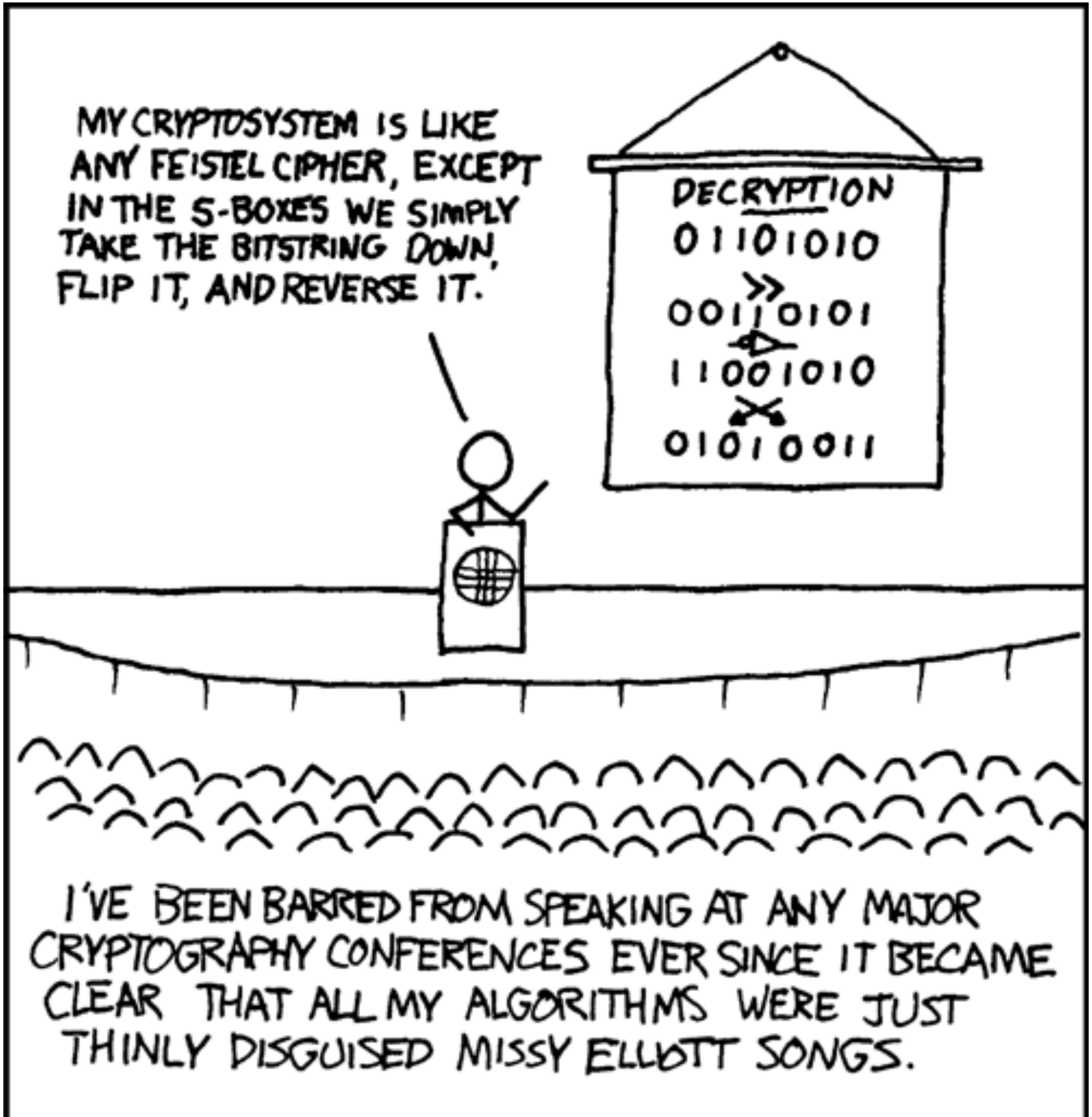
CS 465
Computer Security

AES

Daniel Zappala, adapted
Fall 2018


MY CRYPTOSYSTEM IS LIKE ANY FEISTEL CIPHER, EXCEPT IN THE S-BOXES WE SIMPLY TAKE THE BITSTRING DOWN, FLIP IT, AND REVERSE IT.
DECRYPTION
01101010
00110101
11001010
01010011
I'VE BEEN BARRED FROM SPEAKING AT ANY MAJOR CRYPTOGRAPHY CONFERENCES EVER SINCE IT BECAME CLEAR THAT ALL MY ALGORITHMS WERE JUST THINLY DISGUISED MISSY ELLIOTT SONGS.

Programming Lab #1

- Implement AES
- Use the FIPS 197 spec as your guide
  - Avoid looking at code on the Internet
  - Challenge yourself to implement the algorithm based on sources mentioned in the lab specification
  - The standard provides programming language independent pseudo-code
  - 20 pages at the end of the spec has complete, step-by-step debugging information to check your solution

Resources

- FIPS 197 Spec
- Wikipedia articles on AES and Finite Field Arithmetic
- YouTube video
- Stick figure guide to AES

C Data Structures

- uint8_t b; /* byte — 8 bits*/
- uint16_t w; /* word — 16 bits */
- uint32_t l; /* word — 32 bits */
- uint8_t state[4][Nb]; /* two dimensional array */
- void method(uint8_t in[]); /* params */
- void method(uint8_t state[4][Nb]);

Binary Operations

<table>
<tr><th>Operation</th><th>Name</th><th>Example</th><th>Result</th></tr>
<tr><td>a & b</td><td>and</td><td>0x53 & 0x31</td><td>0x11</td></tr>
<tr><td>a | b</td><td>or</td><td>0x53 | 0x31</td><td>0x73</td></tr>
<tr><td>a ^ b</td><td>xor</td><td>0x53 ^ 0x31</td><td>0x62</td></tr>
<tr><td>a << n</td><td>left shift</td><td>0x53 << 1</td><td>0xa6</td></tr>
<tr><td>a >> n</td><td>right shift</td><td>0x53 >> 1</td><td>0x29</td></tr>
</table>

AES Parameters

- Nb – Number of columns in the State
  - For AES, Nb = 4
- Nk – Number of 32-bit words in the Key
  - For AES, Nk = 4, 6, or 8
- Nr – Number of rounds (function of Nb and Nk)
  - For AES, Nr = 10, 12, or 14

Block Cipher

- AES is a block cipher — encryption and decryption on a 4x4 block of bytes
- Intermediate representations of the cipher are stored in the *state* variable
- Bytes stored into and taken out of the state in column order
- See Figure 3

Let’s look at the big picture…

See Figure 3 for State

See Figure 5 for Cipher

Overview

- Finite Field Arithmetic
- Key Expansion
- Transformations
    - AddRoundKey
    - SubBytes
    - ShiftRows
    - MixColumns

Finite Field Arithmetic

Finite Field Arithmetic

- Finite Fields are a mathematical concept.
- They consist of a finite set, an addition (+) operator, and a multiplication (*) operator.
- Addition and multiplication can be defined as any operation
- In AES, finite field arithmetic is done using a byte (8-bits, unsigned)

Finite Fields

- AES uses the finite field $GF(2^8)$
  - Galois Field — finite set of numbers with defined operations (addition, multiplication)
  - Polynomials of degree 8
  - $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$
  - $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$
- Byte notation for the element: $x^6 + x^5 + x + 1$
  - $0x^7 + 1x^6 + 1x^5 + 0x^4 + 0x^3 + 0x^2 + 1x + 1$
  - {01100011} – binary
  - {63} – hex

Finite Field Arithmetic

- Addition (XOR)
  - $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$
  - $\{01010111\} \oplus \{10000011\} = \{11010100\}$
  - $\{57\} \oplus \{83\} = \{d4\}$
- Multiplication is tricky
  - Study section 4.2 in the spec
  - In 4.2.1, a paragraph describes what your implementation will do. Study it. Difficult to interpret.

Finite Field Multiplication (·)

{57} · {83} = {c1}

Step 1: multiply

Identical terms cancel

$(x^6 + x^4 + x^2 + x + 1)\ (x^7 + x + 1)$

$$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

```
          1010111
*        10000011
         ________
          1010111
         1010111
10101110000 0
______________
10101101111001
```

Finite Field Multiplication (·)

{57} · {83} = {c1}

(Not the product of two polynomials)

Irreducible Polynomial

Step 2: modulo

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $(x^8 + x^4 + x^3 + x + 1)$

$= x^7 + x^6 + 1$

$$
\begin{array}{r|l}
 & x^5 + x^3 \\
\hline
x^8 + x^4 + x^3 + x + 1 & x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\
 & x^{13} + \phantom{x^{11} +} x^9 + x^8 + x^6 + x^5 \\
\cline{2-2}
 & \phantom{x^{13} +} x^{11} + \qquad\qquad\qquad x^4 + x^3 + 1 \\
 & \phantom{x^{13} +} x^{11} + \qquad x^7 + x^6 + \quad x^4 + x^3 \\
\cline{2-2}
 & \qquad\qquad\qquad x^7 + x^6 + \qquad\qquad\quad + 1
\end{array}
$$

Finite Field Multiplication (·)

{57} · {83} = {c1}

(Not the product of two polynomials)

Irreducible Polynomial

Step 2: modulo

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo ($x^8 + x^4 + x^3 + x + 1$)

$= x^7 + x^6 + 1$

```
 10101101111001 mod 100011011
^100011011
 ______________
 00100000011001
^  100011011
 ______________
   000011000001
```

Efficient Finite Field Multiply

- There's a better way
  - Patterned after the divide and conquer modular exponentiation algorithm (CS 312)
- xtime() — very efficiently multiplies its input by {02}
  - *It follows that multiplication by x (i.e., {00000010} or {02}) can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with {1b}.*
  - This is the same as multiplying a polynomial by x – what is the binary or hex representation of the polynomial x?
- Multiplication by higher powers can be accomplished through repeated applications of xtime()

Efficient Finite Field Multiply

Example: {57} • {13}

{57} • {02} = xtime({57}) = **{ae}**

{57} • {04} = xtime(**{ae}**) = **{47}**

{57} • {08} = xtime(**{47}**) = **{8e}**

{57} • {10} = xtime(**{8e}**) = {07}

These are hexadecimal numbers!
10 in hex is 16 in decimal.

{57} • **{13}** = {57} • (**{01}** ⊕ **{02}** ⊕ **{10}**)

= ({57} • {01}) ⊕ ({57} • {02}) ⊕ ({57} • {10})

= {57} ⊕ {ae} ⊕ {07}

= {fe}

Efficient Finite Field Multiply

Turn this into a method:

{57} • **{1}** = **{57}**

{57} · **{2}** = xtime({57}) = ({0101 0111} << 1) = {1010 1110} = **{AE}** (high bit set? no)

{57} · {4} = xtime({AE}) = ({1010 1110} << 1) = {1 0101 1100} = {15C} (high bit set? yes)

drop high bit and xor {1B} (e.g. xor {11B}) = {15C} ⊕ {11B} = {47}

{57} · {8} = xtime({47}) = {0100 0111} << 1 = {8E} (high bit set? no)

{57} · **{10}** = xtime({8E}) = ({1000 1110} << 1) ⊕ {11B} = **{07}**

{57} * **{13}** = **{57}** ⊕ **{AE}** ⊕ **{07}**

Key Expansion

5.2

Figure 11

Definition of SubWord()

Sbox

Definition of RotWord()

Rcon

AddRoundKey

5.1.4

State is bytes, key schedule is words

SubBytes

5.1.1

Figure 6

Figure 7

ShiftRows

5.1.2

Figure 8

MixColumns

5.1.3

Figure 9

Equations above Figure 9

Finite Field Multiply