Defending Computer Networks

Lecture 21: Symmetric Key Encryption

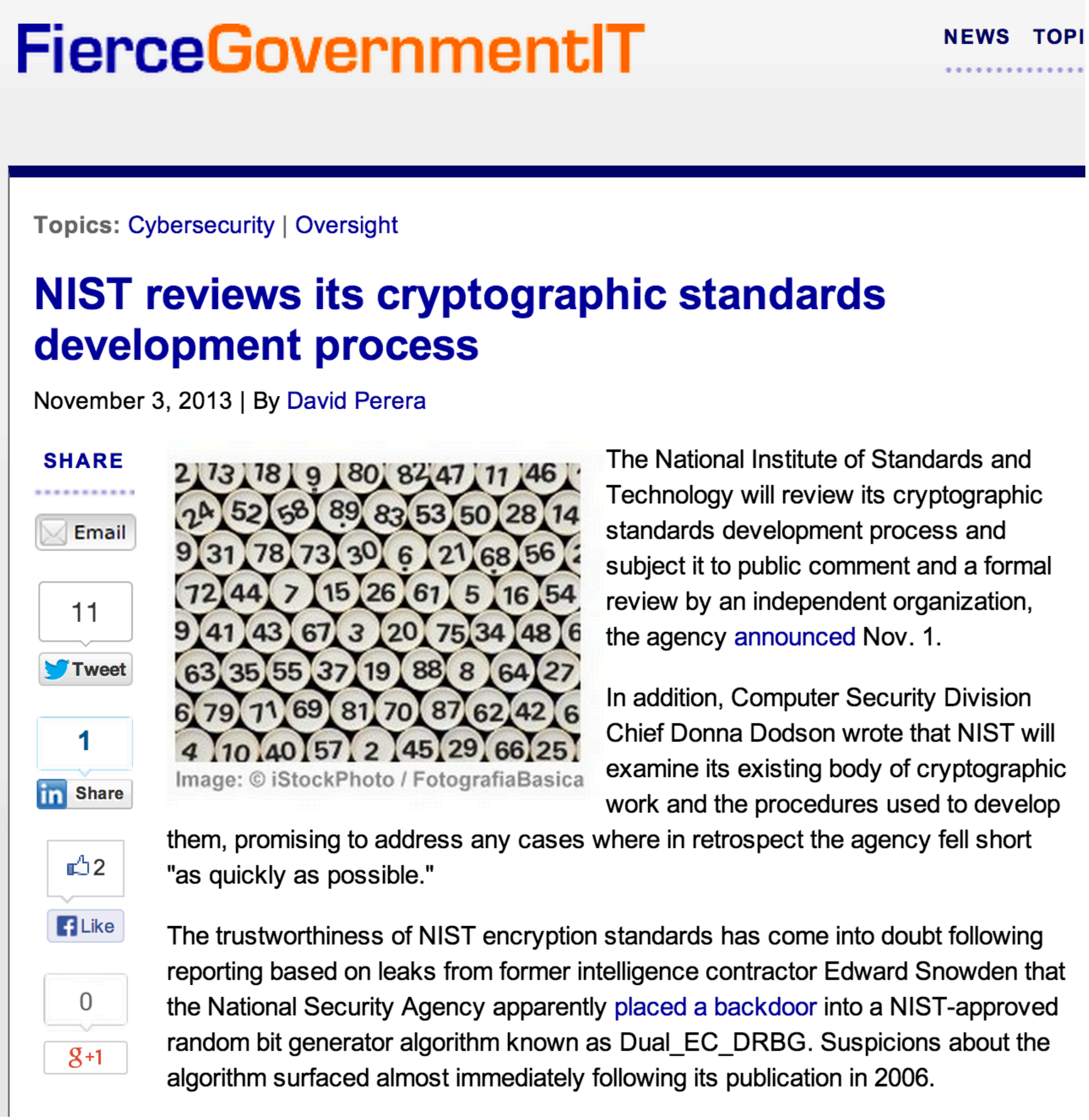
Stuart Staniford

Adjunct Professor of Computer Science

Logistics

- HW4 on website due end of today.
- Quiz 2 on Thursday

Latest News



Topics: Cybersecurity | Oversight

NIST reviews its cryptographic standards development process

November 3, 2013 | By David Perera

Image: © iStockPhoto / FotografiaBasica

The National Institute of Standards and Technology will review its cryptographic standards development process and subject it to public comment and a formal review by an independent organization, the agency announced Nov. 1.

In addition, Computer Security Division Chief Donna Dodson wrote that NIST will examine its existing body of cryptographic work and the procedures used to develop them, promising to address any cases where in retrospect the agency fell short "as quickly as possible."

The trustworthiness of NIST encryption standards has come into doubt following reporting based on leaks from former intelligence contractor Edward Snowden that the National Security Agency apparently placed a backdoor into a NIST-approved random bit generator algorithm known as Dual_EC_DRBG. Suspicions about the algorithm surfaced almost immediately following its publication in 2006.

SC MAGAZINE
FOR IT SECURITY PROFESSIONALS

More than half of corporate breaches go unreported

SC Magazine > News > College networks hit with highest incidence of malware infections, firm finds

Danielle Walker, Reporter
Follow @daniellewlkr

October 17, 2013

College networks hit with highest incidence of malware infections, firm finds

A cloud security company that probed its network of 50 million worldwide users found that colleges and universities most often fall victim to malware attacks.

College and university networks were 300 percent more likely to contain malware.

San Francisco-based OpenDNS discovered that higher education networks were 300 percent more likely to contain malware than government organizations or business entities that faced the same cyber attacks.

During the probe of its network, OpenDNS also found that malware called EXPIRO was the top threat impacting educational organizations.

Expiro is delivered to victims via exploits kits, which target users running vulnerable Java plug-ins or Adobe PDF installations.

“It's a file infector and it looks at a bunch of data on your machine, like web history or websites you've visited and the computer name,” Hubbard said of EXPIRO. “It then puts that information in an encrypted file and sends it to the attacker.”

Users are often infected with EXPIRO via drive-by download, he added.

Main Goals for Today

- Start on encryption – symmetric key
- General note
  - Cryptography is an enormous subject
  - Tens of thousands of careers over thousands of years devoted to it
  - Highly complex and mathematical
  - We will just barely scratch the surface
    - Wouldn’t be responsible in a course like this to say nothing
  - Not my area of expertise (at all)

Goals in Security

- Confidentiality
  - Information is kept secret except to those authorized to know
- Integrity
  - Information provided is correct, not altered
- Availability
  - Information is provided when it’s supposed to be (service is not denied)
- Cryptography primarily used for C&I
  - Can actually be an attack on A (ransomware)

Symmetric Key Encryption

- Using a shared secret to make messages unreadable.
- Same key used for encryption and decryption
- Ancient art – examples known from
  - Egypt 1900 BC
  - Mesopotamia 1500 BC
  - Probably almost as old as writing itself
- Still extensively used in practice

Caesar Cipher

Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher: XYZABCDEFGHIJKLMNOPQRSTUVW

Plaintext: the quick brown fox jumps over the lazy dog
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Here shift is key, -3 in this case

http://en.wikipedia.org/wiki/Caesar_cipher

Trivial To Implement

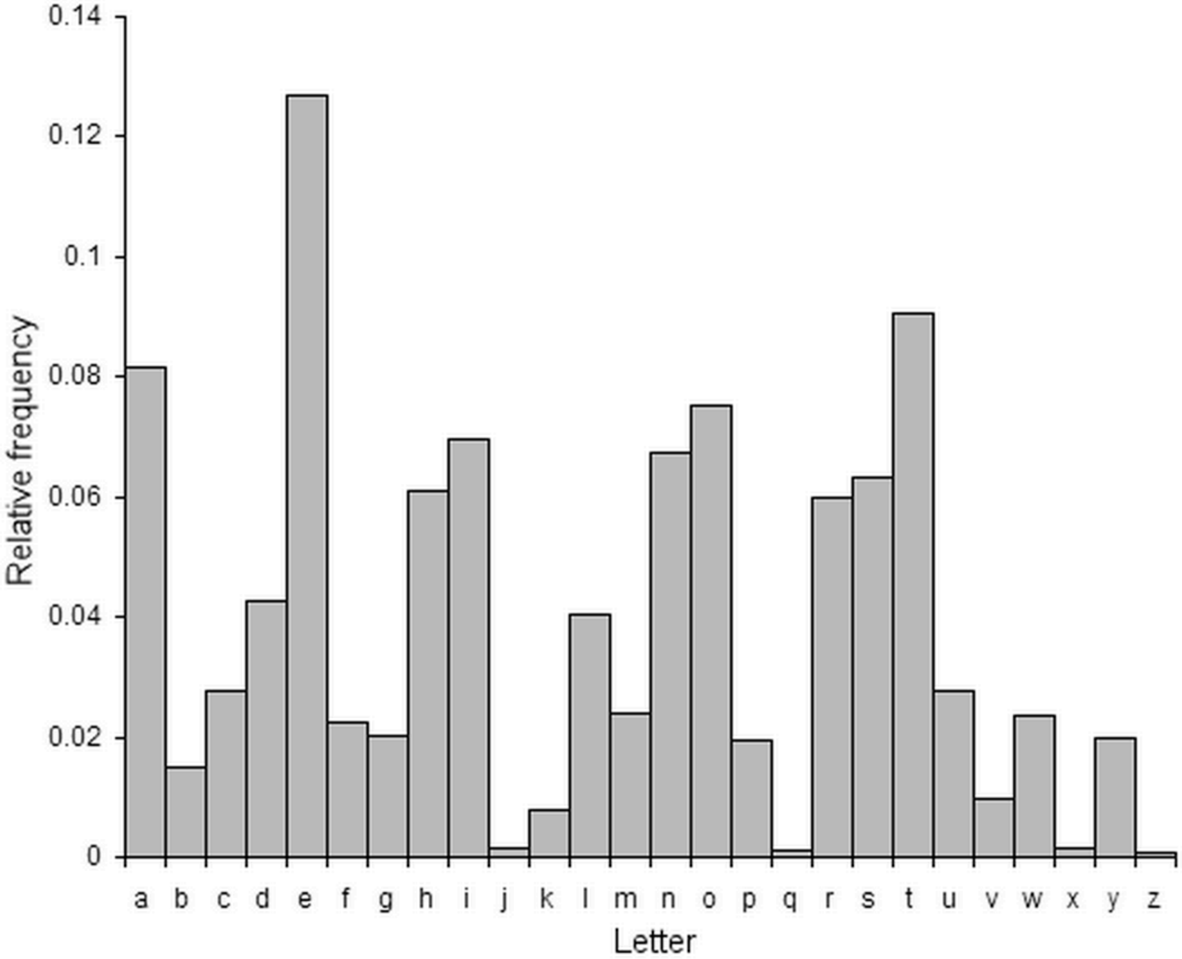
```
#include <stdio.h>
int main(int argc, char* argv[]) {
  char  buf[128];
  char* p;
  int   shift = 5;
  while(fgets(buf, 128, stdin)) {
    for(p = buf; *p; p++) {
      if(*p >= 'a' && *p <= 'z’) {
        *p += shift;
        if(*p > 'z')
          *p -= 26;
      }
      else if(*p >= 'A' && *p <= 'Z’) {
        *p += shift;
        if(*p > 'Z')
          *p -= 26;
      }
    }
    printf("%s\n", buf);
  }
}
```

This is basically modulo addition

How To Cipher-Analyze Caesar?

Lw lv xqnqrzq krz hiihfwlyh wkh Fdhvdu flskhu zdv dw wkh wlph, exw lw lv olnhob wr kdyh ehhq uhdvrqdeob vhfxuh, qrw ohdvw ehfdxvh prvw ri Fdhvdu'v hqhplhv zrxog kdyh ehhq loolwhudwh dqg rwkhuv zrxog kdyh dvvxphg wkdw wkh phvvdjhv zhuh zulwwhq lq dq xqnqrzq iruhljq odqjxdjh. Wkhuh lv qr uhfrug dw wkdw wlph ri dqb whfkqltxhv iru wkh vroxwlrq ri vlpsoh vxevwlwxwlrq flskhuv. Wkh hduolhvw vxuylylqj uhfrugv gdwh wr wkh 9wk fhqwxub zrunv ri Do-Nlqgl lq wkh Dude zruog zlwk wkh glvfryhub ri iuhtxhqfb dqdobvlv.

Frequency Analysis



Example of a known-ciphertext analysis

http://en.wikipedia.org/wiki/Caesar_cipher

Plain Text

It is unknown how effective the Caesar cipher was at the time, but it is likely to have been reasonably secure, not least because most of Caesar's enemies would have been illiterate and others would have assumed that the messages were written in an unknown foreign language. There is no record at that time of any techniques for the solution of simple substitution ciphers. The earliest surviving records date to the 9th century works of Al-Kindi in the Arab world with the discovery of frequency analysis.

XOR Cipher

- Exclusive-Or each byte with key
- Decryption means X-oring again
  - Which gives back the original value
- Widely used in malware currently with single byte key
  - Eg Aurora trojan download
  - Light obfuscation only
  - No stronger than Caesar cipher
  - Readily yields to frequency analysis

Entropy (Information Theoretic)

- Due to Shannon
- Loosely based on thermodynamic entropy
- Intuition: “how many bits of information are required to describe something”
- Suppose random variable X has possible values $x_1..x_n$ and P(X) prob distribution
- $H(X) = E(-\log(P(X))) = Sum[i..n, -P(x_i)*\log(P(x_i)]$

Entropy Example 1

- $H(X) = Sum[i..n, -P(x_i)*log(P(x_i)]$
- Suppose X is one byte and
  - all byte values equally likely
  - $P(x_i) = 1/256$
  - $\log P(x_i) = -8$
  - $H(X) = 8$
  - 8 bits of information

Entropy Example 2

- H(X) = Sum[i..n, -P(x_i)*log(P(x_i)]
- Suppose X is one byte and
  - Only one byte over occurs, say ‘a’
  - P(‘a’) = 1, else P(x_i) = 0
  - log P(‘a’) = 0,
  - H(X) = 0
  - No information in each byte – entirely predictable

Entropy Example 3

- H(X) = Sum[i..n, -P(x_i)*log(P(x_i)]
- Suppose X is one byte and
  - Only two bytes over occurs, say ‘a’ and ‘b’
  - P(‘a’) = 1/2, P(‘b’) = 1/2 else P(x_i) = 0
  - log P(‘a’) = log P(‘b’) = -1,
  - H(X) = 1
  - One bit of information in each byte
    - Either ‘a’ or ‘b’ – code with 0 or 1.

Entropy of English

- Prior examples assume successive picks are independent
- Not true of natural languages, eg “qu”
- Have to account for these dependencies by making the state vector larger
- English has about 1 bit per character
- Highly relevant for cryptanalysis

One Time Pad

- Like Caesar Cipher
  - – Modulo addition on characters, but
  - – Instead of a single constant shift
  - – Key is random, different on each character
  - – Requires a key that is as long as the plaintext
  - – The “pad” – keystream – is shared in advance
  - – Pad has full entropy of the alphabet

One Time Pad Example

Supposing we number letters 0-25, and make space 26

Key:	3	15	22	11	19	1	8	25	4	22	7	13	26	12	9	3
Plaintext:	T	H	E		Q	U	I	C	K		B	R	O	W	N	
Ciphertext:	W	W		K	I	V	Q	A	O	V	I	D	N	H	W	C

What about frequency analysis now?

One Time Pad Properties

- Shown by Shannon (1949) that
  - **If** the key is genuinely random/completely unpredictable
  - Then ciphertext contains no information about the plain text.
- Practical difficulties
  - Distributing full length one time pad
    - If you reuse pad, after a while, cryptanalysis possible
  - Alternatively, use an algorithmic RNG
    - Potentially analyzable if algorithm can be discovered

AES

- AES = Advanced Encryption Standard
- Published by US NIST (2001)
  - Following an open competition
  - Including public crypto-analysis attempts
- Replaced earlier (1977) DES standard
- Widely used in practice
  - Fairly fast
- Currently believed secure
  - Some slight uncertainty following Snowden
    - https://www.schneier.com/blog/archives/2012/03/can_the_nsa_bre.html

AES Design

- Block oriented cipher
  - Takes blocks of plaintext 128 bits at a time
    - 16 bytes
  - Other sizes must be padded
- Key sizes of 128, 192, and 256 bits
- What's known as a substitution-permutation network
- Repeated numerous rounds
- Will sketch how it works while skipping worst of the math

AES Rounds

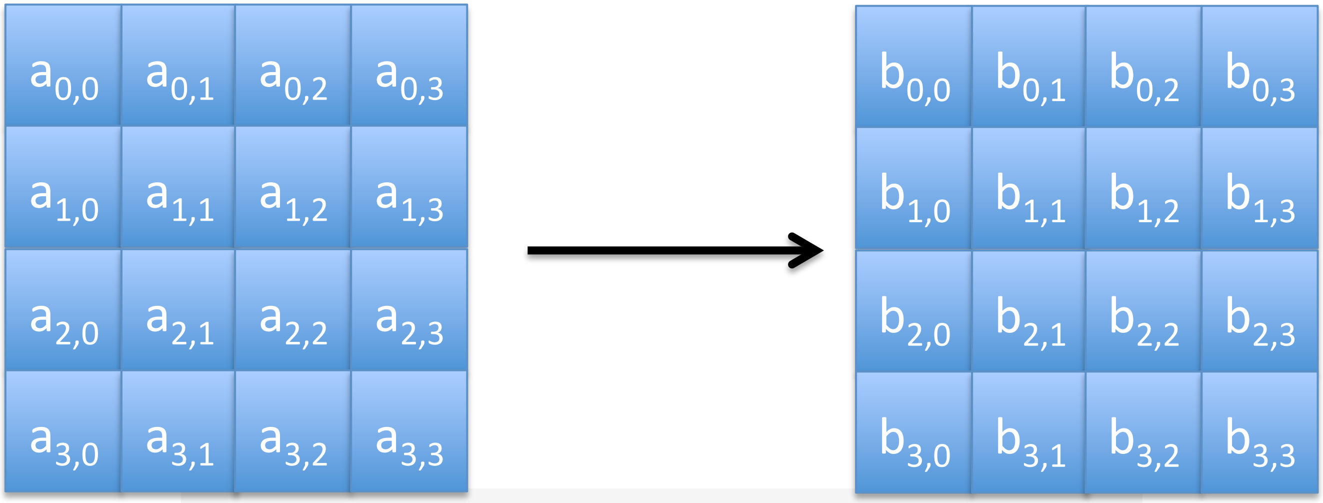
- The basic cipher operations are performed repeatedly on each block
  - 128 bit keys, 10 rounds
  - 192 bit keys, 12 rounds
  - 256 bit keys, 14 rounds
- Each round uses a different key
  - Derived from the master key using a set of complex algebraic operations
  - Depending on the algebra of finite (Galois) fields.

AES State Matrix

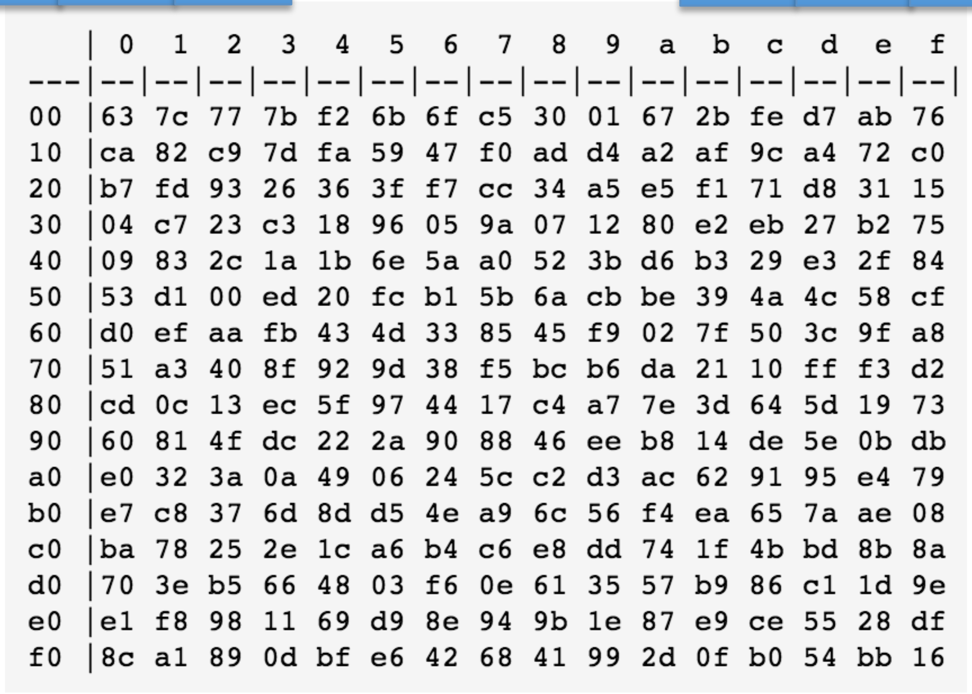
- Each 16 byte block is arranged in a 4x4 state matrix
- Used to keep track of what block will turn into

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

First Step: S-box (substitution)

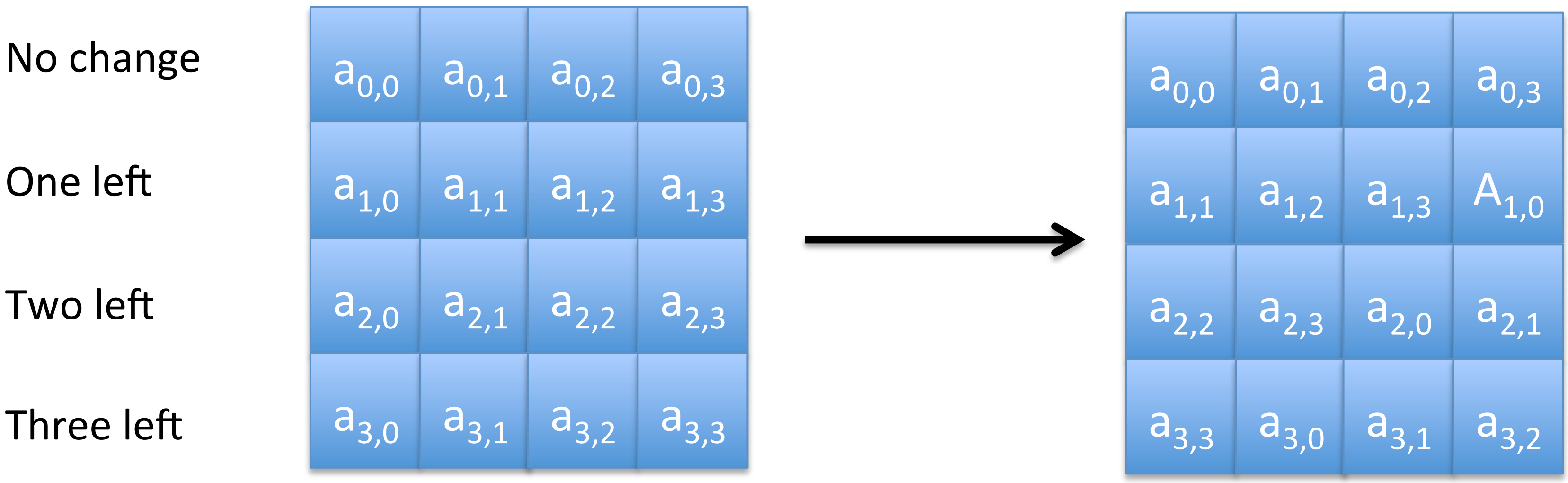


$$\begin{array}{cccc} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{array} \longrightarrow \begin{array}{cccc} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{array}$$



```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75
40 |09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
50 |53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
60 |d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8
70 |51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
80 |cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
90 |60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db
a0 |e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79
b0 |e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08
c0 |ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
d0 |70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e
e0 |e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
f0 |8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16
```

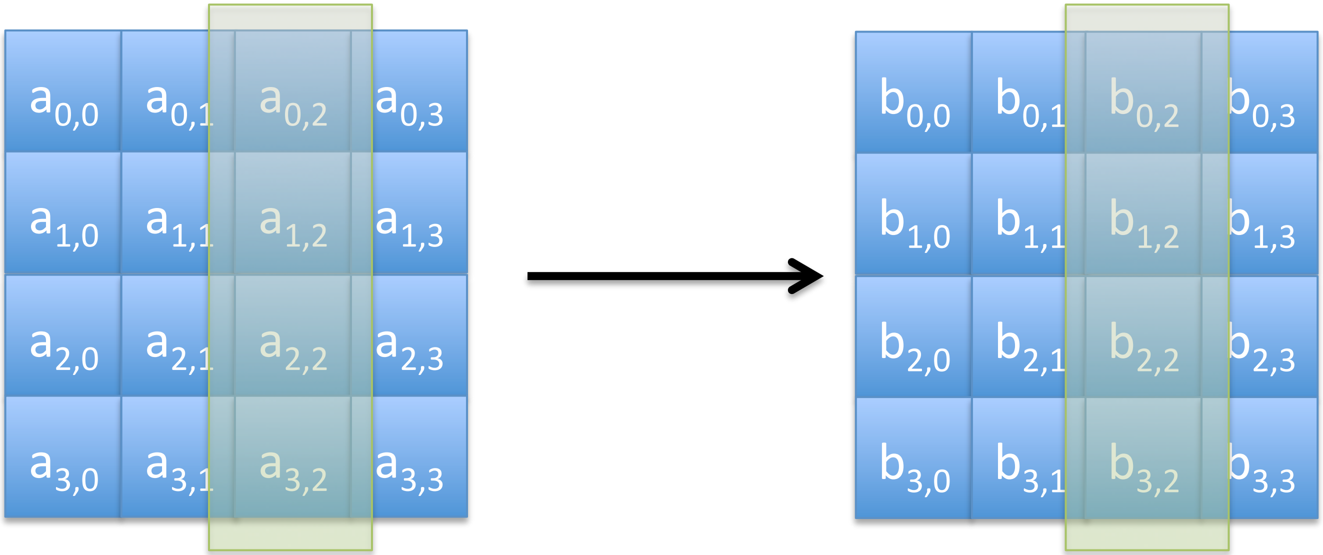
Shift Rows Step



No change	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
One left	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
Two left	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
Three left	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$A_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

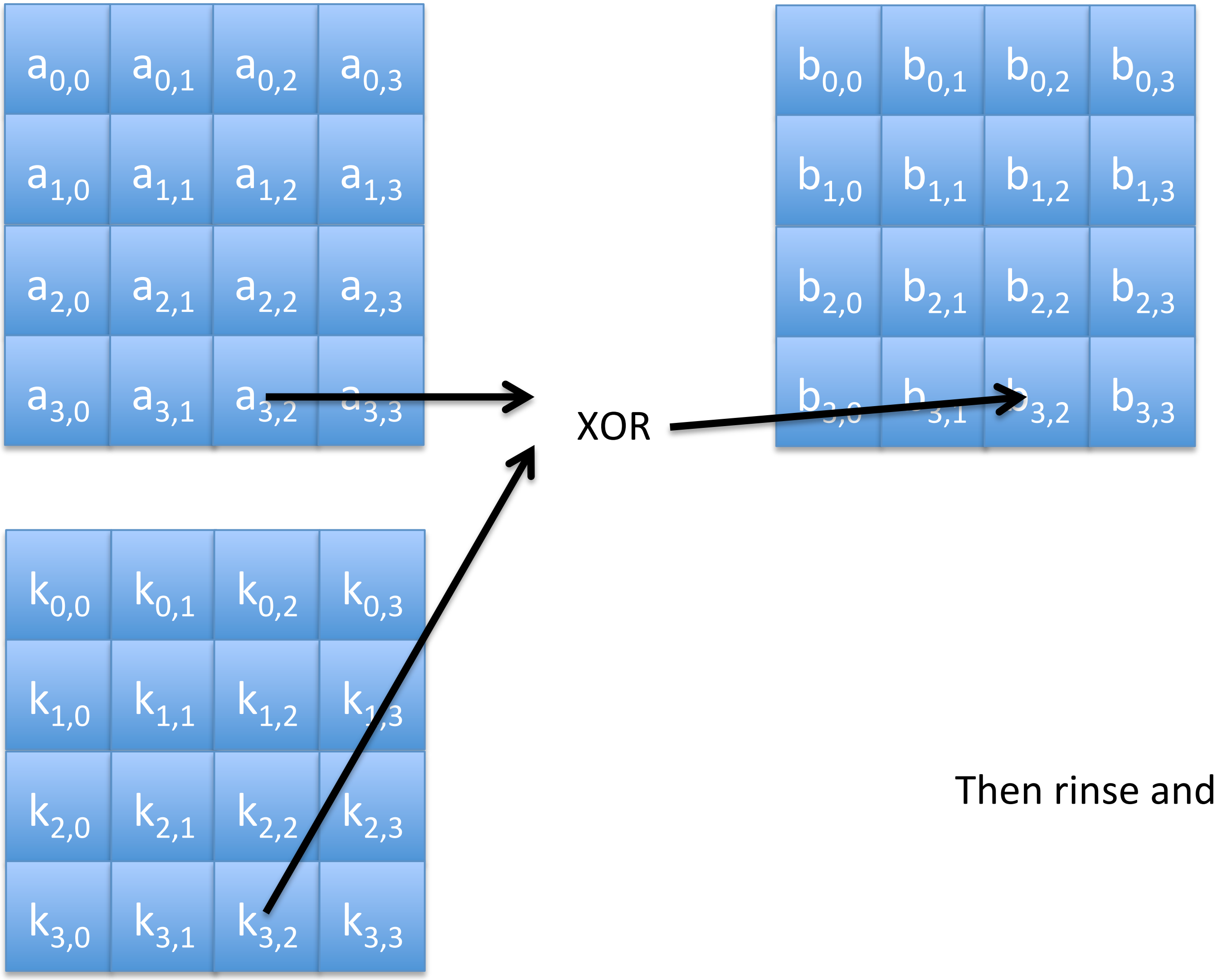
Mix Columns Step



$$\begin{matrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{matrix} \longrightarrow \begin{matrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{matrix}$$

Each column is transformed by a complex (but fixed) algebraic equation that takes input from all four bytes and incorporates them into each output byte

Add Round Key



$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

XOR

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

Then rinse and repeat…