1.0. Overview

Secure Chat

Alice wants to send a secret message to Bob?
- Can you read the secret message $gX76W3v7K$?
- But Bob can. How?

A Cryptographic Example

How to make a simple machine that produces "random" bits.
- Linear feedback shift register.

What we can do with it.
- Encrypt and decrypt secret messages.
- Encrypt DVDs with CSS.
- Decrypt DVDs with DeCSS!
- Use as subroutine in military cryptosystems.

Science behind it.

Encryption Machine

Goal: design a machine to encrypt and decrypt data.

<table>
<thead>
<tr>
<th>SEND MONEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$gX76W3v7K$</td>
</tr>
</tbody>
</table>

Encrypt

<table>
<thead>
<tr>
<th>SEND MONEY</th>
</tr>
</thead>
</table>

Decrypt

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.
Digital Data

Computers store all data as a sequence of bits. Text, images (JPEG), audio (MP3), video (DivX).

Base64 encoding.

- Use 6 bits to represent each alphanumeric symbol.

<table>
<thead>
<tr>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 A</td>
<td>001011 L</td>
<td>010110 W</td>
<td>100001 h</td>
<td>101100 s</td>
</tr>
<tr>
<td>000001 B</td>
<td>001100 M</td>
<td>010111 X</td>
<td>100101 i</td>
<td>101101 t</td>
</tr>
<tr>
<td>000010 C</td>
<td>001101 N</td>
<td>010100 Y</td>
<td>100111 j</td>
<td>101110 u</td>
</tr>
<tr>
<td>000011 D</td>
<td>001110 O</td>
<td>010101 Z</td>
<td>101000 k</td>
<td>101111 v</td>
</tr>
<tr>
<td>000100 E</td>
<td>001111 P</td>
<td>010110 a</td>
<td>100101 l</td>
<td>110000 w</td>
</tr>
<tr>
<td>000101 F</td>
<td>010000 Q</td>
<td>010111 b</td>
<td>100110 m</td>
<td>110001 x</td>
</tr>
<tr>
<td>000110 G</td>
<td>010001 R</td>
<td>011000 c</td>
<td>100111 n</td>
<td>110010 y</td>
</tr>
<tr>
<td>000111 H</td>
<td>010010 S</td>
<td>011001 d</td>
<td>101000 o</td>
<td>110011 z</td>
</tr>
<tr>
<td>001000 I</td>
<td>010011 T</td>
<td>011010 e</td>
<td>101001 p</td>
<td>111000 0</td>
</tr>
<tr>
<td>001001 J</td>
<td>010100 U</td>
<td>011101 f</td>
<td>101100 q</td>
<td>111010 1</td>
</tr>
<tr>
<td>001010 K</td>
<td>010101 Y</td>
<td>100000 g</td>
<td>101101 r</td>
<td>110110 2</td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

1. Convert text message to N bits.

<table>
<thead>
<tr>
<th>Base64 Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>Y</td>
</tr>
</tbody>
</table>

2. Generate N random bits (one-time pad).
A Russian One-Time Pad

Random Numbers

Are these 2000 numbers random?

If not, what is the pattern?

One-Time Pad Encryption

1. Convert text message to N bits.
2. Generate N random bits (one-time pad).
3. Take bitwise XOR of two strings.
   - Sum pair of bits (1 if sum is odd, 0 if even).

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>

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>000000</td>
</tr>
<tr>
<td>X</td>
<td>23</td>
<td>010111</td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

1. Convert text message to N bits.
2. Generate N random bits (one-time pad).
3. Take bitwise XOR of two strings.
4. Convert binary back into text.
One-Time Pad Decryption

1. Convert encrypted message to binary.
2. Use same N random bits (one-time pad).
3. Take bitwise XOR of two strings.
One-Time Pad Decryption

1. Convert encrypted message to binary.
2. Use same N random bits (one-time pad).
3. Take bitwise XOR of two strings.
4. 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>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Y</td>
<td>24</td>
<td>011000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Base64 Encoding**

<table>
<thead>
<tr>
<th>Base64 Encoding</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>original message</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>one-time pad</td>
</tr>
<tr>
<td>^</td>
<td>XOR operator</td>
<td></td>
</tr>
<tr>
<td>a ^ b</td>
<td>encrypted message</td>
<td></td>
</tr>
<tr>
<td>(a ^ b) ^ b</td>
<td>decrypted message</td>
<td></td>
</tr>
</tbody>
</table>

**Why Does It Work?**

**Crucial property:** \((a ^ b) ^ b = a\).
- Decrypted message = original message.

**Why is crucial property true?**
- Use properties of XOR.
- \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)
  - always 0

---

**An Cryptographic Example**

How to make a simple machine that produces “random” bits.
- Linear feedback shift register.

- What we can do with it.
  - Encrypt and decrypt secret messages.
  - Encrypt DVDs with CSS.
  - Decrypt DVDs with DeCSS!
  - Use as subroutine in military cryptosystems.

Science behind it.

---

**Random Numbers**

"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

How might the "random number machine" be built?
- Linear feedback shift register.
- Linear congruential generator.

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

Linear feedback shift register.
- Machine consists of 11 bits.
- Bit values change at discrete time points.
- Bit values at time $T+1$ determined by bit values at time $T$.
  - new bits 1 - 10 are old bits 0 - 9
  - new bit 0 is XOR of previous bits 8 and 10
  - output bit 0

Random Numbers

Are these 2000 numbers random?

No! Real machines are deterministic.

How did the computer scientist die in the shower?
The instructions on the shampoo read "lather, rinse, repeat."

Will bit pattern repeat itself?
Yes, after $2^{11} - 1 = 2047$ steps.

What if I need more bits?
Scalable: 20 cells for 1 million bits, 30 for 1 billion.

Will the machine work equally well if we XOR bits 4 and 10?
No, need to understand theory of abstract rings.

How many cells do I need to guarantee a certain level of security?
Subject of active research.

The Science Behind It

The Science Behind It

Random Numbers

Are these 2000 numbers random?

No! Real machines are deterministic.

How did the computer scientist die in the shower?
The instructions on the shampoo read "lather, rinse, repeat."

Will bit pattern repeat itself?
Yes, after $2^{11} - 1 = 2047$ steps.

What if I need more bits?
Scalable: 20 cells for 1 million bits, 30 for 1 billion.

Will the machine work equally well if we XOR bits 4 and 10?
No, need to understand theory of abstract rings.

How many cells do I need to guarantee a certain level of security?
Subject of active research.
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.

Simulating The Abstract Machine in Java

Java program prints exactly same bits as LFBSR.
- You'll understand this program by next week.

```java
public class LFBSR {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        boolean b10 = false, b9 = true, b8 = true, b7 = false;
        boolean b6 = true, b5 = false, b4 = false, b3 = false;
        boolean b2 = false, b1 = true, b0 = false;

        for (int i = 0; i < N; i++) {
            boolean bit = b8 ^ b10;
            b10 = b9; b9 = b8; b8 = b7; b7 = b6; b6 = b5;
            b5 = b4; b4 = b3; b3 = b2; b2 = b1; b1 = b0;
            b0 = bit;
            if (bit) System.out.print(1);
            else System.out.print(0);
        }
        System.out.println();
    }
}
```

% java LFBSR 2000
010011001000000110001000101110101011110010011110011101001000110111111100101 ...