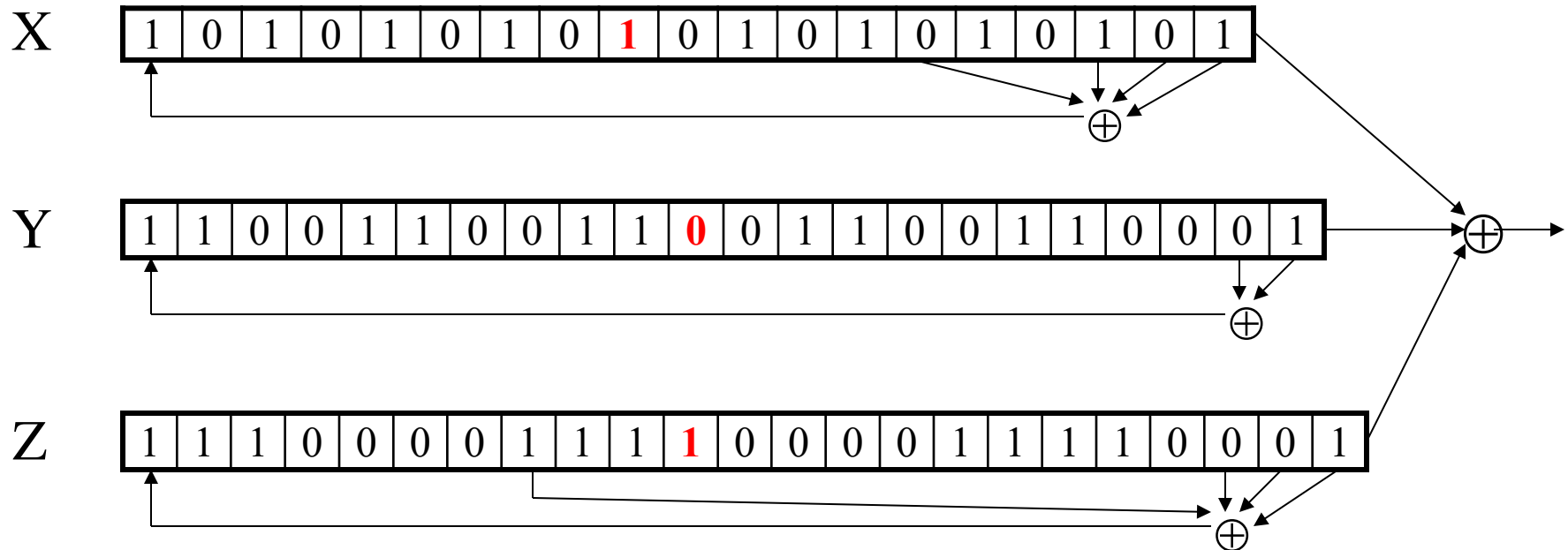


Symmetric Key Crypto

- ❑ Stream cipher — generalize 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 — generalized codebook
 - Block cipher key determines a codebook
 - Each key yields a different codebook
 - Employs both “confusion” and “diffusion”

A5/1



- ❑ In this example, $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(\mathbf{1}, \mathbf{0}, \mathbf{1}) = \mathbf{1}$
- ❑ Register X steps, Y does not step, and Z steps
- ❑ Keystream bit is XOR of right bits of registers
- ❑ Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

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

- At each step, swap elements in table and select keystream byte

$i = (i + 1) \bmod 256$

$j = (j + S[i]) \bmod 256$

swap($S[i]$, $S[j]$)

$t = (S[i] + S[j]) \bmod 256$

keystreamByte = $S[t]$

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

Block Ciphers



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 $i = 1, 2, \dots, n$, 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**

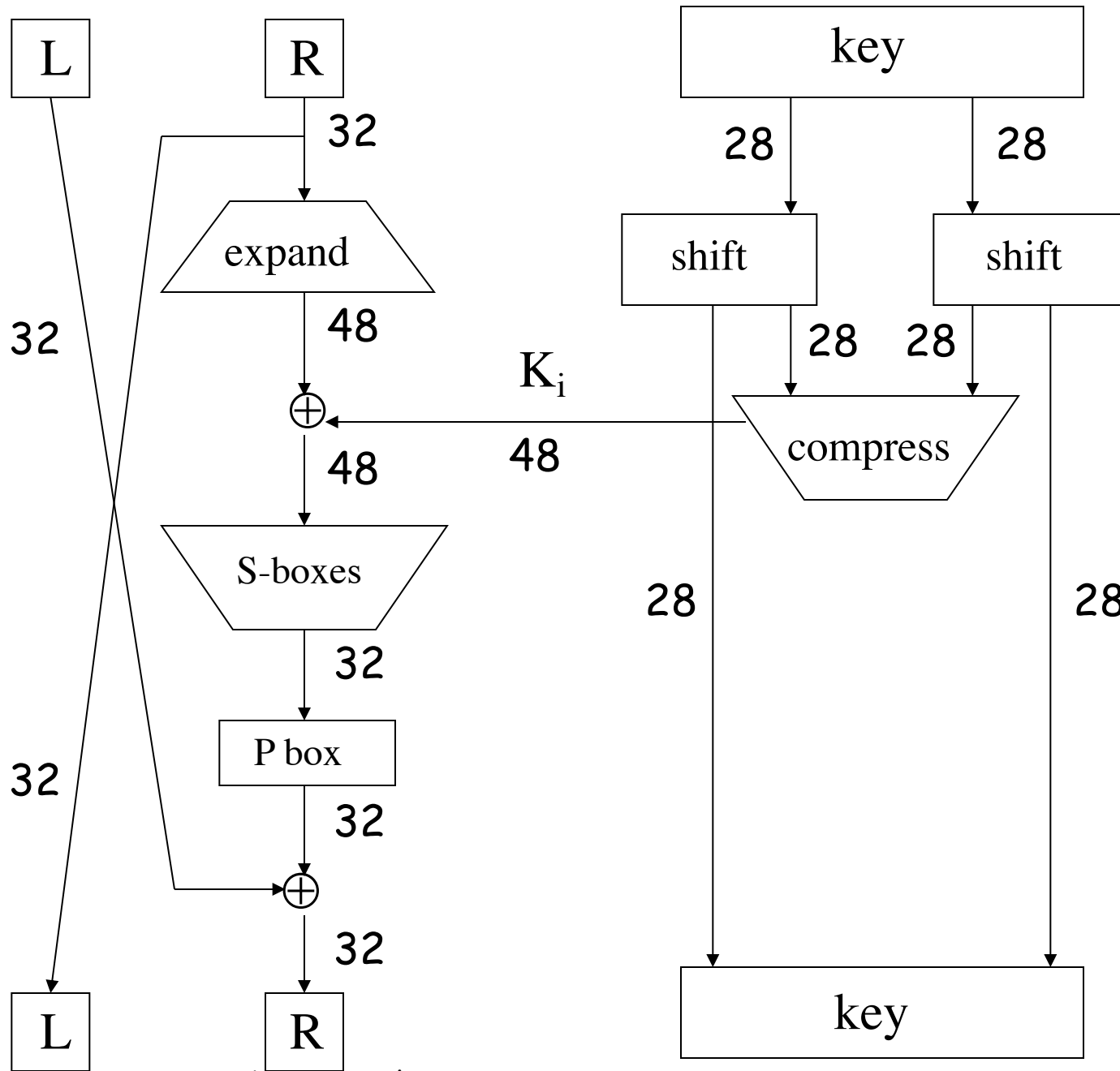
- ❑ **Ciphertext**: $C = (L_n, R_n)$

Feistel Cipher: Decryption

- ❑ Start with ciphertext $C = (L_n, R_n)$
- ❑ For each round $i = n, n-1, \dots, 1$, 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: $P = (L_0, R_0)$
- ❑ Decryption 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
- ❑ Development of DES was controversial
 - NSA secretly involved
 - Design process was secret
 - Key length reduced from 128 to 56 bits
 - Subtle changes to Lucifer algorithm



One
Round
of
DES

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 $P \neq D(E(P, K_1), K_2)$ and $C \neq E(D(C, K_1), K_2)$ when $K_1 \neq K_2$

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)
 - $C = E(D(E(P, K_1), K_2), K_1)$
 - $P = D(E(D(C, K_1), K_2), K_1)$
- ❑ Why Encrypt-Decrypt-Encrypt with 2 keys?
 - Backward compatible: $E(D(E(P, K), K), K) = E(P, K)$
 - And 112 is a lot of bits

3DES

- ❑ Why not $C = E(E(P, K), K)$ instead?
 - Trick question — still just 56 bit key
- ❑ Why not $C = E(E(P, K_1), K_2)$ instead?
- ❑ A (semi-practical) **known plaintext** attack
 - Pre-compute table of $E(P, K_1)$ for every possible key K_1 (resulting table has 2^{56} entries)
 - Then for each possible K_2 compute $D(C, K_2)$ until a match in table is found
 - When match is found, have $E(P, K_1) = D(C, K_2)$
 - Result gives us keys: $C = E(E(P, K_1), K_2)$

Advanced Encryption Standard

- ❑ Replacement for DES
- ❑ AES competition (late 90's)
 - NSA openly involved
 - Transparent selection 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: Executive Summary

- ❑ **Block size:** 128 bits (others in Rijndael)
- ❑ **Key length:** 128, 192 or 256 bits
(independent of block size in Rijndael)
- ❑ 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 4x4 byte array

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

- 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

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

AES MixColumn

- Invertible, applied to each column

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

- 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:

$(K[0], K[1], K[2], K[3]) = 128 \text{ bit key}$

$(L, R) = \text{plaintext (64-bit block)}$

$\text{delta} = 0x9e3779b9$

$\text{sum} = 0$

for $i = 1$ to 32

$\text{sum} += \text{delta}$

$L += ((R \ll 4) + K[0]) \wedge (R + \text{sum}) \wedge ((R \gg 5) + K[1])$

$R += ((L \ll 4) + K[2]) \wedge (L + \text{sum}) \wedge ((L \gg 5) + K[3])$

next i

ciphertext = (L, R)

TEA Decryption

Assuming 32 rounds:

$(K[0], K[1], K[2], K[3]) = 128 \text{ bit key}$

$(L, R) = \text{ciphertext (64-bit block)}$

$\text{delta} = 0x9e3779b9$

$\text{sum} = \text{delta} \ll 5$

for $i = 1$ to 32

$R \text{ -= } ((L \ll 4) + K[2]) \wedge (L + \text{sum}) \wedge ((L \gg 5) + K[3])$

$L \text{ -= } ((R \ll 4) + K[0]) \wedge (R + \text{sum}) \wedge ((R \gg 5) + K[1])$

$\text{sum} \text{ -= delta}$

next i

$\text{plaintext} = (L, R)$

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 (STEAs) — 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?
 - If so, as impractical as a one-time pad!
- ❑ Encrypt each block independently?
- ❑ Is there any analog of codebook “additive”?
- ❑ 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 approach, but a bad idea
- ❑ 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 $P_0, P_1, \dots, P_m, \dots$
- Most obvious way to use a block cipher:

Encrypt

$$C_0 = E(P_0, K)$$

$$C_1 = E(P_1, K)$$

$$C_2 = E(P_2, K) \dots$$

Decrypt

$$P_0 = D(C_0, K)$$

$$P_1 = D(C_1, K)$$

$$P_2 = D(C_2, K) \dots$$

- 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:

P_0 = “Alice di”, P_1 = “gs Bob. ”,

P_2 = “Trudy di”, P_3 = “gs Tom. ”

- Ciphertext: C_0, C_1, C_2, C_3

- Trudy cuts and pastes: C_0, C_3, C_2, C_1

- Decrypts as

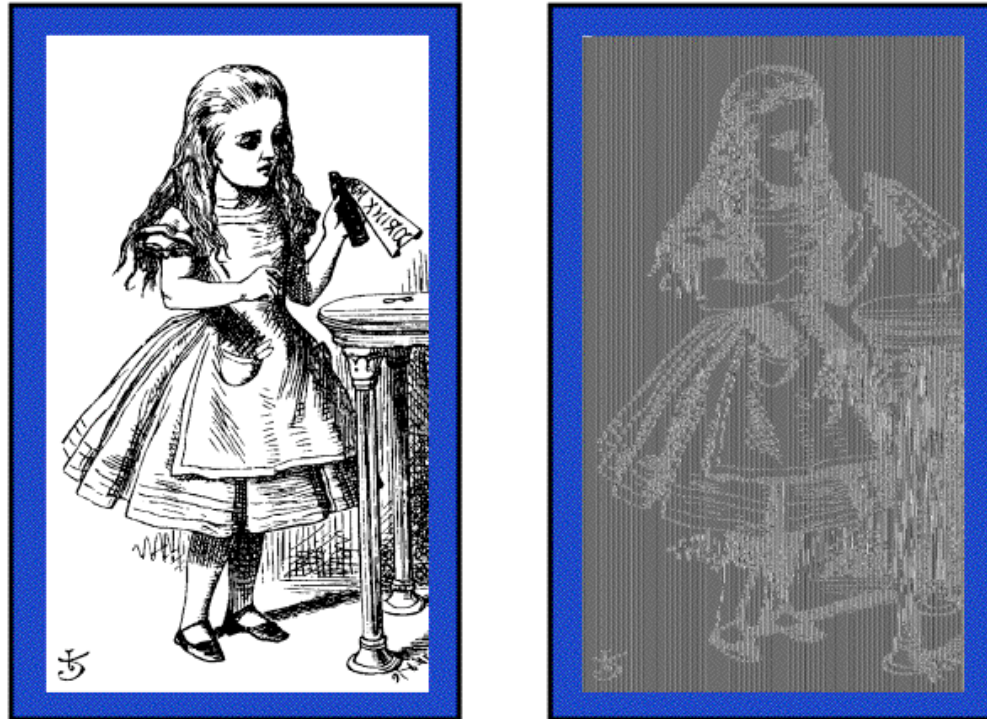
Alice digs Tom. Trudy digs Bob.

ECB Weakness

- ❑ Suppose $P_i = P_j$
- ❑ Then $C_i = C_j$ and Trudy knows $P_i = P_j$
- ❑ This gives Trudy some information, even if she does not know P_i or P_j
- ❑ Trudy might know P_i
- ❑ 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

$$\begin{aligned}C_0 &= E(\text{IV} \oplus P_0, K), \\C_1 &= E(C_0 \oplus P_1, K), \\C_2 &= E(C_1 \oplus P_2, K), \dots\end{aligned}$$

Decryption

$$\begin{aligned}P_0 &= \text{IV} \oplus D(C_0, K), \\P_1 &= C_0 \oplus D(C_1, K), \\P_2 &= C_1 \oplus D(C_2, K), \dots\end{aligned}$$

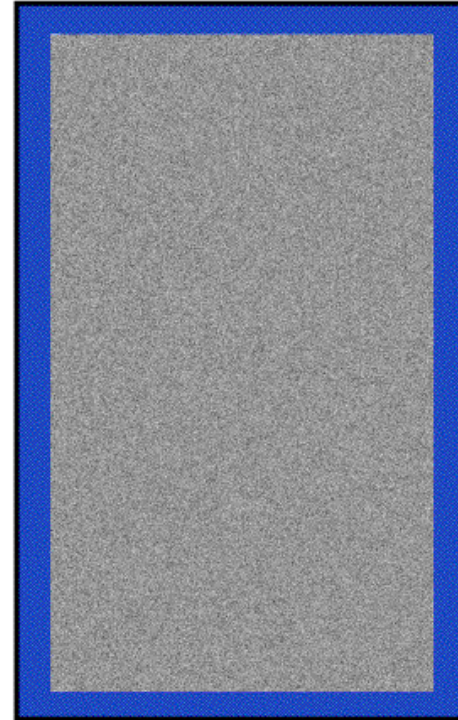
- ❑ Analogous to classic codebook *with additive*

CBC Mode

- ❑ Identical plaintext blocks yield different ciphertext blocks — this is very good!
- ❑ But what about errors in transmission?
 - If C_1 is garbled to, say, G then
$$P_1 \neq C_0 \oplus D(G, K), P_2 \neq G \oplus D(C_2, K)$$
 - But $P_3 = C_2 \oplus D(C_3, K), P_4 = C_3 \oplus D(C_4, K), \dots$
 - 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

$$C_0 = P_0 \oplus E(\text{IV}, K),$$

$$C_1 = P_1 \oplus E(\text{IV}+1, K),$$

$$C_2 = P_2 \oplus E(\text{IV}+2, K), \dots$$

Decryption

$$P_0 = C_0 \oplus E(\text{IV}, K),$$

$$P_1 = C_1 \oplus E(\text{IV}+1, K),$$

$$P_2 = C_2 \oplus E(\text{IV}+2, K), \dots$$

Integrity

Data Integrity

- ❑ **Integrity** — detect unauthorized writing (i.e., detect unauthorized mod 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., 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
 - The MAC serves as a cryptographic checksum for data

MAC Computation

- ❑ MAC computation (assuming N blocks)

$$C_0 = E(IV \oplus P_0, K),$$

$$C_1 = E(C_0 \oplus P_1, K),$$

$$C_2 = E(C_1 \oplus P_2, K), \dots$$

$$C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = \text{MAC}$$

- ❑ Send $IV, P_0, P_1, \dots, P_{N-1}$ and MAC

- ❑ Receiver does same computation and verifies that result agrees with MAC

- ❑ Both sender and receiver must know K

Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes
$$C_0 = E(IV \oplus P_0, K), C_1 = E(C_0 \oplus P_1, K),$$
$$C_2 = E(C_1 \oplus P_2, K), C_3 = E(C_2 \oplus P_3, K) = \text{MAC}$$
- Alice sends IV, P_0, P_1, P_2, P_3 and **MAC** to Bob
- Suppose Trudy changes P_1 to X
- Bob computes
$$C_0 = E(IV \oplus P_0, K), C_1 = E(C_0 \oplus X, K),$$
$$C_2 = E(C_1 \oplus P_2, K), C_3 = E(C_2 \oplus P_3, K) = \text{MAC} \neq \text{MAC}$$
- It works since 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
- ❑ 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)

❑ Anything you can do with a hash function (upcoming chapter...)