Secure Chat

Alice wants to send a secret message to Bob?
- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.
- Bob uses the same pad to decrypt the message.

Key point. Without the pad, Eve cannot understand the message.

Encryption Machine

Goal. Design a machine to encrypt and decrypt data.

 SENDMONEY

<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>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>

encrypt

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.
A Digital World

Data is a sequence of bits. \([\text{bit} = 0 \text{ or } 1]\)

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

File formats: txt, pdf, java, exe, docx, pptx, jpeg, mp3, divx, ...

One-Time Pad Encryption

Encryption.
- Convert text message to \(N\) bits.

Base64 Encoding

<table>
<thead>
<tr>
<th>Binary Char</th>
<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>000001 B</td>
<td>000010 C</td>
<td>000011 D</td>
<td>000100 E</td>
<td>000101 F</td>
</tr>
<tr>
<td>000110 G</td>
<td>000111 H</td>
<td>001000 I</td>
<td>001001 J</td>
<td>001010 K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

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

message

Base64

random bits
One-Time Pad Encryption

Encryption.
- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

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>

Secure Chat (review)

Alice wants to send a secret message to Bob?
- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.
- Bob uses the same pad to decrypt the message.

Key point. Without the pad, Eve cannot understand the message.

One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.

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>w</td>
<td>22</td>
<td>010110</td>
</tr>
</tbody>
</table>

Encrypted message:
- base64: SENDMONEY
- random bits: 010010 000100 001101 000011 001100 001110 001101 000100 011000
- XOR: 100000 010111 111011 111010 010110 110111 101111 111011 001010

Base64 decoding:
- message: alice and bob
- random bits: 110010 010011 110110 111001 011010 111001 100010 111111 010010
- XOR: g x 7 6 m 3 v 7 k

Encrypted:
- g x 7 6 m 3 v 7 k

Alice uses the pad to encrypt the message, and Bob uses the same pad to decrypt the message.
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>X</td>
<td>22</td>
<td>110110</td>
</tr>
</tbody>
</table>
- -   | -   | -      |

**XOR Truth Table**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \oplus y$</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**

- Original message: SEND MONEY
- Encrypted: 100000 010111 111011 111010 010110 110111 101111 111011 001010
- Decrypted: 010010 000100 001101 000011 001100 001110 001101 000100 011000
- Original message: SEND MONEY

**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>
</tbody>
</table>
- -   | -   | -      |

![Binary Conversion Diagram](https://via.placeholder.com/150)

![XOR Truth Table](https://via.placeholder.com/150)

![Base64 Encoding Table](https://via.placeholder.com/150)
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.
- \((a \land b) \lor (b \land \overline{b}) = a \land 0 = a\)

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>M</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64</td>
<td>22</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>20</td>
<td>22</td>
<td>20</td>
<td>22</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).
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>M</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>01011</td>
<td>11011</td>
<td>01000</td>
<td>01011</td>
<td>01011</td>
<td>11011</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>01110</td>
<td>11011</td>
<td>01111</td>
<td>01011</td>
<td>01011</td>
<td>11011</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>01111</td>
<td>11011</td>
<td>01111</td>
<td>01011</td>
<td>01011</td>
<td>11011</td>
<td>01011</td>
<td></td>
</tr>
</tbody>
</table>

**encrypted base64**
```
100000 010111 110111 010000 010111 010111 110111 010111
```

**wrong bits**
```
101000 011100 110110 010010 011110 011001 111001 100101
```

**XOR**
```
010000 010111 111011 111010 010110 110111 101111 111011
```

**encrypted base64**
```
100000 010111 110111 010000 010111 010111 110111 010111
```

**wrong bits**
```
101000 011100 110110 010010 011110 011001 111001 100101
```

**XOR**
```
010000 010111 111011 111010 010110 110111 101111 111011
```

**wrong message**
```
I L O V E O R R A
```

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.

**Practical for Web commerce**

A Russian one-time pad
Pseudo-Random Bit Generator

Practical middle-ground.
- Let’s make a “random”-bit generator gadget.
- Alice and Bob each get identical small gadgets.

How to make small gadget that produces “random” bits.
- Enigma machine.
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.

“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”
− Jon 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 $b_8$ is is XOR of previous bits $b_8$ and $b_{10}$.
- Pseudo-random bit = $b_0$.

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 0110100001!
LFSR Encryption

- Convert text message to N bits.
- Initialize LFSR with small 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>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>S</td>
<td>22</td>
<td>10110</td>
</tr>
</tbody>
</table>

message

base64

SENDMONEY

LFSR bits

000011011111011011110111100110111011101101110101101110011101110011

XOR

0100100001001100001101100000110011001110100001100111011101100011

encrypted

g X 7 6 w 3 v 7 K

base64

100001010101110111101110111101011110110011101110011

LFSR bits

01011000000111001110011010111110101101011001110011

XOR

0100100001001100001101100000110011001110100001100111011101100011

message

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 LFSR Encryption

Goods.
- Easily computed with simple machine.
- Very simple encryption/decryption process.
- 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 ]

Bads.
- 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.
  [ more complicated machines needed ]

Goods and Bads of LFSR Encryption

- Easily computed with simple machine.
- Very simple encryption/decryption process.
- 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 ]

LFSR Decryption

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

LFSR Decryption

- Convert encrypted message to N bits.
- Initialize identical LFSR with same seed.
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.
LFSR vs. 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, step, load</td>
<td>some</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>4 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 computer can be programmed to simulate any abstract machine.

```java
public class LFSR {
    // declare instance variables
    private int N;       // number of bits in the LFSR
    private int[] reg;   // reg[i] = ith bit of LFSR, reg[0] is rightmost bit
    private int tap;     // index of the tap bit

    // constructor to create LFSR with the given initial seed and tap
    public LFSR(String seed, int t) {
        // PUT YOUR CODE HERE
    }

    // simulate one step and return the new bit as 0 or 1
    public int step() {
        // PUT YOUR CODE HERE
    }

    // simulate k steps and return k-bit integer
    public int generate(int k) {
        // PUT YOUR CODE HERE
    }

    // return a string representation of the LFSR
    public String toString()  {
        // PUT YOUR CODE HERE
    }

    // test client
    public static void main(String[] args)  {
        // PUT YOUR TEST CODE HERE
    }
}
```