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.

SEND MONEY

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

encrypt

decrypt
**Encryption Machine**

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

\[
\begin{array}{cccccc}
S & E & N & D & M & O \\
\hline
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
\end{array}
\]

---

**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, ...

---

**One-Time Pad Encryption**

**Encryption.**

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

---

**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>

---

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

One-Time Pad Encryption

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

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

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.

\[
\begin{array}{ccccccccccc}
\text{encrypted} & g & X & 7 & 6 & W & 3 & v & 7 & K \\
\text{base64} & 10000 & 01011 & 11101 & 11101 & 11011 & 11011 & 111011 & 001010 \\
\text{random bits} & 11010 & 01001 & 11101 & 11101 & 11101 & 11101 & 111011 & 001010 \\
\end{array}
\]

• Use same N random bits (one-time pad).

- Key point: Bob and Alice agreed on the one-time pad beforehand

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

<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>C</td>
<td>2</td>
<td>000010</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>000011</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>000100</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>000101</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>000110</td>
</tr>
<tr>
<td>H</td>
<td>7</td>
<td>000111</td>
</tr>
<tr>
<td>I</td>
<td>8</td>
<td>001000</td>
</tr>
<tr>
<td>J</td>
<td>9</td>
<td>001001</td>
</tr>
<tr>
<td>K</td>
<td>10</td>
<td>001010</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
<td>001011</td>
</tr>
<tr>
<td>M</td>
<td>12</td>
<td>001100</td>
</tr>
</tbody>
</table>

Base64 Encoding

SEND MONEY
100000 010111 111011 000100 011011 010001 101111 110011 011010
000100 000110 001110 001110 001110 001000 001000

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.

<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 010111 111011 000100 011011 010001 101111 110011 011010</td>
<td></td>
<td></td>
<td></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.

<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>
| 100000 010111 111011 110100 011001 110010 101111 110011 011010 | base64
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>111011</td>
<td>101010</td>
<td>101010</td>
<td>001010</td>
</tr>
</tbody>
</table>

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".
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 . . .) eavesdropper Eve sees only random bits

---

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.** [Ginsburg · Moretti · Browning · Dai · Davey · Dror · Drutskoy · Gabai · Ghosh · Ghosh · Kao · Miller · Nelson · Pereira · Tong · Yang]

- 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 tonight!

**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 exams (in class, 3/13-14 and 5/1-2). 50%.
- 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.

Course text.
Sedgewick and Wayne.
Intro to Programming in Java: An Interdisciplinary Approach.

Recommended reading (lectures 18-20).
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!

Due. Mondays 11pm 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.

Assignment 0. [www.princeton.edu/~cos126/assignments.php]
- Due Monday 11PM.
- 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

**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.

**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

**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.

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

– Jon von Neumann (left)
– ENIAC (right)
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 = \( b_0 \).

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>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>00100</td>
<td>01101</td>
<td>000011</td>
<td>00110</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>01100</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>11010</td>
<td>01101</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>01011</td>
<td>11011</td>
<td>111010</td>
<td>011101</td>
<td>110111</td>
<td>101111</td>
<td>01101</td>
<td></td>
</tr>
<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>
</tbody>
</table>
```

LFSR Decryption

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

```
<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>11101</td>
<td>111010</td>
<td>011010</td>
<td>110111</td>
<td>101111</td>
<td>010110</td>
<td></td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>11010</td>
<td>111001</td>
<td>01101</td>
<td>110101</td>
<td>100010</td>
<td>111111</td>
<td></td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>00101</td>
<td>000111</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td></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>
```

Base64 Encoding

```
A | d | 0 | 000000
B | 1 | 000001
... | ... | ...
w | 22 | 010110
... | ... | ...
```

Base64 Encoding

```
A | d | 0 | 000000
B | 1 | 000001
... | ... | ...
M | 12 | 001100
... | ... | ...
```

Random Numbers

- Are these 2000 numbers random?
- If not, what is the pattern?

```
01101000010...
```

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.
  • 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, \(N = 61\)).

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: \(2^{70}\)

Ex: (1, 2) LFSR

<table>
<thead>
<tr>
<th>001</th>
<th>010</th>
<th>011</th>
<th>111</th>
<th>110</th>
<th>100</th>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^{11}-1 = 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

Copyright (C) 2000 Charles M. Hannum <root@ihack.net>, DVD Jon (Norwegian hacker)

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: \(2^{70}\)

Ex: (1, 2) LFSR

<table>
<thead>
<tr>
<th>001</th>
<th>010</th>
<th>011</th>
<th>111</th>
<th>110</th>
<th>100</th>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^{11}-1 = 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

Copyright (C) 2000 Charles M. Hannum <root@ihack.net>, DVD Jon (Norwegian hacker)
**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>

**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) {
        this.seed = new int[seed.length];
        this.tap = tap;
        this.N = N;
    }
    public int step() {
        // implementation...
    }
    public static void main(String[] args) {
        LFSR lfsr = new LFSR("01101000010", 8);
        for (int i = 0; i < 2000; i++)
            StdOut.println(lfsr.step());
    }
}
```

```bash
% java LFSR
01101000010
011010010011111011
110011000101111110...
```

A Profound Idea