Original Source:
http://www.cs.sjsu.edu/~stamp/infosec/

Part I: Crypto

Chapter 3:
Symmetric Key Crypto

Chapter 3: Symmetric Key Crypto

The chief forms of beauty are order and symmetry…
— Aristotle

"You boil it in sawdust: you salt it in glue:
You condense it with locusts and tape:
Still keeping one principal object in view —
To preserve its symmetrical shape."
— Lewis Carroll, *The Hunting of the Snark*

Symmetric Key Crypto

- Stream cipher — based on one-time pad
  - Except that key is relatively short
  - Key is stretched into a long **keystream**
  - Keystream is used just like a one-time pad
- Block cipher — based on codebook concept
  - Block cipher key determines a codebook
  - Each key yields a different codebook
  - Employs both “confusion” and “diffusion”

Stream Ciphers

Stream Ciphers

- Once upon a time, not so very long ago, stream ciphers were the king of crypto
- Today, not as popular as block ciphers
- We’ll discuss two stream ciphers…
- A5/1
  - Based on shift registers
  - Used in GSM mobile phone system
- RC4
  - Based on a changing lookup table
  - Used many places

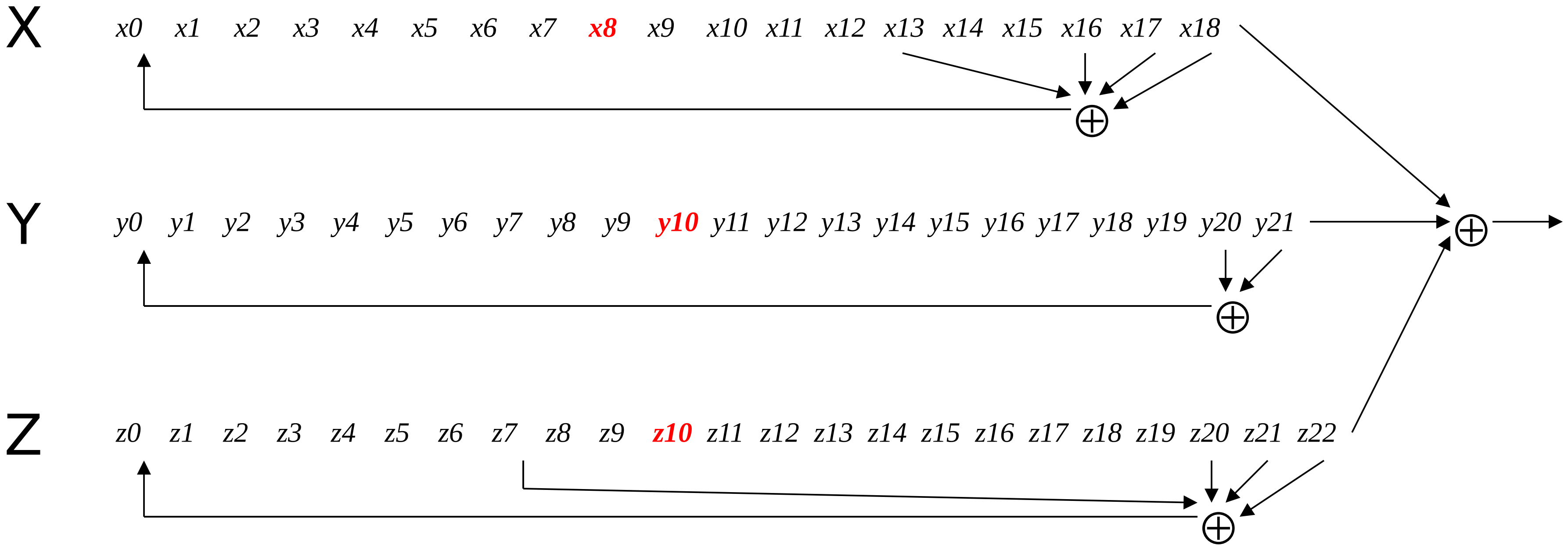
A5/1: Shift Registers

- A5/1 uses 3 *shift registers*
  - X: 19 bits $(x0, x1, x2, \ldots, x18)$
  - Y: 22 bits $(y0, y1, y2, \ldots, y21)$
  - Z: 23 bits $(z0, z1, z2, \ldots, z22)$

A5/1: Keystream

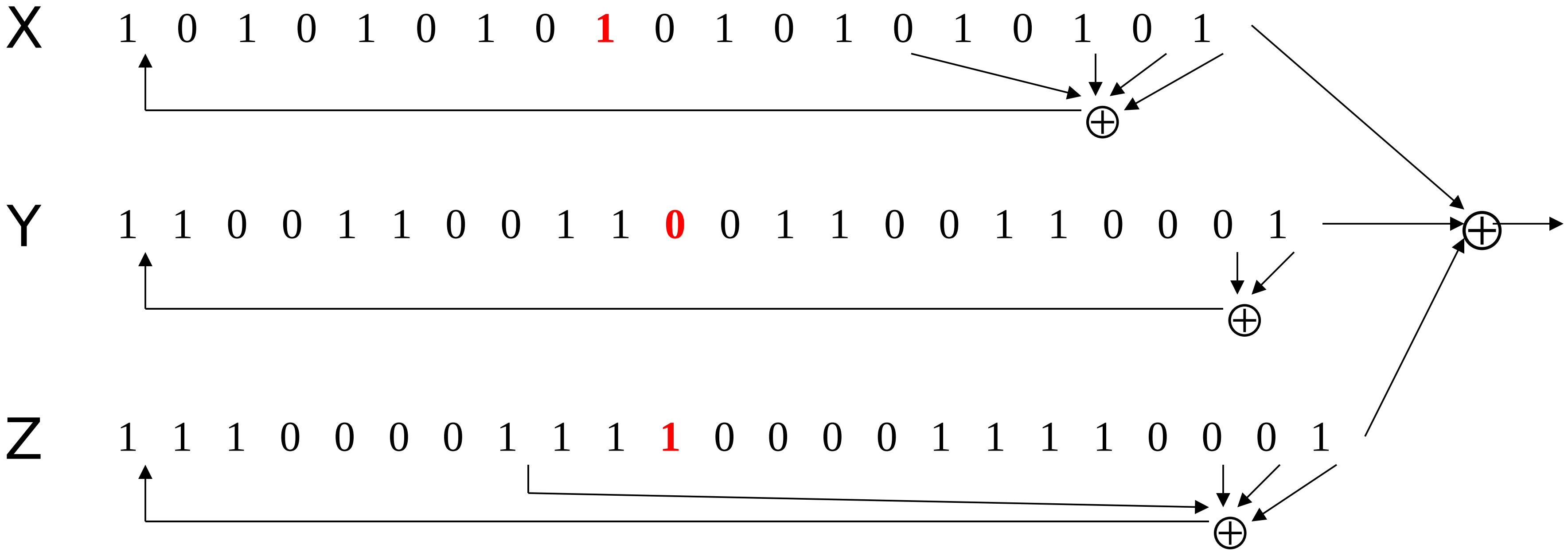
- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X *steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y *steps*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ and $y_0 = t$
- If $z_{10} = m$ then Z *steps*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$
- Keystream **bit** is $x_{18} \oplus y_{21} \oplus z_{22}$

A5/1



- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on $\mathrm{maj}(x8, y10, z10)$
- Keystream bit is XOR of rightmost bits of registers

A5/1



- In this example, $m = \text{maj}(x8, y10, z10) = \text{maj}(\mathbf{1},\mathbf{0},\mathbf{1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

Shift Register Crypto

- Shift register crypto efficient in hardware
- Often, slow if implement in software
- In the past, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used some
  - Resource-constrained devices

RC4

- A self-modifying lookup table
- Table always contains a permutation of the byte values 0,1,…,255
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a **byte**
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware

RC4 Initialization

- S[] is permutation of 0,1,...,255
- key[] contains N bytes of key

```
for i = 0 to 255
   S[i] = i
   K[i] = key[i (mod N)]
next i
j = 0
for i = 0 to 255
   j = (j + S[i] + K[i]) mod 256
   swap(S[i], S[j])
next i
i = j = 0
```

RC4 Keystream

- For each keystream byte, swap elements in table and select byte

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```

- Use keystream bytes like a one-time pad
- **Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists

Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - Shamir declared "the death of stream ciphers"
  - May be greatly exaggerated…

Block Ciphers

(Iterated) Block Cipher

- Plaintext and ciphertext consist of fixed-sized blocks
- Ciphertext obtained from plaintext by iterating a **round function**
- Input to round function consists of ***key*** and ***output*** of previous round
- Usually implemented in software

Feistel Cipher: Encryption

- **Feistel cipher** is a type of block cipher, not a specific block cipher
- Split plaintext block into left and right halves: $P = (L_0, R_0)$
- For each round $i = 1,2,\ldots,n$, compute

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

  where F is **round function** and K_i is **subkey**
- Ciphertext: $C = (L_n, R_n)$

Feistel Cipher: Decryption

- Start with ciphertext $C = (L_n, R_n)$
- For each round $i = n, n-1, \ldots, 1$, compute

  $R_{i-1} = L_i$

  $L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$

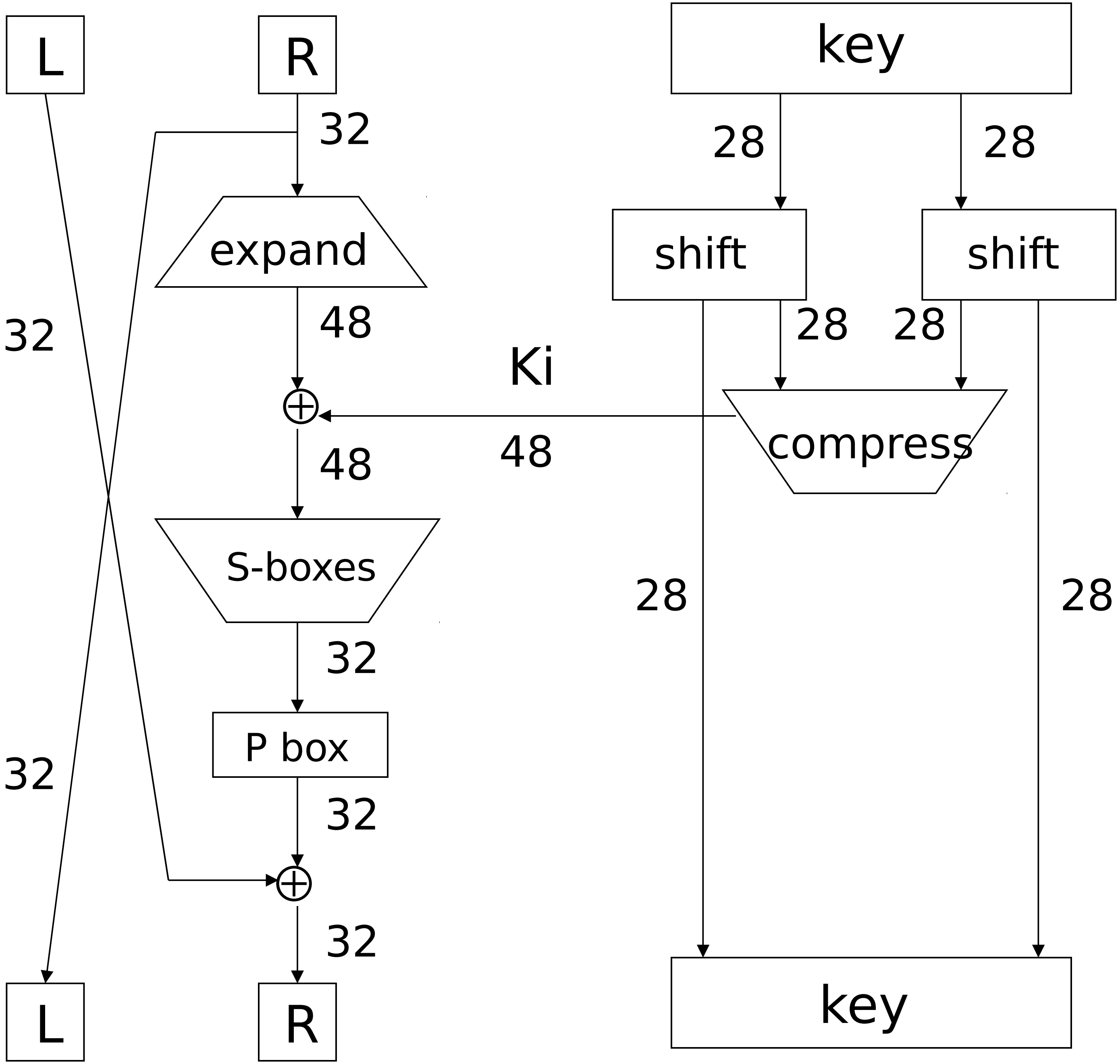
  where F is round function and K_i is subkey
- Plaintext: $P = (L_0, R_0)$
- Formula "works" for any function F
  - But only secure for certain functions F

Data Encryption Standard

- **DES** developed in 1970’s
- Based on IBM’s Lucifer cipher
- DES was U.S. government standard
- DES development was controversial
  - NSA secretly involved
  - Design process was secret
  - Key length reduced from 128 to 56 bits
  - Subtle changes to Lucifer algorithm

DES Numerology

- DES is a Feistel cipher with…
  - 64 bit block length
  - 56 bit key length
  - 16 rounds
  - 48 bits of key used each round (subkey)
- Each round is simple (for a block cipher)
- Security depends heavily on “S-boxes”
  - Each S-boxes maps 6 bits to 4 bits



One Round of DES

DES Expansion Permutation

- Input 32 bits

  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
  16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- Output 48 bits

  31 0 1 2 3 4 3 4 5 6 7 8
  7 8 9 10 11 12 11 12 13 14 15 16
  15 16 17 18 19 20 19 20 21 22 23 24
  23 24 25 26 27 28 27 28 29 30 31 0

DES S-box

- 8 “substitution boxes” or S-boxes
- Each S-box maps 6 bits to 4 bits
- S-box number 1

input bits (0,5)

↓ input bits (1,2,3,4)

```
   | 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
-------------------------------------------------------------------------------------
00 | 1110 0100 1101 0001 0010 1111 1011 1000 0011 1010 0110 1100 0101 1001 0000 0111
01 | 0000 1111 0111 0100 1110 0010 1101 0001 1010 0110 1100 1011 1001 0101 0011 1000
10 | 0100 0001 1110 1000 1101 0110 0010 1011 1111 1100 1001 0111 0011 1010 0101 0000
11 | 1111 1100 1000 0010 0100 1001 0001 0111 0101 1011 0011 1110 1010 0000 0110 1101
```

DES P-box

- Input 32 bits

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
```

- Output 32 bits

```
 15   6  19  20  28  11  27  16   0  14  22  25   4  17  30   9
  1   7  23  13  31  26   2   8  18  12  29   5  21  10   3  24
```

DES Subkey

- 56 bit DES key, numbered 0,1,2,…,55
- Left half key bits, LK

```
49 42 35 28 21 14  7
 0 50 43 36 29 22 15
 8  1 51 44 37 30 23
16  9  2 52 45 38 31
```

- Right half key bits, RK

```
55 48 41 34 27 20 13
 6 54 47 40 33 26 19
12  5 53 46 39 32 25
18 11  4 24 17 10  3
```

DES Subkey

- For rounds `i=1,2,...,16`
  - Let LK = (LK circular shift left by ri)
  - Let RK = (RK circular shift left by ri)
  - Left half of subkey Ki is of LK bits

```
13 16 10 23  0  4  2 27 14  5 20  9
22 18 11  3 25  7 15  6 26 19 12  1
```

  - Right half of subkey Ki is RK bits

```
12 23  2  8 18 26  1 11 22 16  4 19
15 20 10 27  5 24 17 13 21  7  0  3
```

DES Subkey

- For rounds 1, 2, 9 and 16 the shift ri is 1, and in all other rounds ri is 2
- Bits 8,17,21,24 of LK omitted each round
- Bits 6,9,14,25 of RK omitted each round
- **Compression permutation** yields 48 bit subkey Ki from 56 bits of LK and RK
- **Key schedule** generates subkey

DES Last Word (Almost)

- An initial permutation before round 1
- Halves are swapped after last round
- A final permutation (inverse of initial perm) applied to (R16,L16)
- None of this serves security purpose

Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- Thirty+ years of intense analysis has revealed no “back door”
- Attacks, essentially exhaustive key search
- **Inescapable conclusions**
  - Designers of DES knew what they were doing
  - Designers of DES were way ahead of their time

Block Cipher Notation

- P = plaintext block
- C = ciphertext block
- Encrypt P with key K to get ciphertext C
  - $C = E(P, K)$
- Decrypt C with key K to get plaintext P
  - $P = D(C, K)$
- Note: $P = D(E(P, K), K)$ and $C = E(D(C, K), K)$
  - But $P \neq D(E(P, K1), K2)$ and $C \neq E(D(C, K1), K2)$ when $K1 \neq K2$

Triple DES

- Today, 56 bit DES key is too small
  - Exhaustive key search is feasible
- But DES is everywhere, so what to do?
- **Triple DES** or **3DES** (112 bit key)
  - C = E(D(E(P,K1),K2),K1)
  - P = D(E(D(C,K1),K2),K1)
- Why Encrypt-Decrypt-Encrypt with 2 keys?
  - Backward compatible: E(D(E(P,K),K),K) = E(P,K)
  - And 112 bits is enough

3DES

- Why not C = E(E(P,K),K) ?
  - Trick question --- it’s still just 56 bit key
- Why not C = E(E(P,K1),K2) ?
- A (semi-practical) **known plaintext** attack
  - Pre-compute table of E(P,K1) for every possible key K1 (resulting table has 256 entries)
  - Then for each possible K2 compute D(C,K2) until a match in table is found
  - When match is found, have E(P,K1) = D(C,K2)
  - Result gives us keys: C = E(E(P,K1),K2)

Advanced Encryption Standard

- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected (pronounced like "Rain Doll" or "Rhine Doll")
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)

AES Overview

- **Block size:** 128 bits (others in Rijndael)
- **Key length:** 128, 192 or 256 bits (independent of block size)
- 10 to 14 rounds (depends on key length)
- Each round uses 4 functions (3 “layers”)
  - ByteSub (nonlinear layer)
  - ShiftRow (linear mixing layer)
  - MixColumn (nonlinear layer)
  - AddRoundKey (key addition layer)

AES ByteSub

- Treat 128 bit block as 4x6 byte array

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ByteSub} \longrightarrow \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}.$$

- ByteSub is AES's "S-box"
- Can be viewed as nonlinear (but invertible) composition of two math operations

AES “S-box”

Last 4 bits of input

First 4 bits of input

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

AES ShiftRow

- Cyclic shift rows

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ShiftRow} \longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

AES MixColumn

- Invertible, linear operation applied to each column

$$\begin{bmatrix} a_{0i} \\ a_{1i} \\ a_{2i} \\ a_{3i} \end{bmatrix} \longrightarrow \texttt{MixColumn} \longrightarrow \begin{bmatrix} b_{0i} \\ b_{1i} \\ b_{2i} \\ b_{3i} \end{bmatrix} \quad \text{for } i = 0, 1, 2, 3$$

- Implemented as a (big) lookup table

AES AddRoundKey

- XOR subkey with block

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Block Subkey

- RoundKey (subkey) determined by **key schedule** algorithm

AES Decryption

- To decrypt, process must be invertible
- Inverse of MixAddRoundKey is easy, since "$\oplus$" is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)

A Few Other Block Ciphers

- Briefly…
  - IDEA
  - Blowfish
  - RC6
- More detailed…
  - TEA

IDEA

- Invented by James Massey
  - One of the giants of modern crypto
- IDEA has 64-bit block, 128-bit key
- IDEA uses **mixed-mode arithmetic**
- Combine different math operations
  - IDEA the first to use this approach
  - Frequently used today

Blowfish

- Blowfish encrypts 64-bit blocks
- Key is variable length, up to 448 bits
- Invented by Bruce Schneier
- Almost a Feistel cipher

  $R_i = L_{i-1} \oplus K_i$

  $L_i = R_{i-1} \oplus F(L_{i-1} \oplus K_i)$
- The round function F uses 4 S-boxes
  - Each S-box maps 8 bits to 32 bits
- **Key-dependent S-boxes**
  - S-boxes determined by the key

RC6

- Invented by Ron Rivest
- Variables
  - Block size
  - Key size
  - Number of rounds
- An AES finalist
- Uses **data dependent rotations**
  - Unusual for algorithm to depend on plaintext

Time for TEA

- Tiny Encryption Algorithm (TEA)
- 64 bit block, 128 bit key
- Assumes 32-bit arithmetic
- Number of rounds is variable (32 is considered secure)
- Uses “weak” round function, so large number of rounds required

TEA Encryption

Assuming 32 rounds:

```
(K[0],K[1],K[2],K[3]) = 128 bit key

(L,R) = plaintext (64-bit block)

delta = 0x9e3779b9

sum = 0

for i = 1 to 32

    sum += delta

    L += ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])

    R += ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])

next i

ciphertext = (L,R)
```

TEA Decryption

Assuming 32 rounds:

```
(K[0],K[1],K[2],K[3]) = 128 bit key
(L,R) = ciphertext (64-bit block)
delta = 0x9e3779b9
sum = delta << 5
for i = 1 to 32
    R −= ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
    L −= ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
    sum −= delta
next i
plaintext = (L,R)
```

TEA Comments

- **Almost** a Feistel cipher
  - Uses + and - instead of $\oplus$ (XOR)
- Simple, easy to implement, fast, low memory requirement, etc.
- Possibly a “related key” attack
- eXtended TEA (XTEA) eliminates related key attack (slightly more complex)
- Simplified TEA (STEA) — insecure version used as an example for cryptanalysis

Block Cipher Modes

Multiple Blocks

- How to encrypt multiple blocks?
- Do we need a new key for each block?
  - As bad as (or worse than) a one-time pad!
- Encrypt each block independently?
- Make encryption depend on previous block?
  - That is, can we “chain” the blocks together?
- How to handle partial blocks?
  - We won’t discuss this issue

Modes of Operation

- Many modes — we discuss 3 most popular
- Electronic Codebook (**ECB**) mode
  - Encrypt each block independently
  - Most obvious, but has a serious weakness
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
  - More secure than ECB, virtually no extra work
- Counter Mode (**CTR**) mode
  - Block ciphers acts like a stream cipher
  - Popular for random access

ECB Mode

- Notation: C = E(P,K)
- Given plaintext P0,P1,…,Pm,…
- Most obvious way to use a block cipher:

Encrypt	**Decrypt**
C0 = E(P0, K)	P0 = D(C0, K)
C1 = E(P1, K)	P1 = D(C1, K)
C2 = E(P2, K) …	P2 = D(C2, K) …

- For fixed key K, this is “electronic” version of a codebook cipher (without additive)
  - With a different codebook for each key

ECB Cut and Paste

- Suppose plaintext is

  `Alice digs Bob. Trudy digs Tom.`
- Assuming 64-bit blocks and 8-bit ASCII:

  P0 = “`Alice di`”, P1 = “`gs Bob. `”,

  P2 = “`Trudy di`”, P3 = “`gs Tom. `”
- Ciphertext: C0,C1,C2,C3
- Trudy cuts and pastes: C0,C3,C2,C1
- Decrypts as

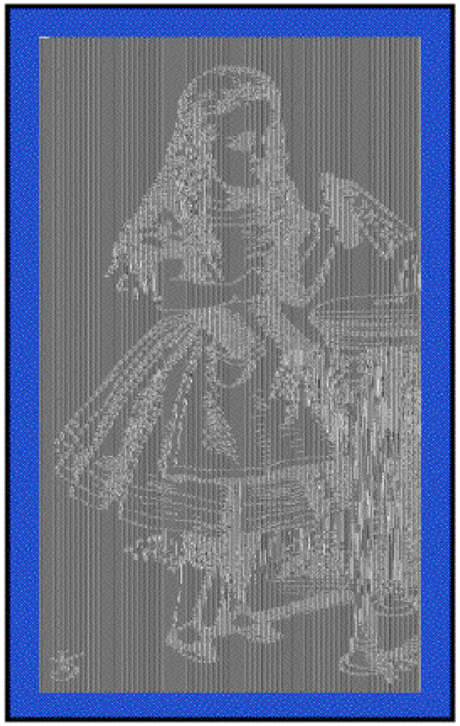
  `Alice digs Tom. Trudy digs Bob.`

ECB Weakness

- Suppose Pi = Pj
- Then Ci = Cj and Trudy knows Pi = Pj
- This gives Trudy some information, even if she does not know Pi or Pj
- Trudy might know Pi
- Is this a serious issue?

Alice Hates ECB Mode

- Alice’s uncompressed image, and ECB encrypted (TEA)



- Why does this happen?
- Same plaintext yields same ciphertext!

CBC Mode

- Blocks are "chained" together
- A random initialization vector, or IV, is required to initialize CBC mode
- IV is random, but not secret

Encryption	**Decryption**
$C0 = E(IV \oplus P0, K)$,	$P0 = IV \oplus D(C0, K)$,
$C1 = E(C0 \oplus P1, K)$,	$P1 = C0 \oplus D(C1, K)$,
$C2 = E(C1 \oplus P2, K)$,…	$P2 = C1 \oplus D(C2, K)$,…

- Analogous to classic codebook *with additive*

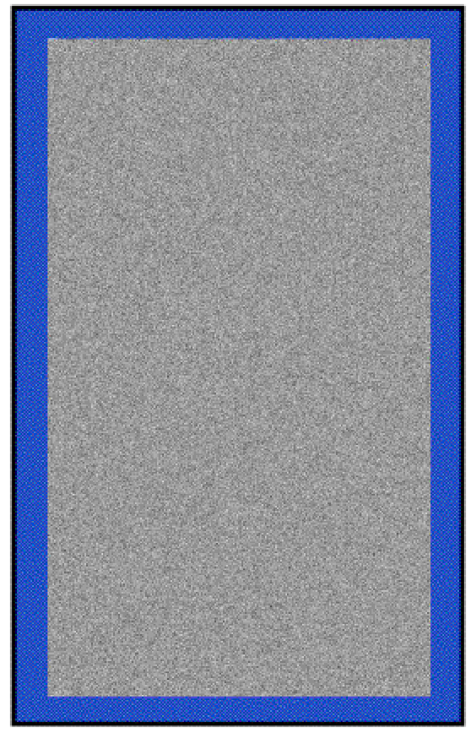
CBC Mode

- Identical plaintext blocks yield different ciphertext blocks — this is good!
- If C1 is garbled to, say, G then

  $P1 \neq C0 \oplus D(G, K), P2 \neq G \oplus D(C2, K)$
- But $P3 = C2 \oplus D(C3, K), P4 = C3 \oplus D(C4, K),\ldots$
- Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

Alice Likes CBC Mode

- Alice's uncompressed image, Alice CBC encrypted (TEA)



- Why does this happen?
- Same plaintext yields different ciphertext!

Counter Mode (CTR)

- CTR is popular for random access
- Use block cipher like a stream cipher

  Encryption Decryption

  $C0 = P0 \oplus E(IV, K)$, $P0 = C0 \oplus E(IV, K)$,

  $C1 = P1 \oplus E(IV+1, K)$, $P1 = C1 \oplus E(IV+1, K)$,

  $C2 = P2 \oplus E(IV+2, K)$,… $P2 = C2 \oplus E(IV+2, K)$,…
- CBC can also be used for random access
  - With a significant limitation…

Integrity

Data Integrity

- **Integrity** — detect unauthorized writing (i.e., modification of data)
- Example: Inter-bank fund transfers
  - Confidentiality may be nice, integrity is critical
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc.

MAC

- Message Authentication Code (MAC)
  - Used for data **integrity**
  - Integrity **not** the same as confidentiality
- MAC is computed as **CBC residue**
  - That is, compute CBC encryption, saving only final ciphertext block, the MAC

MAC Computation

- MAC computation (assuming N blocks)

  $C0 = E(IV \oplus P0, K)$,

  $C1 = E(C0 \oplus P1, K)$,

  $C2 = E(C1 \oplus P2, K),\ldots$

  $CN-1 = E(CN-2 \oplus PN-1, K) = MAC$
- MAC sent with IV and plaintext
- Receiver does same computation and verifies that result agrees with MAC
- Note: receiver must know the key K

Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  C0 = E(IV⊕P0,K), **C1** = E(**C0**⊕P1,K),

  C2 = E(**C1**⊕P2,K), **C3** = E(**C2**⊕P3,K) = **MAC**
- Alice sends IV,P0,P1,P2,P3 and **MAC** to Bob
- Suppose Trudy changes P1 to X
- Bob computes

  C0 = E(IV⊕P0,K), ***C1*** = E(**C0**⊕X,K),

  C2 = E(***C1***⊕P2,K), ***C3*** = E(***C2***⊕P3,K) = ***MAC*** ≠ **MAC**
- That is, error propagates into **MAC**
- Trudy can't make ***MAC*** == **MAC** without K

Confidentiality and Integrity

- Encrypt with one key, MAC with another key
- Why not use the same key?
  - Send last encrypted block (MAC) twice?
  - This cannot add any security!
- Using different keys to encrypt and compute MAC works, even if keys are related
  - But, twice as much work as encryption alone
  - Can do a little better — about 1.5 “encryptions”
- Confidentiality and integrity with same work as one encryption is a research topic

Uses for Symmetric Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later…)
- Anything you can do with a hash function (upcoming chapter…)