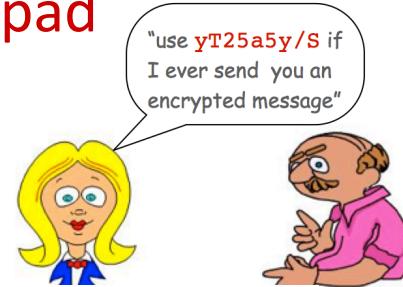


# Secure Chat

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



The image shows two side-by-side windows of a "Chat Client 1.0" application. Both windows have a title bar with the logo and the text "Chat Client 1.0: [alice]" or "[bob]". The left window is for Alice, and the right window is for Bob. Both windows show the following messages:

[alice]: Hey Bob.  
[bob]: Hi Alice!  
[alice]: gX76W3v7K

At the bottom of each window, there is a red text instruction:

Encrypt SENDMONEY with yT25a5y/S      Decrypt gX76W3v7K with yT25a5y/S

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

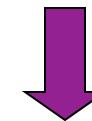


# Encryption Machine

Goal: Design a machine to encrypt and decrypt data

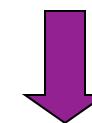
S	E	N	D	M	O	N	E	Y
---	---	---	---	---	---	---	---	---

encrypt



g	x	7	6	w	3	v	7	k
---	---	---	---	---	---	---	---	---

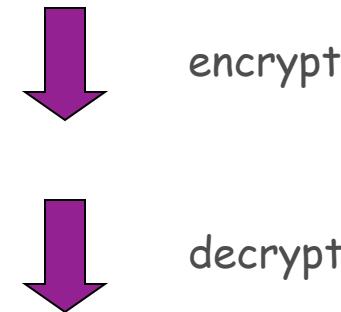
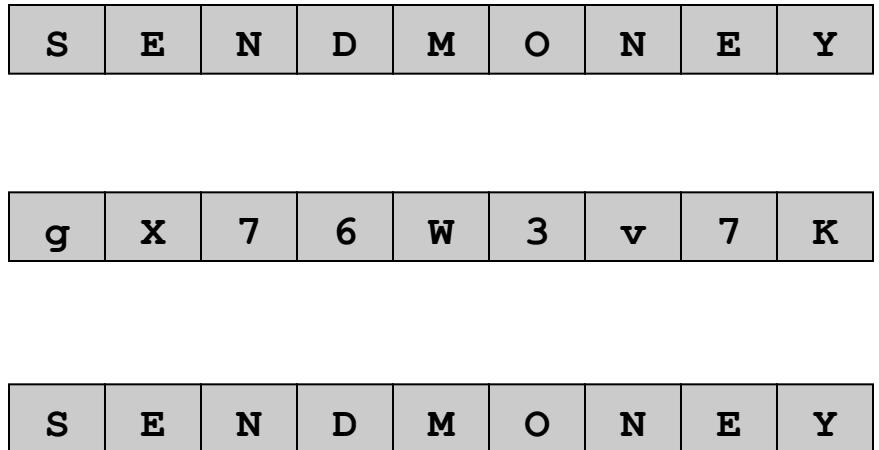
decrypt



S	E	N	D	M	O	N	E	Y
---	---	---	---	---	---	---	---	---

# 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, ...

Base64 encoding Use 6 bits to represent each alphanumeric symbol.

| Binary Char |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 000000      | A           | 001011      | L           | 010110      | W           | 100001      | h           |
| 000001      | B           | 001100      | M           | 010111      | X           | 100010      | i           |
| 000010      | C           | 001101      | N           | 011000      | Y           | 100011      | j           |
| 000011      | D           | 001110      | O           | 011001      | Z           | 100100      | k           |
| 000100      | E           | 001111      | P           | 011010      | a           | 100101      | l           |
| 000101      | F           | 010000      | Q           | 011011      | b           | 100110      | m           |
| 000110      | G           | 010001      | R           | 011100      | c           | 100111      | n           |
| 000111      | H           | 010010      | S           | 011101      | d           | 101000      | o           |
| 001000      | I           | 010011      | T           | 011110      | e           | 101001      | p           |
| 001001      | J           | 010100      | U           | 011111      | f           | 101010      | q           |
| 001010      | K           | 010101      | V           | 100000      | g           | 101011      | r           |
|             |             |             |             |             |             | 110110      | 2           |

# One-Time Pad Encryption

*Base64 Encoding*

## Encryption

- Convert text message to N **bits**

char	dec	binary
A	0	000000
B	1	000001
...	...	...
M	12	001100
...	...	...

S	E	N	D	M	O	N	E	Y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64

# One-Time Pad Encryption

## Encryption

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

S	E	N	D	M	O	N	E	Y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits

# One-Time Pad Encryption

XOR Truth Table

x	y	$x \wedge y$
0	0	0
0	1	1
1	0	1
1	1	0

## Encryption

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

sum corresponding pair of bits: 1 if sum is odd, 0 if even

s	e	n	d	m	o	n	e	y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR

0 ^ 1 = 1

# One-Time Pad Encryption

*Base64 Encoding*

## Encryption

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

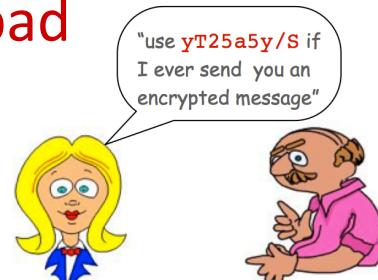
char	dec	binary
A	0	000000
B	1	000001
...	...	...
w	22	010110
...	...	...

s	e	n	d	m	o	n	e	y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR
g	x	7	6	w	3	v	7	k	encrypted

# 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



The image shows two side-by-side windows of a "Chat Client 1.0" application. Both windows have a title bar with the logo and the text "Chat Client 1.0: [alice]" or "[bob]". The left window is for Alice, and the right window is for Bob. Both windows show the following messages:

[alice]: Hey Bob.  
[bob]: Hi Alice!  
[alice]: gX76W3v7K

At the bottom of each window, there is a red text overlay:

Encrypt SENDMONEY with yT25a5y/S (in the Alice window)  
Decrypt gX76W3v7K with yT25a5y/S (in the Bob window)



# One-Time Pad Decryption

## Decryption

- Convert encrypted message to binary

g	x	7	6	w	3	v	7	k	encrypted
---	---	---	---	---	---	---	---	---	-----------

# One-Time Pad Decryption

*Base64 Encoding*

## Decryption

- Convert encrypted message to binary

char	dec	binary
A	0	000000
B	1	000001
...	...	...
W	22	010110
...	...	...

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64

# One-Time Pad Decryption

## Decryption

- Convert encrypted message to binary
- Use **same** N random bits (one-time pad)

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits

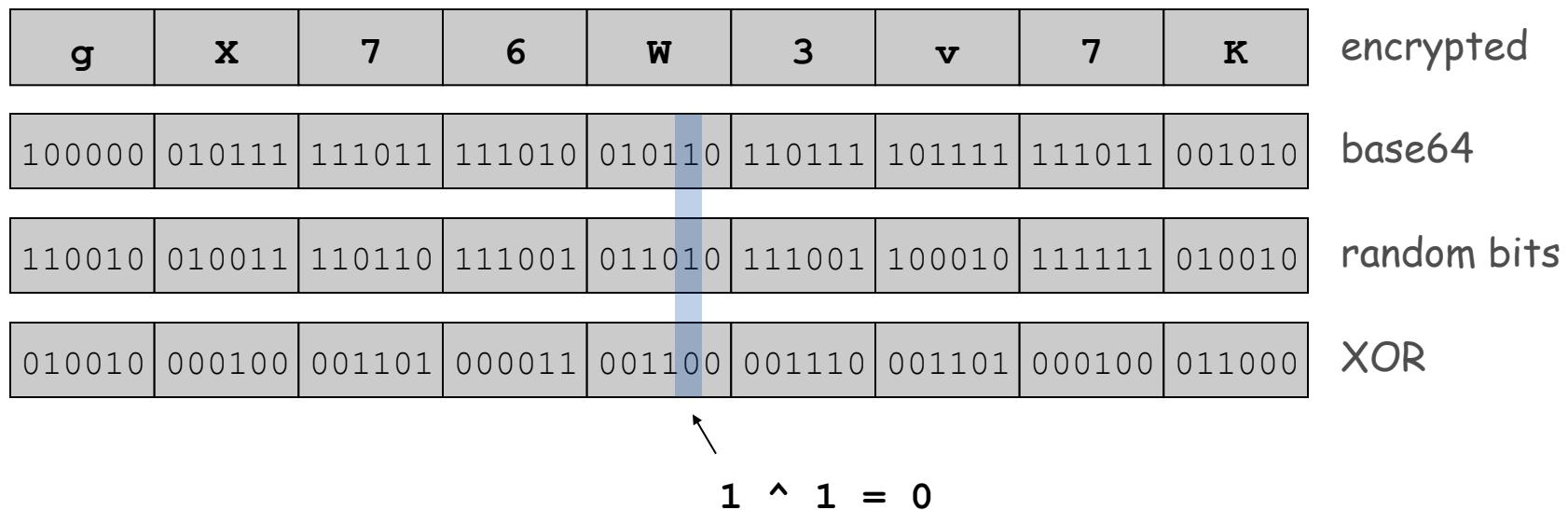
# One-Time Pad Decryption

XOR Truth Table

x	y	$x \wedge y$
0	0	0
0	1	1
1	0	1
1	1	0

## Decryption

- Convert encrypted message to binary
- Use same N random bits (one-time pad)
- Take bitwise XOR of two bitstrings



$$1 \wedge 1 = 0$$

# One-Time Pad Decryption

*Base64 Encoding*

char	dec	binary
A	0	000000
B	1	000001
...	...	...
M	12	001100
...	...	...

## Decryption

- Convert encrypted message to binary
- Use same N random bits (one-time pad)
- Take bitwise XOR of two bitstrings
- Convert back into text

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
010010	000100	001101	000011	001100	001110	001101	000100	011000	XOR
s	e	n	d	m	o	n	e	y	message

# Why Does It Work?

Crucial property Decrypted message = original message

Notation	Meaning
a	original message bit
b	one-time pad bit
$\wedge$	XOR operator
$a \wedge b$	encrypted message bit
$(a \wedge b) \wedge b$	decrypted message bit

Why is crucial property true?

- Use properties of XOR.
- $(a \wedge b) \wedge b = a \wedge (b \wedge b) = a \wedge 0 = a$   
    ↑                      ↑                      ↑  
    associativity of  $\wedge$     always 0        identity

*XOR Truth Table*

x	y	$x \wedge y$
0	0	0
0	1	1
1	0	1
1	1	0

# One-Time Pad Decryption (with the wrong pad)

## Decryption

- Convert encrypted message to binary

g	x	7	6	w	3	v	7	k	encrypted
---	---	---	---	---	---	---	---	---	-----------

# One-Time Pad Decryption (with the wrong pad)

## Decryption

- Convert encrypted message to binary

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64

# One-Time Pad Decryption (with the wrong pad)

## Decryption

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

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	<b>wrong bits</b>

# 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

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	<b>wrong bits</b>
001000	001011	001110	010101	000100	001110	001010	010001	000000	XOR

# 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**

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	wrong bits
001000	001011	001110	010101	000100	001110	001010	010001	000000	XOR
I	L	o	v	E	o	k	R	A	wrong message

# Goods and Bads of One-Time Pads

## Good

- Easily computed by hand
- Very simple encryption/decryption processes
- Provably unbreakable if bits are truly random

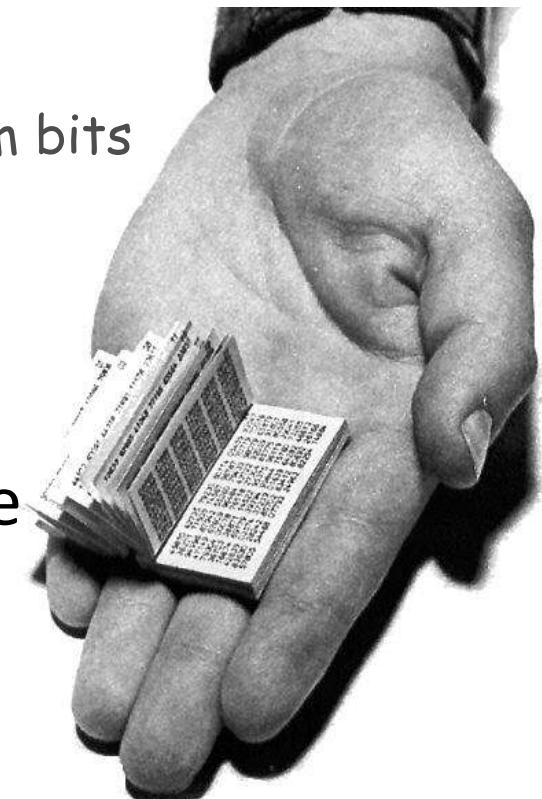
[Shannon, 1940s]



eavesdropper Eve sees only random bits

## Bad

- Easily breakable if pad is re-used
- Pad must be as long as the message
- Truly random bits are hard to generate
- **Pad must be distributed securely**
  - impractical for Web commerce



a Russian one-time pad

# Pseudo-Random Bit Generator

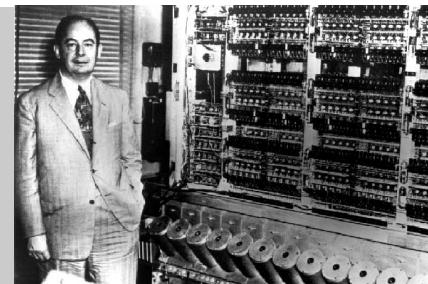
## Practical middle-ground

- Let's make a "random"-bit generator gadget
- Alice and Bob each get identical small gadgets

## How to make a gadget that produces "random" bits

- Enigma machine
- Linear feedback shift register
- Linear congruential generator
- Blum-Blum-Shub generator

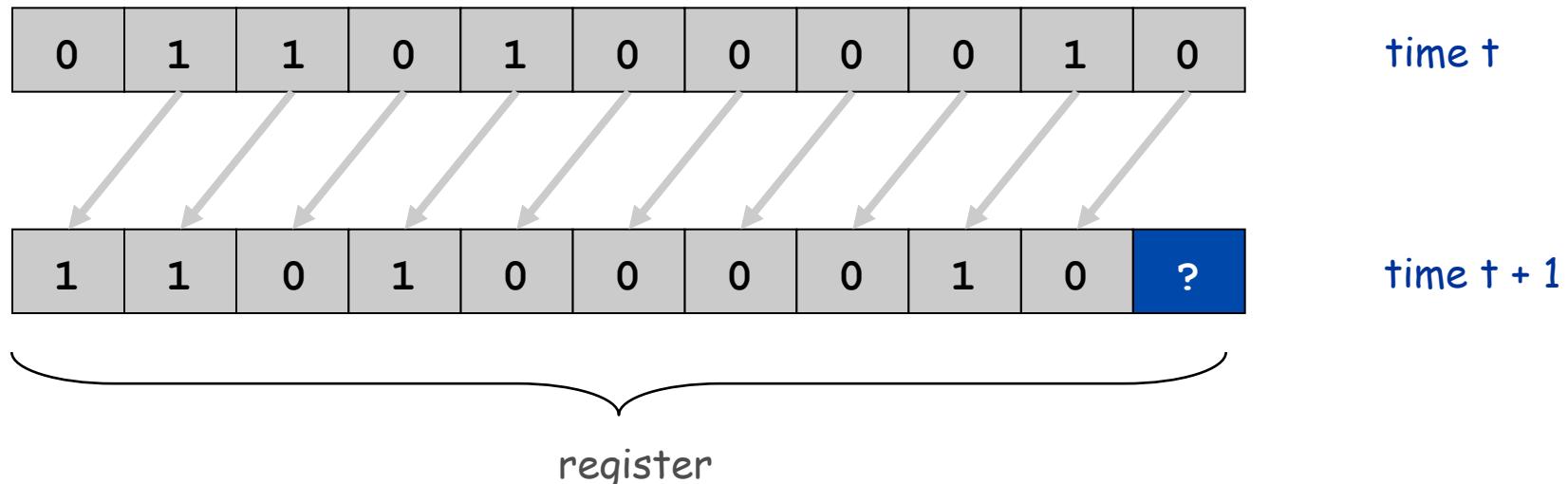
*“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”*  
– Jon von Neumann (left)  
– ENIAC (right)



# Shift Register

## Shift register terminology.

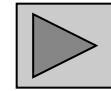
- Bit: 0 or 1
- Cell: storage element that holds one bit
- Register: sequence of cells
- Seed: initial sequence of bits
- Shift register: when clock ticks, bits propagate one position to left



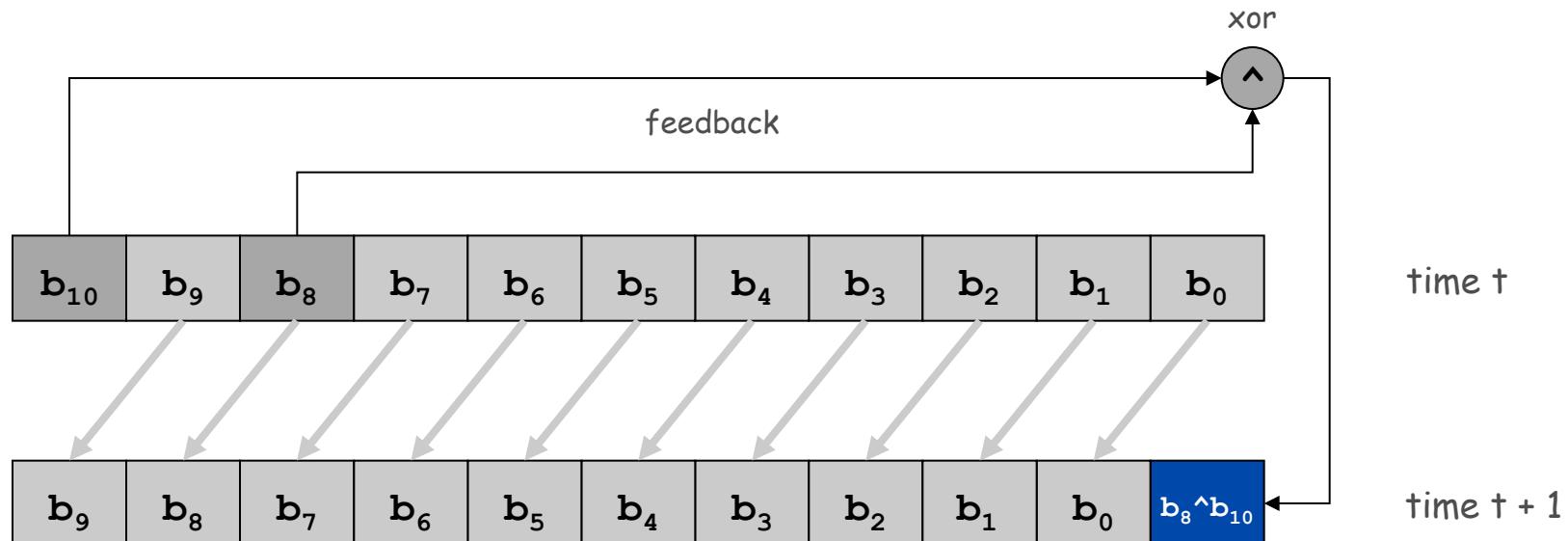
# Linear Feedback Shift Register (LFSR)

## {8, 10} linear feedback shift register

- Shift register with 11 cells
- Bit  $b_0$  is XOR of previous bits  $b_8$  and  $b_{10}$
- Pseudo-random bit =  $b_0$



LFSR demo



# Random Numbers

Q. Are these 2000 numbers random?

```
11001001001110110111001011010111001000101111101001000010011010010111001100100111111  
10111000001010110001000011101010011010000111100100110011101111110101000001000010001010  
010101000110000010111100010010011010111000110100110111001011110010001001110101  
0111010000010100100011010101110000001011000001001110001011101101001011001100010001001110101  
1111110011000001111100011000011011110011001110100111101001110100111011101010101000  
000000010000000001010000001000100001010100100000001101000001110010001101110101110100  
0101000010100010010001010110101000011000010011110010111001011110111001001010111011  
00001010111001000010111010010010101100011110111011001010101111000000100110000101111  
100100100011101101011000110001110111101101010010110000110011100111110111100001010  
0110010001111101011000010001110010101110000110101100111110110110001011011101101100010110  
01101010011110000111001100110111111010000000100100000101101000100110010101111100001  
000011001010011110001110001101101110110101101100001101110001110101101100011  
01100101110111100101010110000011101100011010111011100010101011010000011001000011110  
100110001001111010111000100010110101001100000011110000110001100111101111100101000  
11100010011011010111101100010010111010110010100011110001011001101001111100111000011110  
1100110010111111001000000111010000110100100111001101111010100100000011001000011110  
1000001001010001011000101001110100011101001011010011001111111100000000011000000  
1111000001100110001111111011000000101110000100101100101100111100111100111100011110011  
011001111101111100010100011010010110010111000110010110111100110100111100101110010110  
0011100111010111101011010010001100110101111110001000001101010001110000101101100100110  
111101111010010010001101111101110100010101001010000011000100011110101011001000001  
111010001100100101111011001000101111010100100000110110100111011001110101111101000100  
0100101010110000000111000000110110000111011001010111100001000110001010111101010
```

A. No. This is output of {8, 10} LFSR with seed 01101000010!

# LFSR Encryption

*Base64 Encoding*

char	dec	binary
A	0	000000
B	1	000001
...	...	...
w	22	010110
...	...	...

## 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

S	E	N	D	M	O	N	E	Y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	LFSR bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR
g	x	7	6	w	3	v	7	k	encrypted

# LFSR Decryption

*Base64 Encoding*

char	dec	binary
A	0	000000
B	1	000001
...	...	...
M	12	001100
...	...	...

## 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

g	x	7	6	w	3	v	7	k	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	LFSR bits
010010	000100	001101	000011	001100	001110	001101	000100	011000	XOR
s	e	n	d	m	o	n	e	y	message

# Goods and Bads of LFSR Encryption

a commercially available LFSR

## Goods

- Easily computed with simple machine
- Very simple encryption/decryption process
- Scalable: 20 cells = 1 million bits; 30 cells = 1 billion bits  
[ need theory of finite groups to know where to put taps ]



## Bads

- 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 ]

# Other LFSR Applications

## What else can we do with a LFSR?

- DVD encryption with CSS
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems



DVD Jon  
(Norwegian hacker)

```
/*      efddt.c      Author: Charles M. Hannum <root@ihack.net>      */
/*      Usage is: cat title-key scrambled.vob | efddt >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,j=      m(4)     17^ m(3)     9^k*     2-k%8
                        ^8,a      =0,c      =26;for     (s[y]      -=16;
                        --c;j *=2)a=      a*2^i&     1,i=i /2^j&1
                        <<24;for(j=      127;       ++j<n;c=c>
                                y)
                                c

                        +=y=i^i/8^i>>4^i>>12,
                        i=i>>8^y<<17,a^=a>>14,y=a^a*8^a<<6,a=a
                        >>8^y<<9,k=s[j],k      ="7Wo~'G_\216"[k
                        &7]+2^"cr3sfw6v;*k+>/n."[k>>4]*2^k*257/
                        8,s[j]=k^(k&k*2&34)*6^c+~y
                        ;}}
```