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.

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, ...

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

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)

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

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).

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>

message

S E N D M O N E Y

010010 000100 001101 000011 001110 001101 000100 011000

base64

S E N D M O N E Y

010010 000100 001101 000011 001110 001101 000100 011000

random bits

110010 010011 111010 011010 111001 100010 111111 010010
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>
</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>110010</td>
<td>010011</td>
<td>111010</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</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>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>011010</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

Message: SENDME Y
Base64: 010010 000100 001101 000011 001100 001110 001101 000100 011000

XOR:

<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>

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>

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>w</td>
<td>22</td>
<td>010110</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>
</table>
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>W</td>
<td>22</td>
<td>010110</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>

encrypted

base64

10000 010111 111011 111010 010110 110111 111011 001010

110010 010011 110110 111011 011010 110111 111011 001010

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.
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.
  - \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)
  - Associativity of ^
  - Identity

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.

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

base64

One-Time Pad Decryption (with the wrong pad)

Decryption.

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

\[
\begin{array}{cccccccc}
\text{g} & \text{X} & 7 & 6 & \text{W} & 3 & \nu & 7 & \text{K} \\
100000 & 010111 & 111011 & 111010 & 010110 & 110111 & 101111 & 111011 & 001010 \\
101000 & 011100 & 110101 & 101111 & 010010 & 111001 & 100101 & 101010 & 001010 \\
\end{array}
\]

base64

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.

<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>encrypted base64</td>
<td>100000 010111 111011 111010 010110 110111 111011 111010 001010</td>
<td>101000 011100 110101 101111 010101 110101 101010 001010</td>
<td>001000 001011 001110 010101 000100 001110 001010 010001 000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.**

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”.  

**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. [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

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 (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 19-20). [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.

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.

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
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.

\[
\begin{array}{cccccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

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**

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

**LFSR 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.

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>00000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>w</td>
<td>22</td>
<td>010110</td>
</tr>
</tbody>
</table>

**LFSR 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.

**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!
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.
- $2^N$ 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).

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 microseonds: $2^{70}$

Other LFSR Applications

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

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.

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.

[ but need theory of finite groups to know where to put taps ]

[a commercially available LFSR]

[ 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.

**Critical difference.** General purpose machine can be programmed to simulate ANY abstract machine.

<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>

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) { ... }
}
```

% java LFSR
```plaintext
010010000101111110100100001001110100101111001100100111...
```