

Hidden flaw jeopardizes millions of online transactions

Mathematicians discover weakness in commonly used encryption schemes

Jump to discuss

comments below

X

Below:  Discuss  Related



By Paul Wagenseil



updated 2/15/2012 6:01:37 PM ET

Mathematicians based in Switzerland and the United States have discovered a small but the most commonly used digital encryption schemes, a flaw that could undermine the security of communication.

Blogs



IT Blogwatch

A Daily Digest of IT Blogs from Richi Jennings



[More posts](#) | [Read bio](#)

February 15, 2012 - 6:00 A.M.

RSA crypto: 'flawed', 'risky', 'quagmire of vulnerabilities'

2 Comments

Like < 5

+1 < 1

TAGS: certificate, crypto, cryptography, Diffie-Hellman, encryption, key management, RSA, SSL

IT TOPICS: Applications, Cybercrime & Hacking, Financial IT, Government & Regulation, Internet, Privacy, Security, Security Hardware & Software

A group of six academic researchers have concluded that real-world RSA encryption keys are riskier than Diffie-Hellman-based ones. It seems that some of the random numbers used to generate them weren't, erm, random. In [IT Blogwatch](#), bloggers wonder if the sky is falling.

The Register®

Hardware Software Music & Media Networks **Security** Cloud Public Sector Business Science
Crime Malware **Enterprise Security** Spam ID

[Print](#)[Comment](#)[Tweet](#)[Like](#)

< 1

[Alert](#)

'Predictably random' public keys can be cracked - crypto boffins Battling researchers argue over whether you should panic

By [John Leyden](#) • Get more from this author

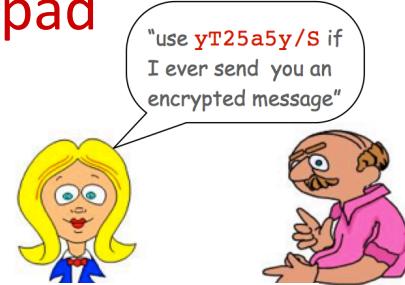
Posted in Enterprise Security, 16th February 2012 13:38 GMT

Analysis Cryptography researchers have discovered flaws in the key generation that underpins the security of important cryptography protocols, including SSL.

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]" and "Chat Client 1.0: [bob]" respectively. The left window contains the following text:
[alice]: Hey Bob.
[bob]: Hi Alice!
[alice]: gX76W3v7K
At the bottom of this window, there is a red text overlay: "Encrypt SENDMONEY with yT25a5y/S". The right window contains the same text:
[alice]: Hey Bob.
[bob]: Hi Alice!
[alice]: gX76W3v7K
At the bottom of this window, there is a red text overlay: "Decrypt gX76W3v7K with yT25a5y/S".

Key point Without the pad, Eve cannot understand the message

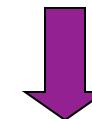


Encryption Machine

Goal: Design a machine to encrypt and decrypt data

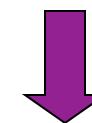
S	E	N	D	M	O	N	E	Y
---	---	---	---	---	---	---	---	---

encrypt



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

decrypt

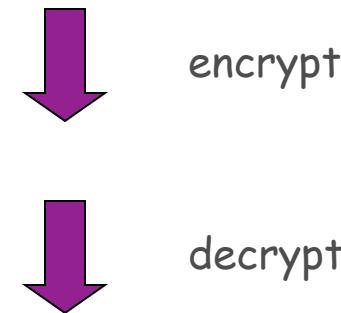


S	E	N	D	M	O	N	E	Y
---	---	---	---	---	---	---	---	---

Encryption Machine

Goal: Design a machine to encrypt and decrypt data

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



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

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

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, ...

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



computer with
a lens



computer with
earbuds



computer with
a radio

A Digital World

Data is a sequence of bits [bit = 0 or 1]

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

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



computer with
a cash dispenser



computer with
a ballot box



computer with
a heating element

001010 0010, 0010
00101010 0010001110:
00100100100001 00100100001110101 0001001 00100101
010100010010 00 00100 00100010010 0010100010 (100100
0001010 0001010010010 001000100100 0010010) 0010010
0010001000100001 0010 00100100 . 001000100 00100100 010
0001000100 00001100100 0010001001000010 0010001 0001.
001001001 01001001 0100010 ? 0010 1001001 0000100 0100
0010001, "1001000 0010 10001 00101010 001 00100100 0010
001001 - 100100 100 001000 001 001000 101001." 010010
001001 000100 01000 001000 00100 001000 100011111 10001
10001001 01000100 00101100010 00001000 01101010 00
01001.

1001000 1000 01000 001001 1010001 000 1000 00
0100100 0100100 01000 1010100110 010001110001
0010011110001 0100101 10100010010010.

10011000101000100100,

100110101

BINARY LETTER FROM GRANDMA

NUMBER SYSTEMS AND DATA REPRESENTATION

There are 10 kinds of people in the world, those who use binary, and those who don't.

Base 10 (Decimal numbers)

What does 157 mean?

$$\begin{aligned}157 &= 1 \times 100 + 5 \times 10 + 7 \times 1 \\&= 1 \times 10^2 + 5 \times 10^1 + 7 \times 10^0\end{aligned}$$

Decimal Numbering System

The Arabic numbering system uses ten symbols and is a base ten numbering system. In any numbering system the base is raised to consecutive powers to determine positional (place) values.

10^3	10^2	10^1	10^0	base ^{powers}
1000	100	10	1	place values
	2	3	1	
	4	1	3	
7	8	6	0	

$231 = 2 * 10^2 + 3 * 10^1 + 1 * 10^0$

Computer Representation

Binary System

At the hardware level computers must store numbers using either the presence or absence of a voltage. Then there are only two states (symbols) available for number representation. So base 2 (binary) is a suitable system for computers.

2^6	2^5	2^4	2^3	2^2	2^1	2^0	base ^{powers}
64	32	16	8	4	2	1	place values

Positional Systems

What value does 1110 represent?

Undetermined until the numbering system base is stated.

Base 10 place values: 10,000 1,000 100 10 1

Base 2 place values: 16 8 4 2 1

 1 1 1 0

base

base 10

10? $1*1000 + 1*100 + 1*10 + 0*1$ 1,110

2? $1*8 + 1*4 + 1*2 + 0*1$ 14

Base 10 vs Base 2

Base 10

157

$$\begin{aligned}157 &= 1 \times 100 + 5 \times 10 + 7 \times 1 \\&= 1 \times 10^2 + 5 \times 10^1 + 7 \times 10^0\end{aligned}$$

Base 2

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$1011 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$

Binary to Base Ten

01010010

0	1	0	1	0	0	1	0
↓	↓	↓	↓	↓	↓	↓	↓
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Binary to Base Ten

01010010

0	1	0	1	0	0	1	0
↓	↓	↓	↓	↓	↓	↓	↓
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

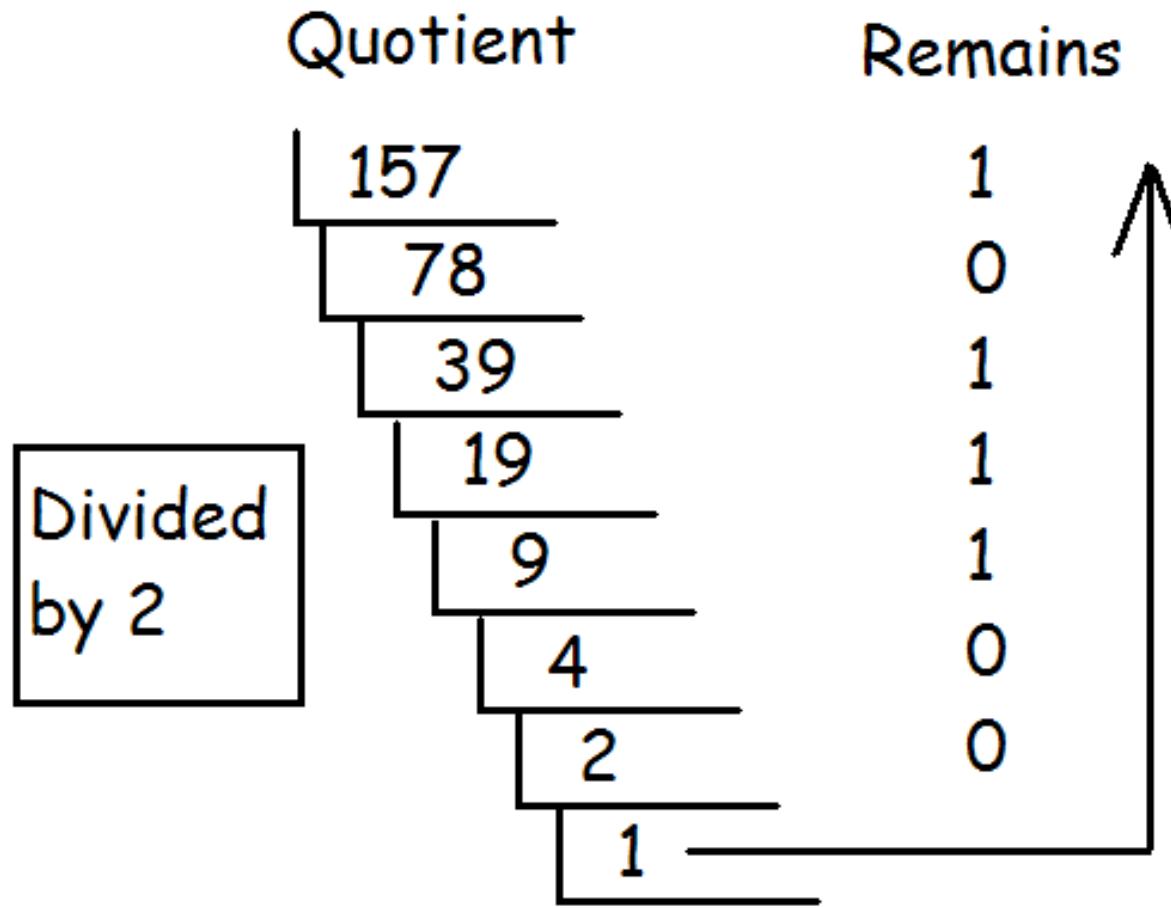
$$0 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

$$0 + 64 + 0 + 16 + 0 + 0 + 2 + 0$$

82

Base 10 to Binary

What is 157_{10} in binary?



Answer: 10011101

Binary mathematics

$$0+0=0$$

$$1+0=1$$

$$1+1=10$$

$$\begin{array}{r} 11001011 \\ + 11100110 \\ \hline 110110001 \end{array}$$

Representing Negative Integers

TOY words are 16 bits each.

- We could use 16 bits to represent 0 to $2^{16} - 1$.
- We want negative integers too.
- Reserving half the possible bit-patterns for negative seems fair.

Highly desirable property. If x is an integer, then the representation of $-x$, when added to x , is zero.

$$\begin{array}{r} x \quad 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \\ + (-x) \quad + \ ?\ ?\ ?\ ?\ ?\ ?\ ?\ ? \\ \hline 0 \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{array}$$

flip bits and add 1

$$\begin{array}{r} x \quad 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \\ + (-x) \quad + \ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ \hline 0 \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{array}$$

~~1 1 1 1 1 1 1 1~~
+ 1

Two's Complement Integers

To compute $-x$ from x :

- Start with x .

+4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

The diagram shows a 17-bit binary representation of the integer +4. The bits are arranged in a sequence of 0s followed by a 1. A red arrow points to the first bit (the most significant bit), which is labeled "leading bit".

- Flip bits.

-5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1

The diagram shows a 17-bit binary representation of the integer -5. All bits are set to 1, except for the least significant bit which is 0.

- Add one.

-4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

The diagram shows a 17-bit binary representation of the integer -4. All bits are set to 1, except for the least significant bit which is 0. This is the result of adding 1 to the binary representation of -5.

Two's Complement Integers

dec	hex	binary															
+32767	7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

.....

+4	0004	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
+3	0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
+2	0002	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
+1	0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
+0	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-2	FFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
-3	FFFD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
-4	FFFC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

.....

-32768	8000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
--------	------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Hexadecimal (base 16)

Binary code is too long in representation.
Hex is much shorter.

Converting a binary number to a Hex number is
relatively easy

- Every 4 bits can convert to a single Hex value

Problem: we are short of numbers

- A=10 B=11 C=12 D=13 E=14 F=15

Hexadecimal (Base 16) Representation

Binary	HEX
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binary	HEX
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Example

	1 0 1 1	1 1 1 1	0 0 1 0	1 1 0 0
Dec	11	15	2	12
Hex	B	F	2	C
Result	BF2C			

Bits, Bytes, and Words

Information in the computer are stored as a groups of binary digits. An individual digit is called a **bit**.

Bits are grouped into 8-bit collections called **bytes**.

Memory is normally measured in terms of bytes.

- 256 possible values from 0-255 base 10 or
- 00000000 to 11111111 base 2

Bytes are further grouped into 4 or more byte groupings to make up a computer **word**. The most common word sizes in modern computers is 32 bits (4 8-bit bytes) or 64-bits

BACK TO ENCRYPTION

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

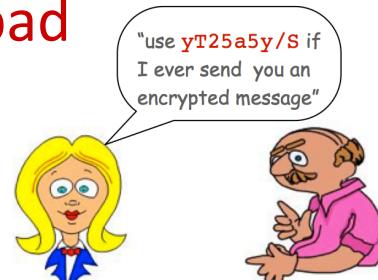
char	dec	binary
A	0	000000
B	1	000001
...
w	22	010110
...

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". The window on the left is titled "Chat Client 1.0: [alice]" and the window on the right is titled "Chat Client 1.0: [bob]". Both windows show a conversation between Alice and Bob:

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

At the bottom of each window, there is red text indicating the action:

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



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

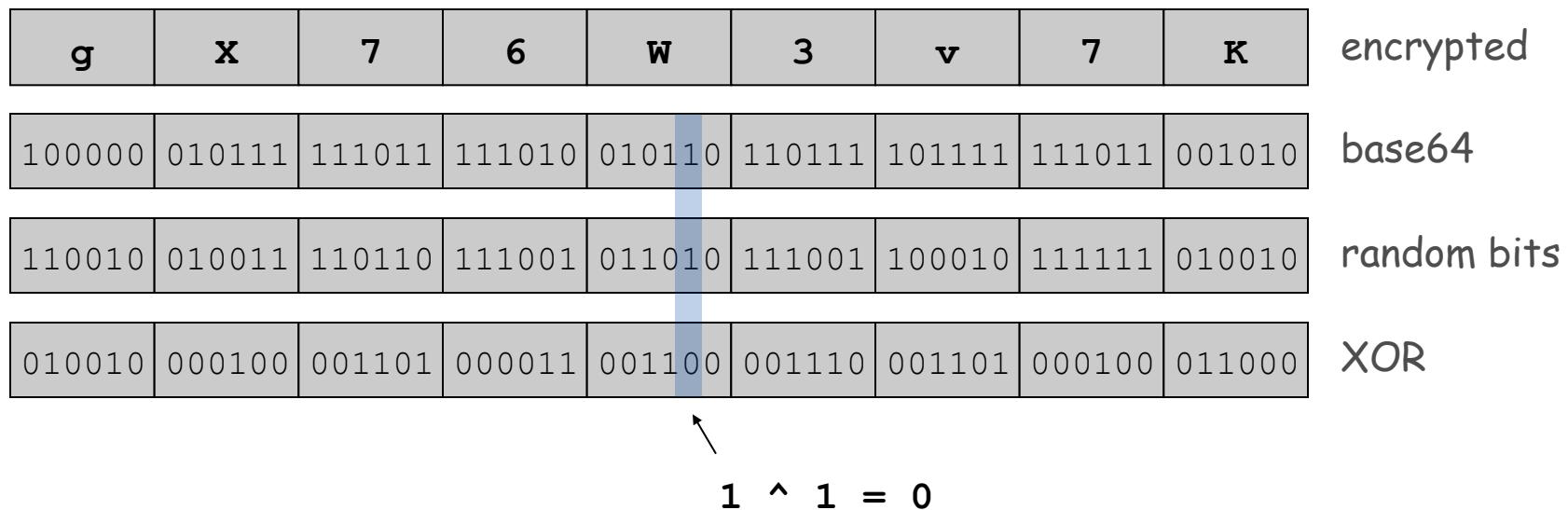
One-Time Pad Decryption

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



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	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

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

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

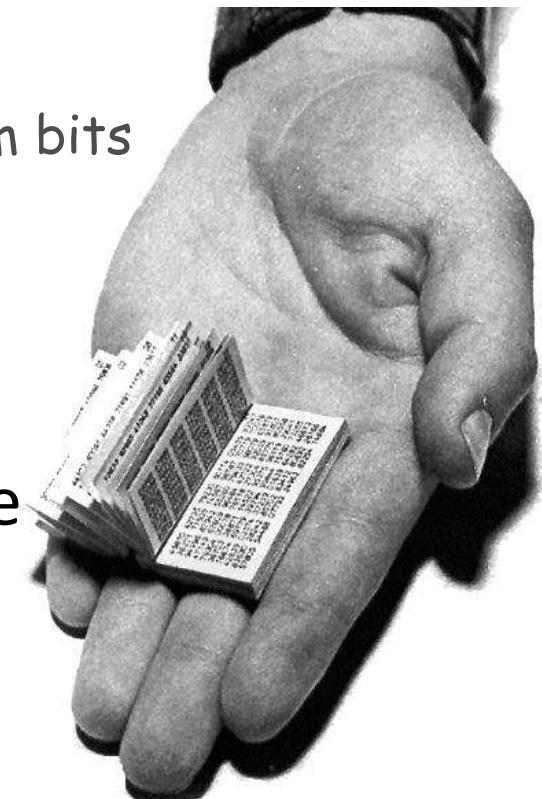
[Shannon, 1940s]



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

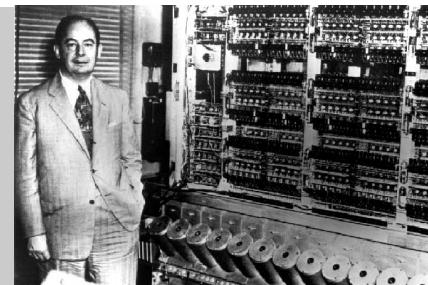
Practical middle-ground

- 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

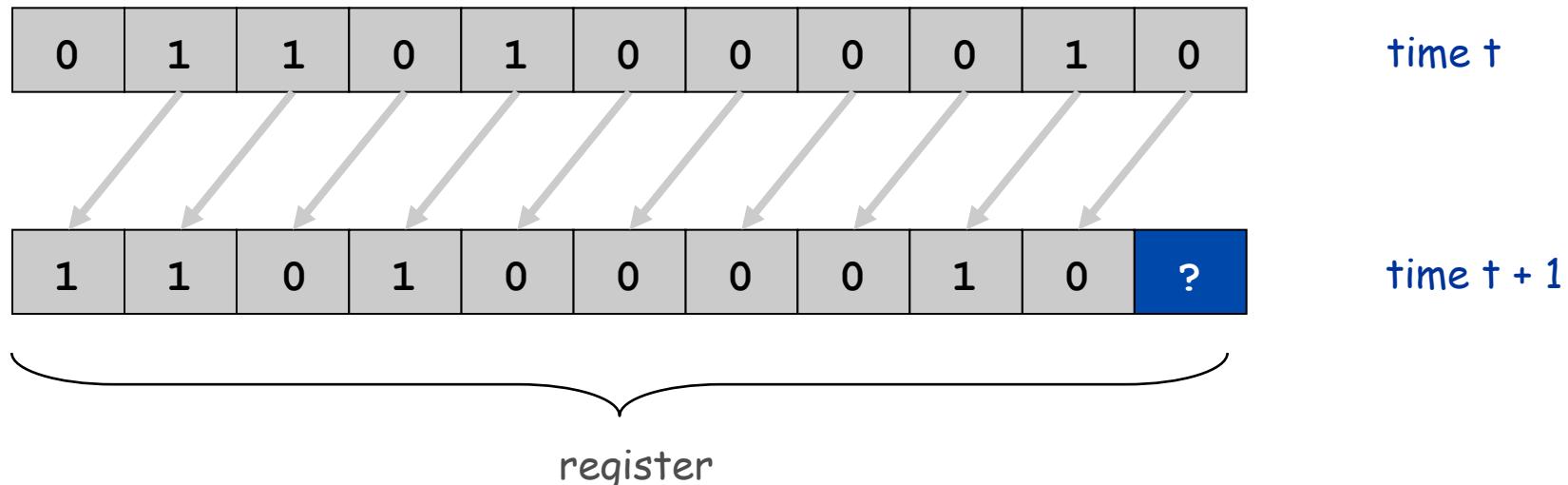
“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”
– Jon von Neumann (left)
– ENIAC (right)



Shift Register

Shift register terminology.

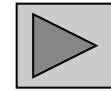
- Bit: 0 or 1
- Cell: storage element that holds one bit
- Register: sequence of cells
- Seed: initial sequence of bits
- Shift register: when clock ticks, bits propagate one position to left



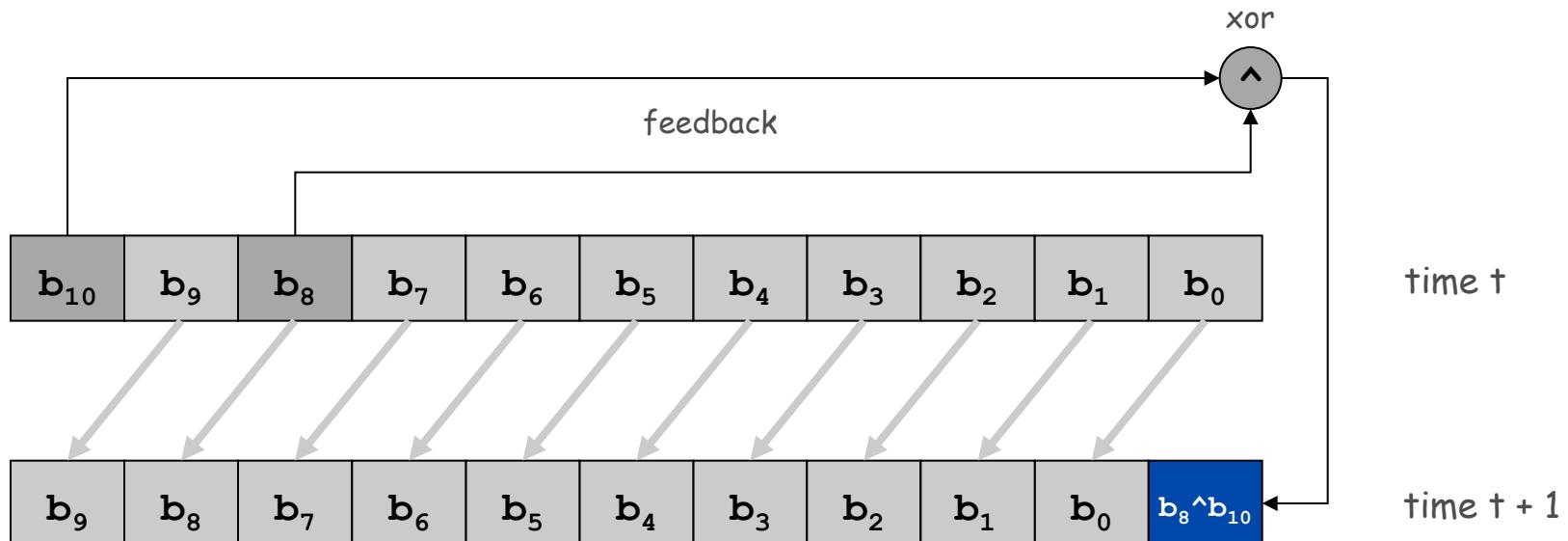
Linear Feedback Shift Register (LFSR)

{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  
011101000001010010001101010111000000101100000100111000101110110100101100110001001110101  
111111001100001111100011000011011110011001111010011110100111101001111011101010101000  
00000001000000001010000001000100001010100100000001101000001110010001101110101110100  
01010000101000100100010101101010000110000100111100101110011110111001001010111011  
00001010111001000010111010010010101100011110111011001010101111000000100110000101111  
100100100011101101011000110001110111101101010010110000110011100111110111100001010  
01100100011111010110000100011100101011100001101011001111101101100010110111011000101101  
01101010011110000111001100110111111010000000100100000101101000100110010101111100001  
000011001010011110001110001101101110110101101100001101110001110101101100011  
011001011101111001010110000011101100011010111000101010110100000110010000111110  
10011000100111110101110001000101101010011000000111110000110001100111101111100101000  
111000100110110101111011000100101110101100100011110001011001101001111100111000011110  
110011001011111110010000001110100001101001001110011011110101001000000110010000111110  
10000010010100010110001010011101000111010010110100110011111111100000000011000000  
1111000001100110001111111011000000101110000100101100101100111100111100111100011110011  
011001111101111100010100011010010110010111000110010110111100110100111100101110010110  
0011100111010111101011010010001100110101111110001000001101010001110000101101100100110  
1111011110100100110001101111101110100010101001010000011000100011110101011001000001  
11101000110010010111110110010001011110101001000001101101001111010101111010001000100  
010010101011000000011100000011011000011101100110101111000001000110001010111101010
```

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
                        ;}}
```

LFSR vs. General-Purpose Computer

Important properties

- Built from simple components
- Scales to handle huge problems
- Requires a deep understanding to use effectively

Basic Component	LFSR	Computer
control	start, stop, load	same
clock	regular pulse	2.8 GHz pulse
memory	11 bits	4 GB
input	seed	sequence of bits
computation	shift, XOR	logic, arithmetic, ...
output	pseudo-random bits	Sequence of bits

A Profound Idea

Programming Can write a Java program to simulate the operations of **any** abstract machine

- Basis for theoretical understanding of computation
- Basis for bootstrapping real machines into

```
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) {  
        LFSR lfsr = new LFSR("01101000010", 8);  
        for (int i = 0; i < 2000; i++)  
            StdOut.println(lfsr.step());  
    }  
}
```

```
% java LFSR  
11001001001111011011100101101  
0111001100010111110100100001  
00110100101111001100100111...
```

A Closing Question

Q. What is a random number?

LFSR does not produce random numbers

- It is a very simple deterministic machine
- But not obvious how to distinguish the bits it produces from random

Q. Are truly random processes found in nature?

- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?

Q. Or, is the natural world a (not-so-simple) deterministic machine?

“God does not play dice.”
— Albert Einstein

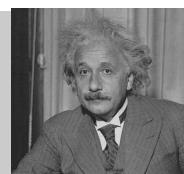


Image Steganography

The art of hiding secret messages in images

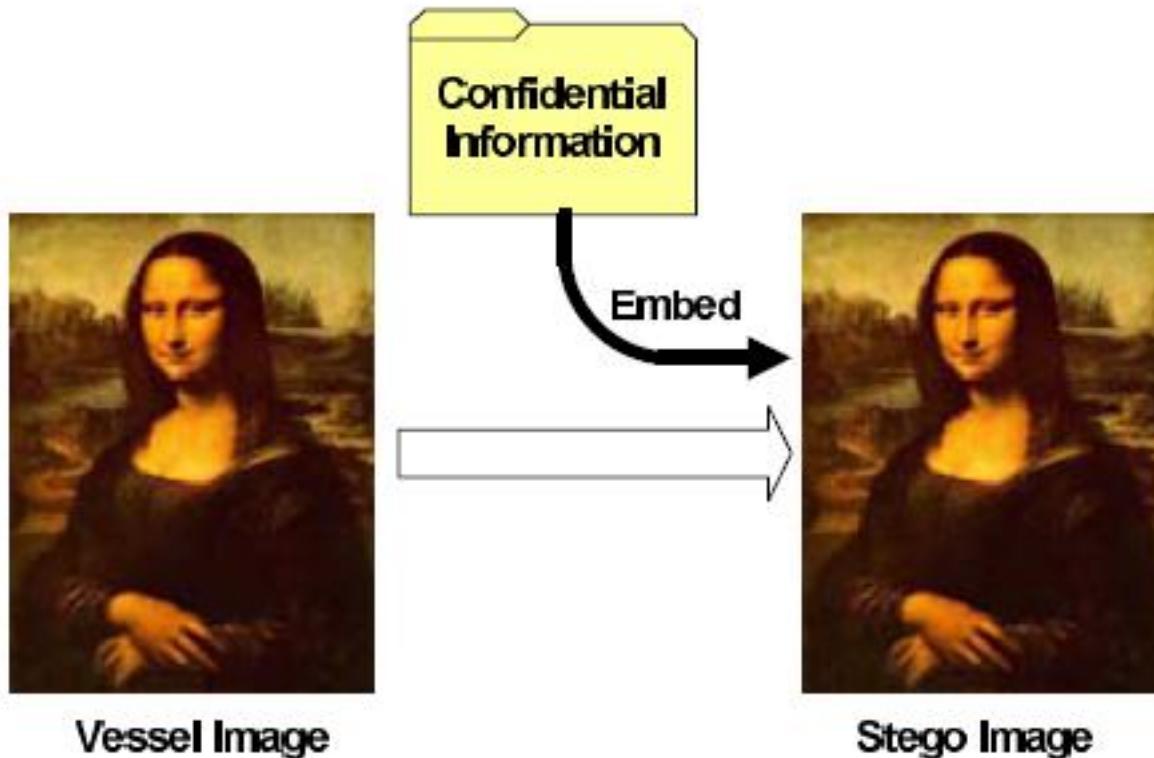


Image Steganography

The art of hiding secret messages in images

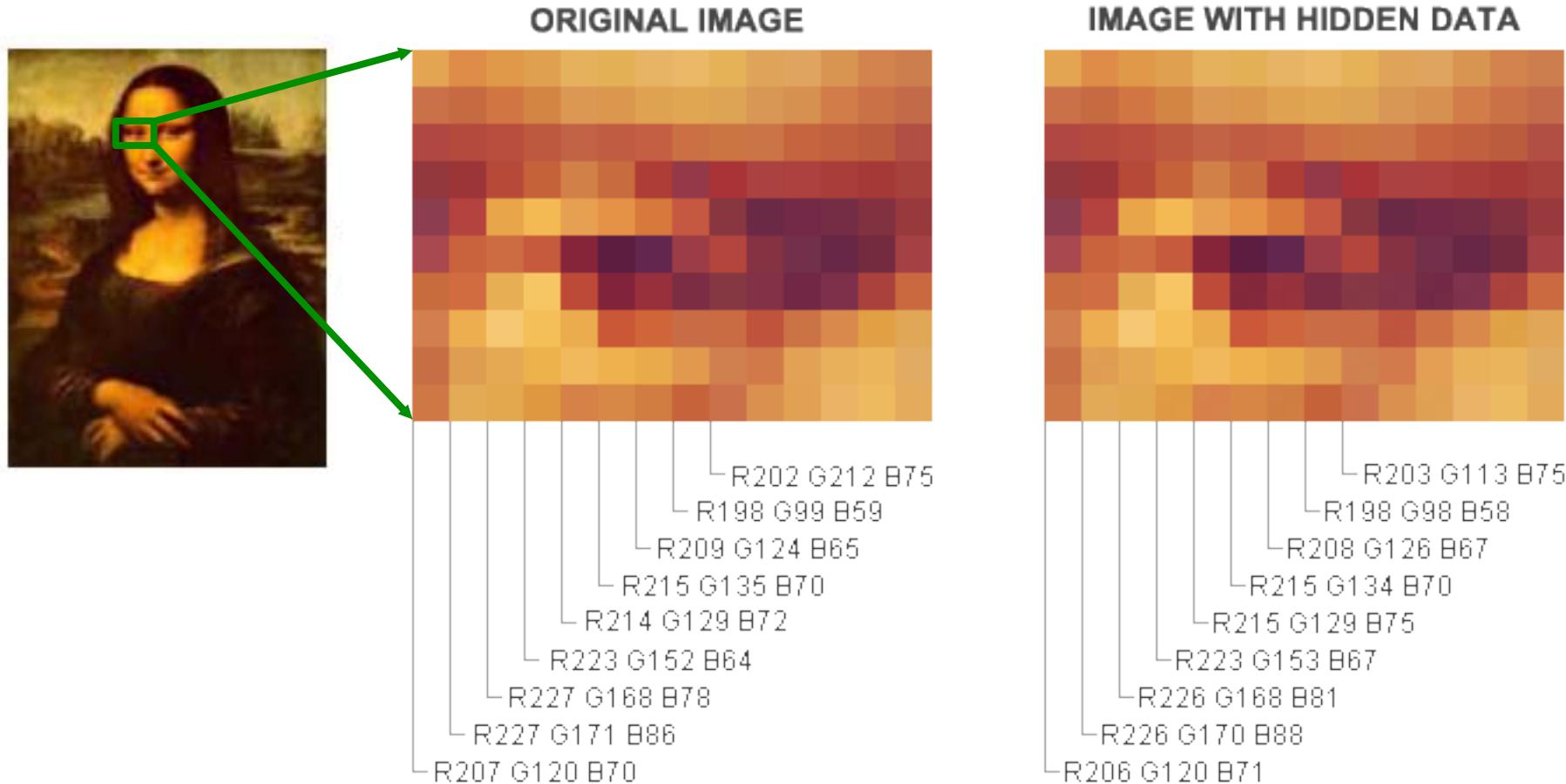
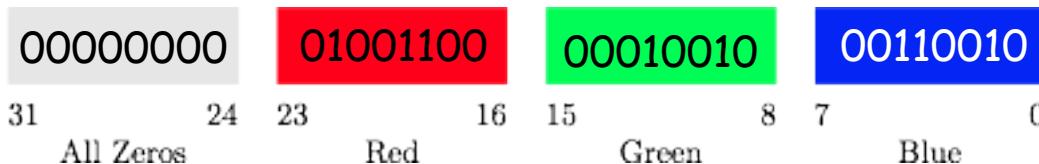


Image Steganography: How it Works

- Recall that each pixel of an image is 32 bits



- The right-most bit of each color is the least significant
 - Changing it by a small amount results in an imperceptible change to the color
- So, we can hide information in it!
- For simplicity, we'll only hide information in the blue channel

Image Steganography: How It Works

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

R207 G120 B71 ← 1000111

ORIGINAL IMAGE

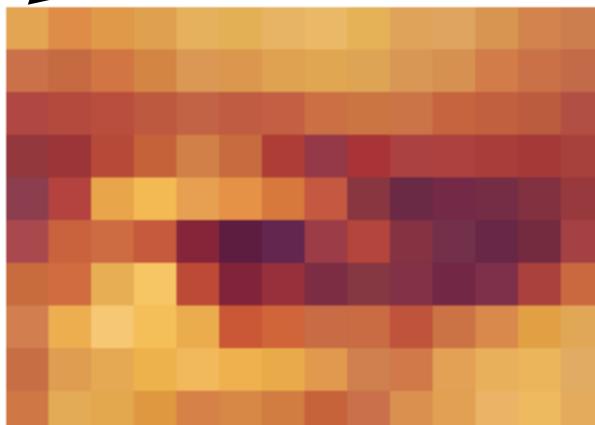


Image Steganography: How It Works

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

R207 G120 **B71** ← 1000111 ~~X~~



R207 G120 **B70** ← 1000110



Image Steganography on Printers

- It has been alleged that laser printer manufacturers, perhaps in collusion with federal agencies, insert steganographic signatures on every piece of paper printed on a laser printer.
 - These signatures encode information identifying the serial number of the printer as well as a timestamp and other information that can be used to trace the originator of a printout.

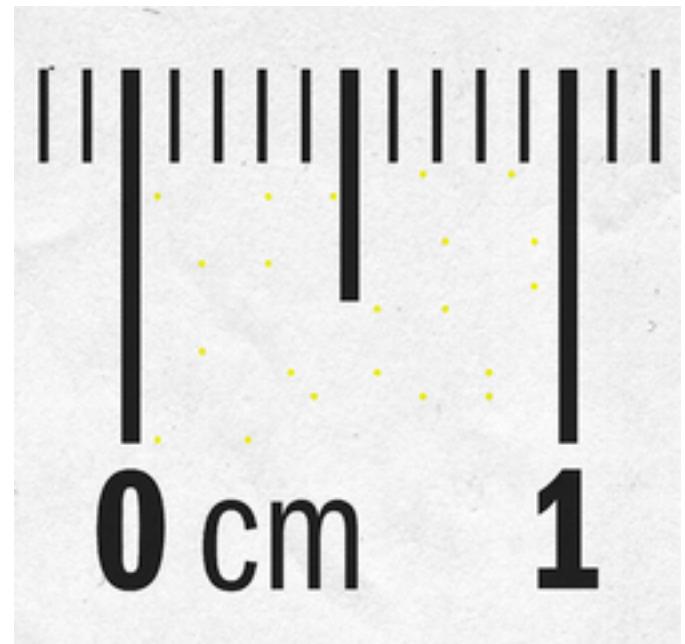
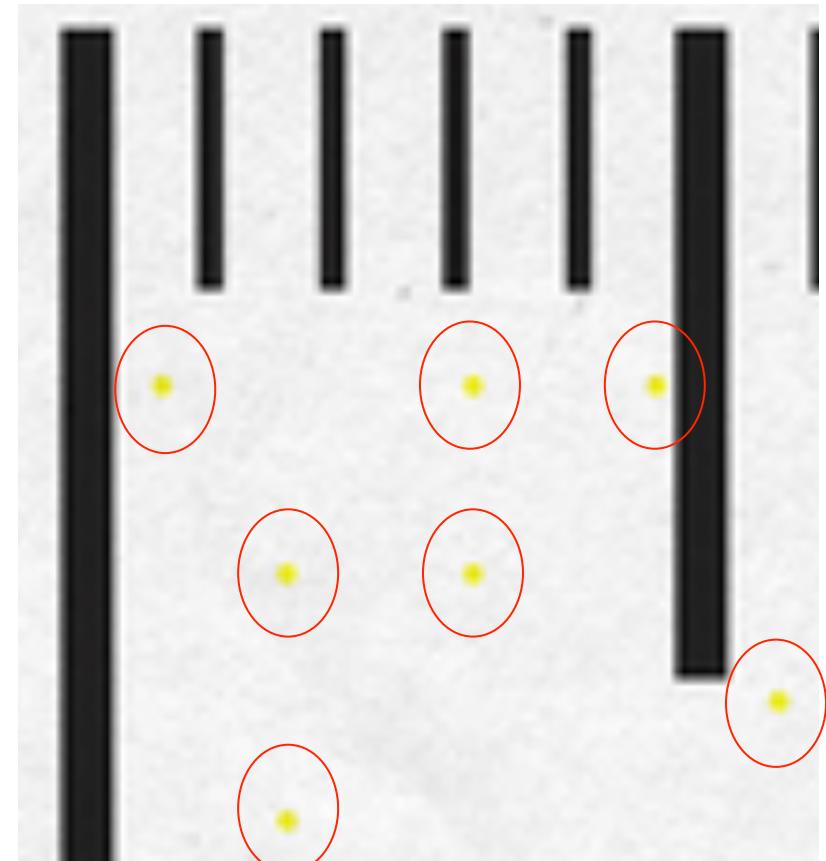


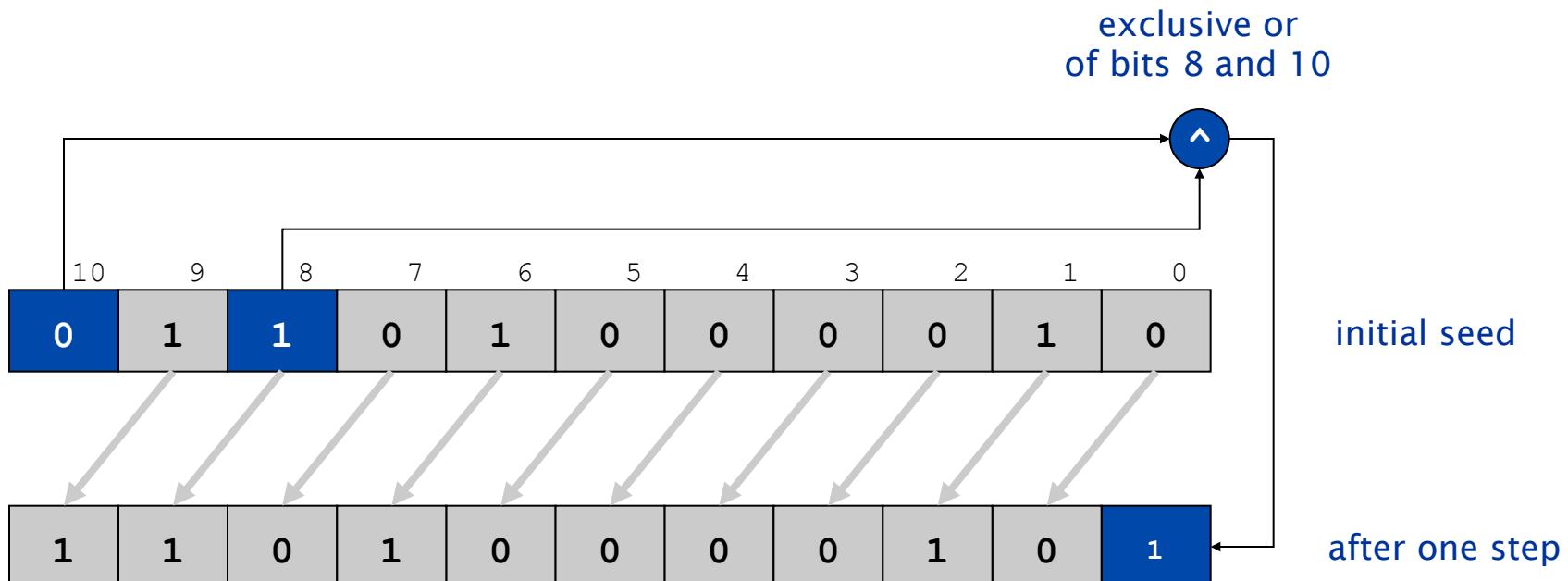
Image Steganography on Printers

- It has been alleged that laser printer manufacturers, perhaps in collusion with federal agencies, insert steganographic signatures on every piece of paper printed on a laser printer.
 - These signatures encode information identifying the serial number of the printer as well as a timestamp and other information that can be used to trace the originator of a printout.



Extra Slides

Linear Feedback Shift Register



One step of an 11-bit LFSR with initial seed 01101000010 and tap at position 8