--y; --y; }
    a = a*2^14, 1, i/2^5*41
    <<24; for(y=127;
    i=177, y)
    c=s+y*c+m*26,for (y) y+=
    c;i++=y=a^a^y>>4^y>>12,
    y+=y+c[m/8]=s[17],k=x++y=
    s[13]^2+c[m/8]^2+c[257]
    % java LFSR
    1100100100111101101110010110101
    1100110001011111101001000010011
    0100101111001100100111...

  System.out.println(lfsr.step());
```

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.