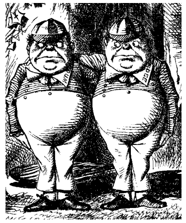
Block Ciphers II

Three fellows throughout the course

Good

Alice

Bob

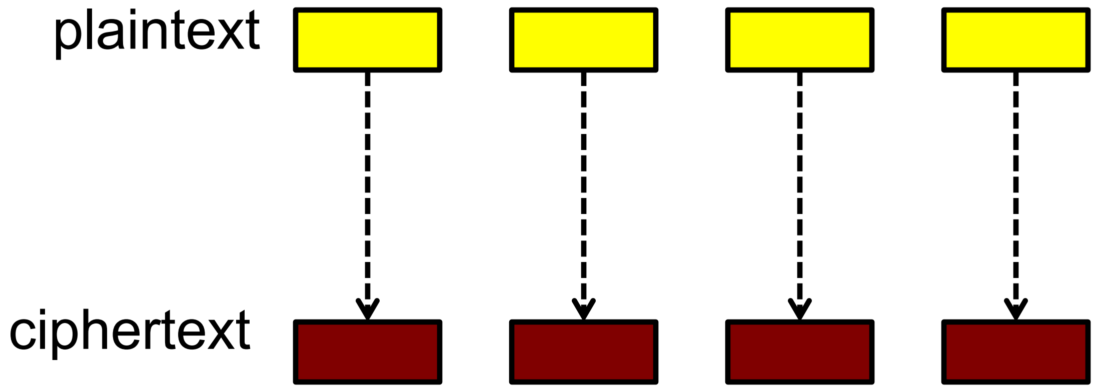


Trudy

Bad

Refresher: Block Cipher

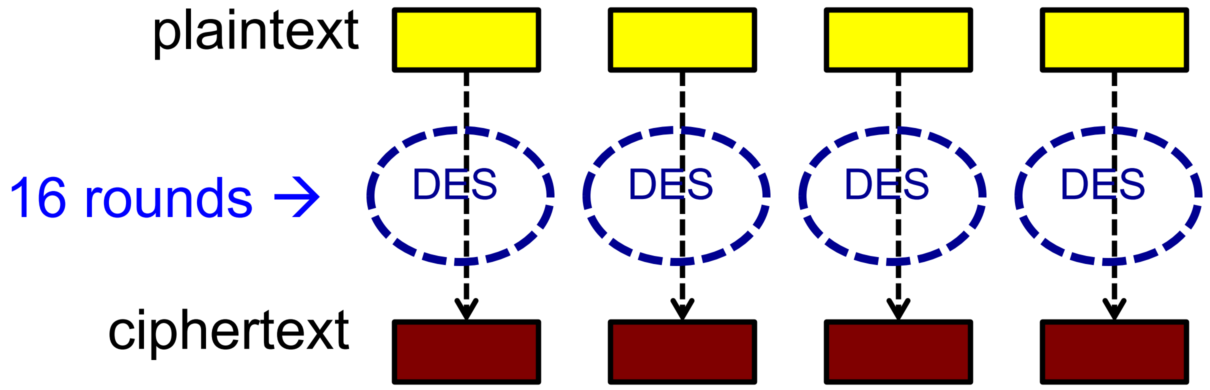
- Plaintext and ciphertext consist of fixed-sized blocks



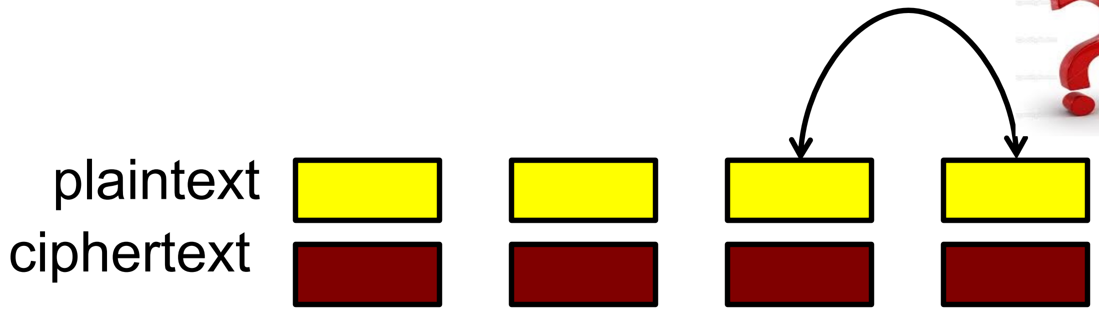
Where is DES (if used) in this picture?

Where is DES in the big picture?

- Plaintext and ciphertext consist of fixed-sized blocks



Multiple Blocks



- How to encrypt multiple blocks?
- Do we need a new key for each block?
- Encrypt each block independently?
- Make encryption depend on previous block?

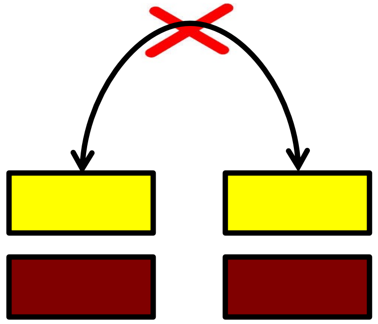
Modes of Operation

- 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

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



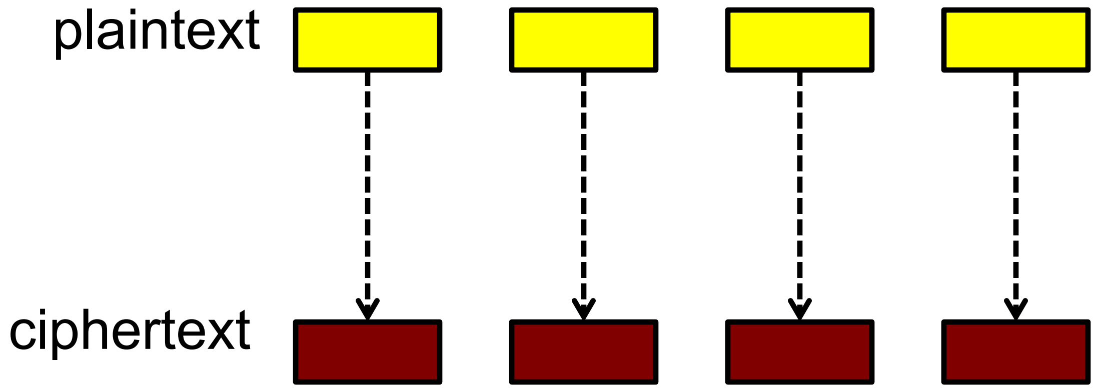
- Notation: $C = E(P, K)$
- Given plaintext $P_0, P_1, \ldots, P_m, \ldots$
- Most obvious way to use a block cipher:

Encrypt	**Decrypt**
$C_0 = E(P_0, K)$	$P_0 = D(C_0, K)$
$C_1 = E(P_1, K)$	$P_1 = D(C_1, K)$
$C_2 = E(P_2, K) \quad \ldots$	$P_2 = D(C_2, K) \quad \ldots$

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

ECB Mode

- Good: both encryption and decryption can be done in parallel



Question: Anything bad about it?

Think about this…

- Suppose that the plaintext is as follows:

Money☐for☐Alice☐is☐$1000

Money☐for☐Trudy☐is☐$2☐☐☐

- ☐ is space
- Each character is 8 bits
- block size is 64bits

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Plaintext blocks

- P_0 = Money□fo
- P_1 = r□Alice□
- P_2 = is□$1000
- P_3 = Money□fo
- P_4 = r□Trudy□
- P_5 = is□$2□□□

Ciphertext blocks

- P_0 = Money□fo
- P_1 = r□Alice□
- P_2 = is□$1000
- P_3 = Money□fo
- P_4 = r□Trudy□
- P_5 = is□$2□□□

$$P_0, P_1, P_2, P_3, P_4, P_5 \xrightarrow{C_i = E(P_i, K)} C_0, C_1, C_2, C_3, C_4, C_5 \xrightarrow{P_i = D(C_i, K)} P_0, P_1, P_2, P_3, P_4, P_5$$

Cut & Paste attack by Trudy

- P_0 = Money□fo
- P_1 = r□Alice□
- P_2 = is□$1000
- P_3 = Money□fo
- P_4 = r□Trudy□
- P_5 = is□$2□□□

$C_i = E(P_i, K)$

$P_i = D(C_i, K)$

$$P_0, P_1, P_2, P_3, P_4, P_5 \longrightarrow C_0, C_1, C_2, C_3, C_4, C_5$$

$$C_0, C_1, C_5, C_3, C_4, C_2 \longrightarrow P_0, P_1, P_5, P_3, P_4, P_2$$

Decrypted message

Money□for□Alice□is□$2□□□

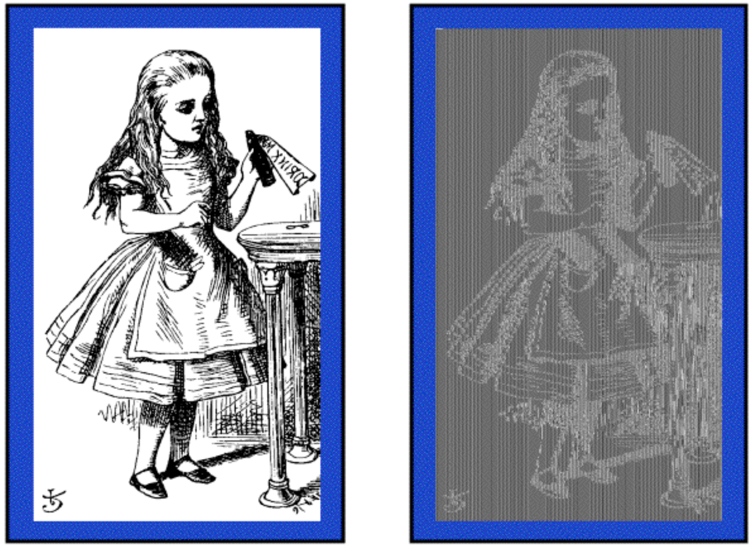
Money□for□Trudy□is□**$1000**

Another ECB Weakness

- Suppose ECB is used
- If $C_i = C_j$ then Trudy knows $P_i = P_j$
- This gives Trudy some information, even if she does not know P_i or P_j
  - Patterns present in the plaintext can remain visible in the ciphertext
- Trudy might know P_i
  - Then P_j is revealed as well

Alice Hates ECB Mode

- Alice's uncompressed image, and ECB encrypted



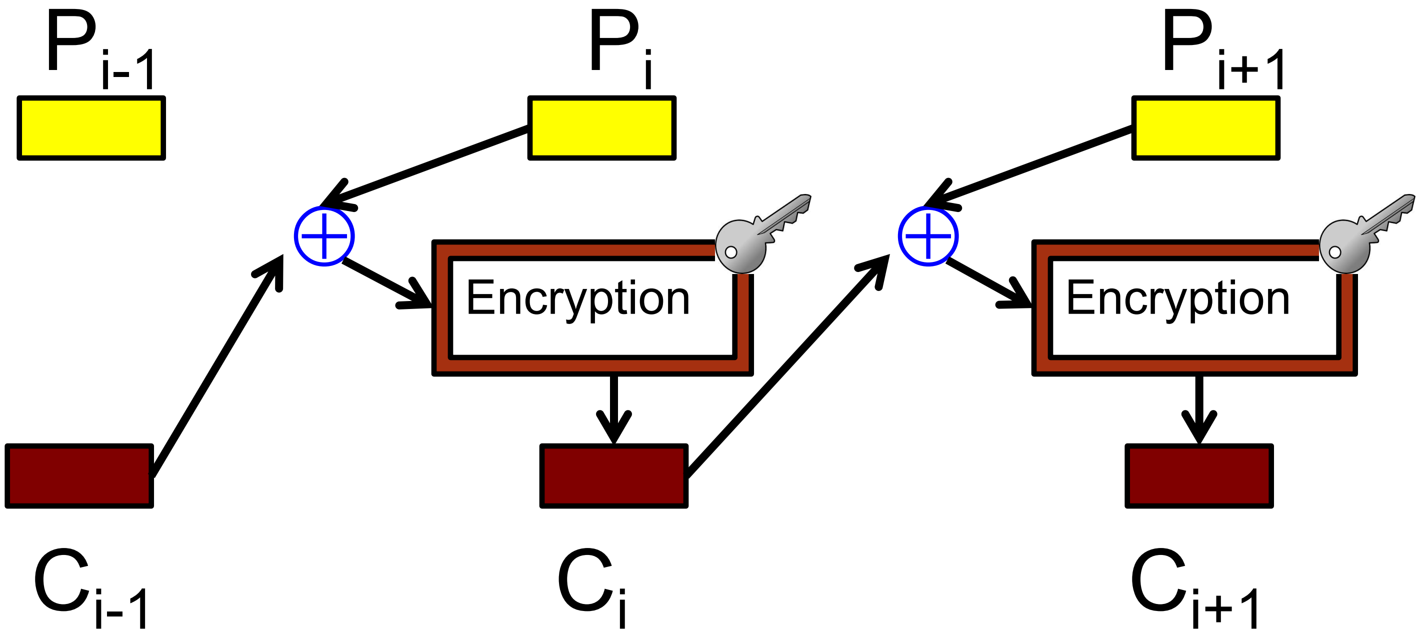
- ❑ Why does this happen?
- ❑ Same plaintext always yields same ciphertext!

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

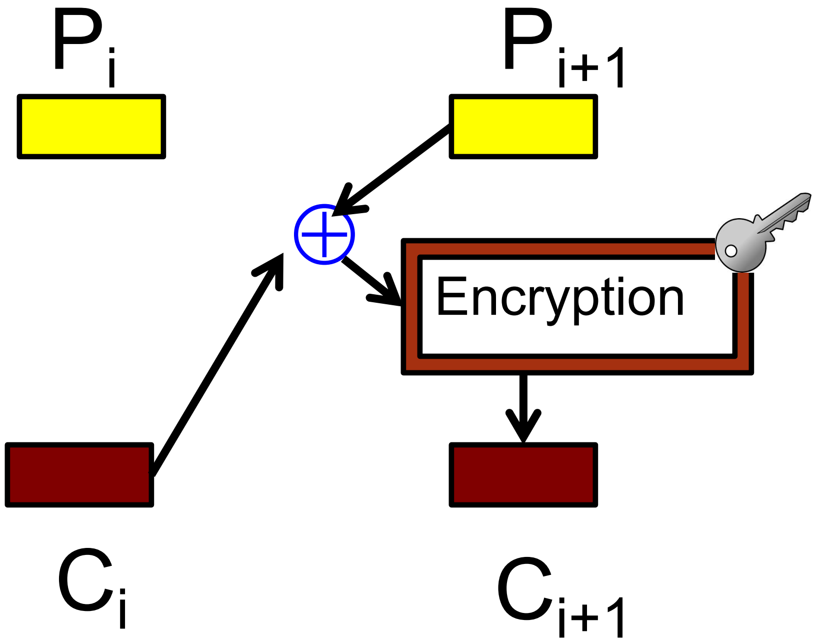
Cipher Block Chaining (CBC) Mode -- (Encryption)

- Blocks are “chained” together: $C_{i+1} = E(C_i \oplus P_{i+1}, K)$



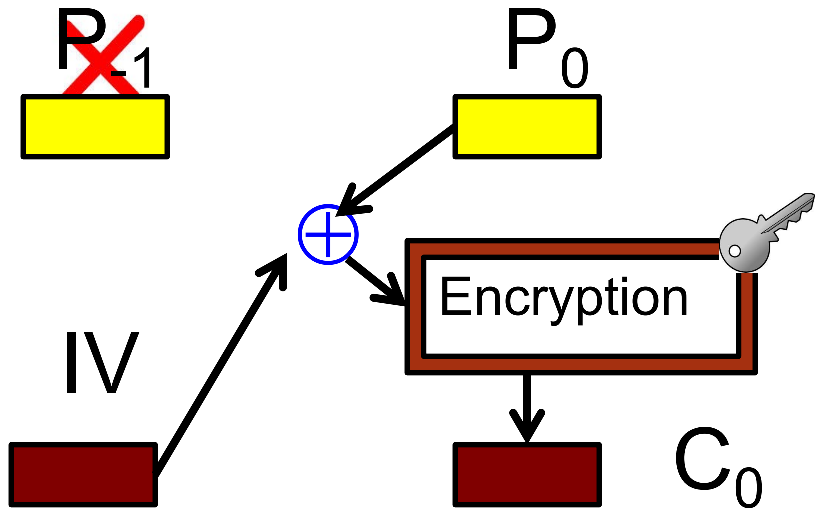
CBC - Decryption

- Given C_{i+1}, C_i
- How to get P_{i+1}?
- $P_{i+1} = D(C_{i+1},K) \oplus C_i$



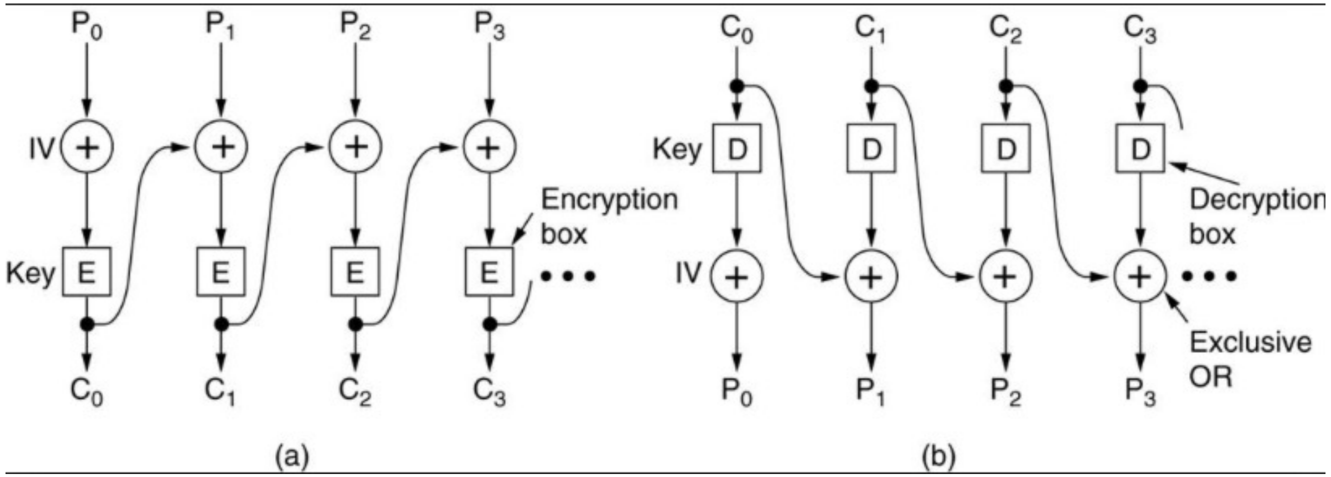
Initialization vector

Question: how does the receiver knows IV for decryption?



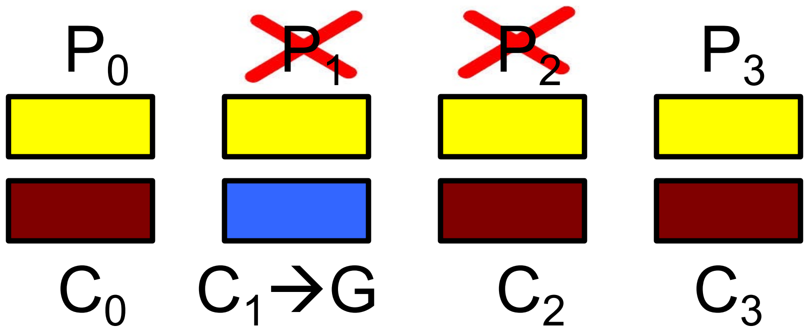
- A random initialization vector (IV) is used to initialize CBC
- IV is random, but not secret
- Analogous to classic codebook *with additive*

A better picture of CBC



CBC with garbled data

Garbled data means
Data changed due to
transmission error
E.g, bit 0 becomes bit 1



$$C_{i+1} = E(C_i \oplus P_{i+1}, K)$$

Here, suppose C_1 is garbled to G, which plaintext block is lost?

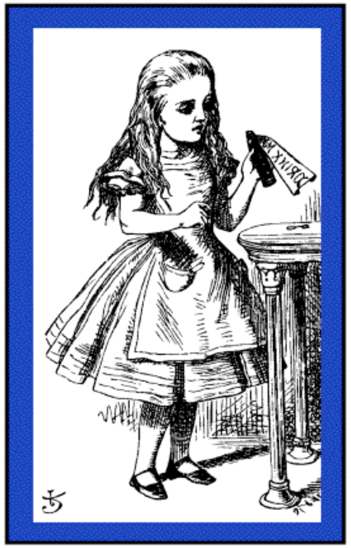
- P_1, P_2

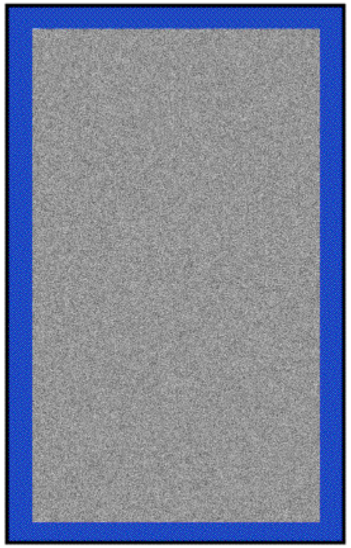
$$G = E(C_0 \oplus P_1, K)$$

$$C_2 = E(G \oplus P_2, K)$$

Alice Likes CBC Mode

- Alice's uncompressed image, Alice CBC encrypted





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

BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK

Any attack against CBC?

- Consider the encrypted message
  IV, C1, C2, C3, C4, C5

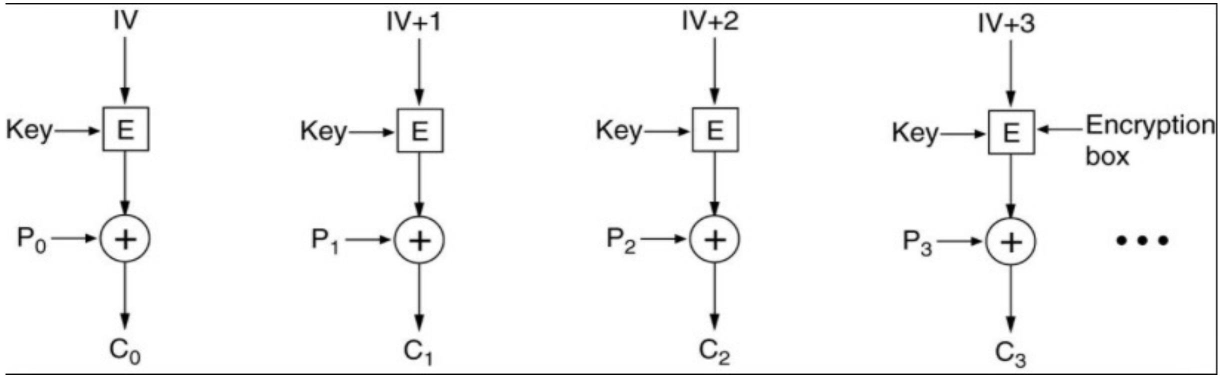
- If receiver receives IV, C1, C2, C3, C4, is it valid?
- If receiver receives C2, C3, C4, C5, is it valid?
- If receiver receives C2, C3, C4, is it valid?
- Any subset of a CBC message will decrypt cleanly.

- If we snip out blocks, leaving IV, C1, C4, C5, we only corrupt one block of plaintext. Which one?

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

Counter Mode (CTR)



Encryption

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

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

$C_2 = P_2 \oplus E(IV+2, K), \ldots$

Decryption

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

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

$P_2 = C_2 \oplus E(IV+2, K), \ldots$

Counter Mode (CTR)

- CTR is popular for random access
- Highly parallelizable; no linkage between stages
- Use block cipher like a stream cipher

Advanced Encryption Standard (AES)

Advanced Encryption Standard - History

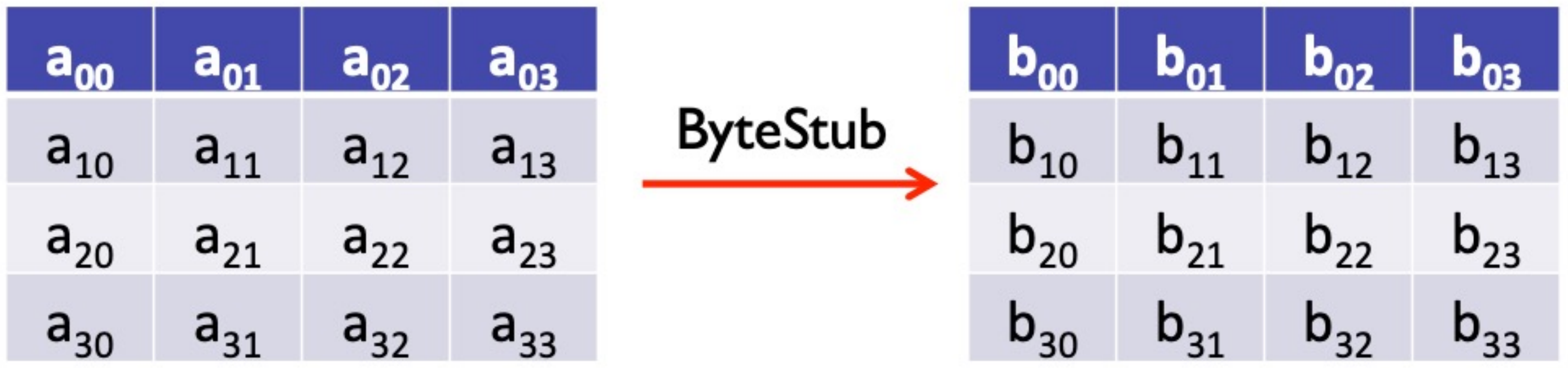
- Replacement for DES, 56bits key too small
- AES competition (late 90’s) brought by NBS again…
  - NSA openly involved as a judge
  - Transparent selection process
  - Many strong algorithms proposed, unlike 20 years ago for DES
  - Rijndael Algorithm ultimately selected (pronounced like “Rain Doll”)
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)

AES: Executive Summary

- **Block size:** 128 bits
- **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 4x4 byte array



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}

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}

- 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

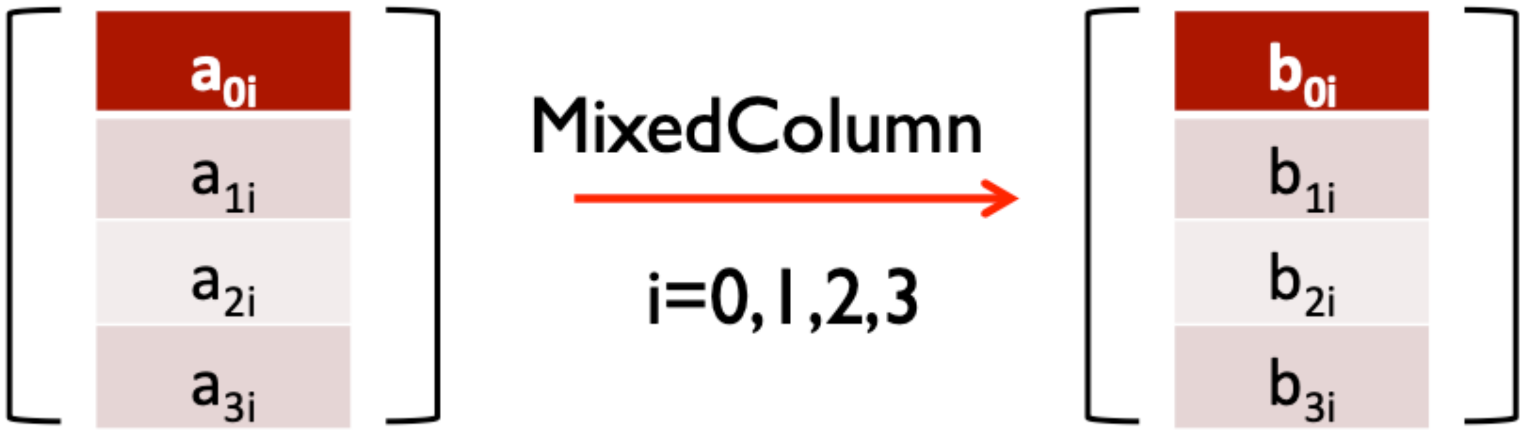
AES ShiftRow

- Circular shift rows

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{array}{c} \text{L-Shift by 0} \\ \xrightarrow{\text{ShiftRow}} \\ \text{L-Shift by 2} \\ \text{L-Shift by 3} \end{array} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

AES MixColumn

- Invertible, linear operation applied to each column



- Implemented as a (big) lookup table

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

AES AddRoundKey

- XOR subkey with block

$$\underbrace{\begin{array}{|c|c|c|c|}\hline a_{00} & a_{01} & a_{02} & a_{03} \\ \hline a_{10} & a_{11} & a_{12} & a_{13} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} \\ \hline a_{30} & a_{31} & a_{32} & a_{33} \\ \hline \end{array}}_{\text{Block}} \oplus \underbrace{\begin{array}{|c|c|c|c|}\hline k_{00} & k_{01} & k_{02} & k_{03} \\ \hline k_{10} & k_{11} & k_{12} & k_{13} \\ \hline k_{20} & k_{21} & k_{22} & k_{23} \\ \hline k_{30} & k_{31} & k_{32} & k_{33} \\ \hline \end{array}}_{\text{Subkey}} = \begin{array}{|c|c|c|c|}\hline a_{00} & a_{01} & a_{02} & a_{03} \\ \hline a_{10} & a_{11} & a_{12} & a_{13} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} \\ \hline a_{30} & a_{31} & a_{32} & a_{33} \\ \hline \end{array}$$

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

AES Decryption

- To decrypt, process must be invertible
- Inverse of MixAddRoundKey is easy, since “⊕” 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)

Quiz (Check all answers that apply)

What is the strength of RC4?

- **A: Its permutation table changes every step**
- B: Its key length has to be 256 bytes
- C: It can be implemented efficiently with hardware
- D: It was designed by a famous cryptographer

Quiz (Check all answers that apply)

Which of the following are correct?

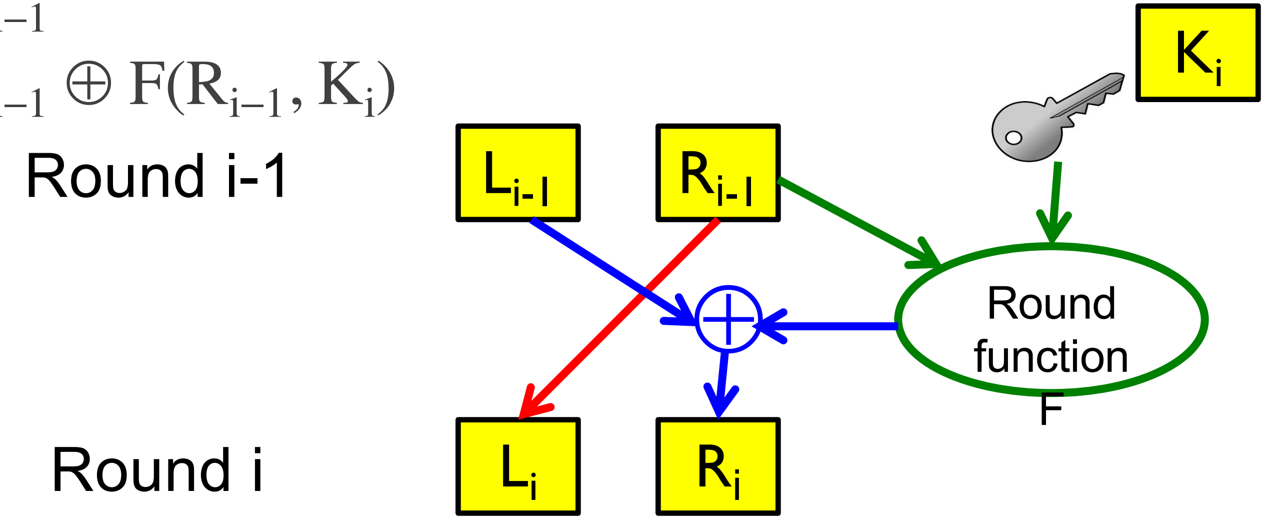
- A: **A5/1 is good for hardware implementation**
- B: **A5/1 was developed for GSM**
- **C: A5/1 produces a bit every step**
- **D: A5/1 uses shift registers**

Quiz: how to do decryption?

- For each round $i = 1, 2, \ldots, n$, compute

$L_i = R_{i-1}$

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



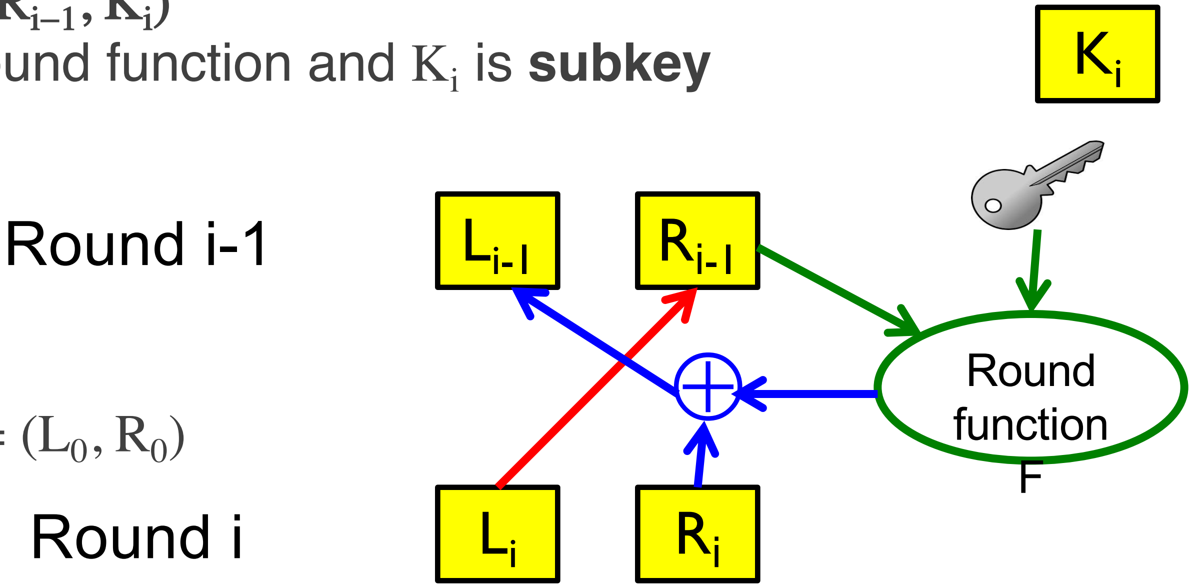
What is $\mathbf{L_{i-1}}$ and $\mathbf{R_{i-1}}$ given $\mathbf{L_i}$, $\mathbf{R_i}$, and $\mathbf{K_i}$?

Feistel Cipher: Decryption

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

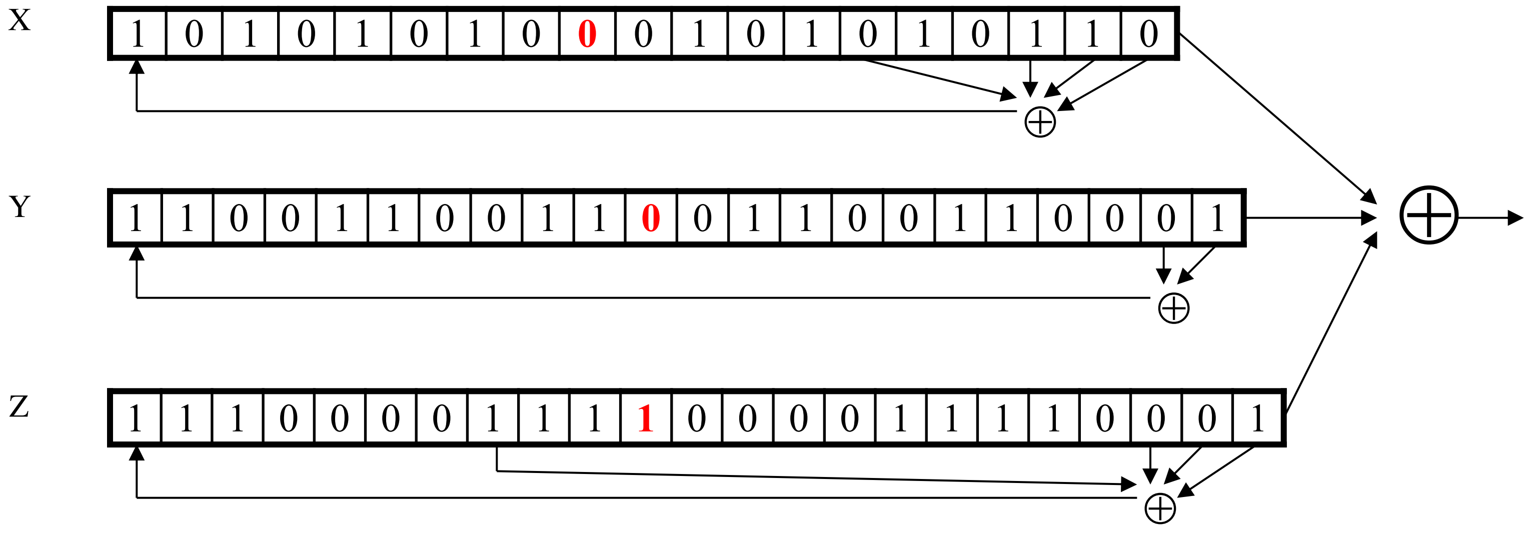
  $R_{i-1} = L_i$
  $L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$
  where F is round function and K_i is **subkey**

- Plaintext: $P = (L_0, R_0)$



BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Quiz : What's the output bit from A5/1?



- A: (X: 0) ⊕ (Y: 1) ⊕(Z: 1) = 0
- **B: (X: 1) ⊕ (Y: 0) ⊕(Z: 1) = 0**
- C: (X: 1) ⊕ (Y: 1) ⊕(Z: 1) = 1
- D: (X: 1) ⊕ (Y: 0) ⊕(Z: 0) = 1

Quiz: Double encryption

- Double encryption involves encrypting plaintext first with key K1 and then encrypting the result again with a different key K2. Which of the following attacks is particularly effective against such a double encryption scheme?
- A:Brute-force attack
- B:Man-in-the-middle attack
- C:Meet-in-the-middle attack
- D:Side-channel attack

Programming Homework

- Implement stream cipher Salsa20
- Read the Salsa20 specification
- Don’t copy code from internet and classmates

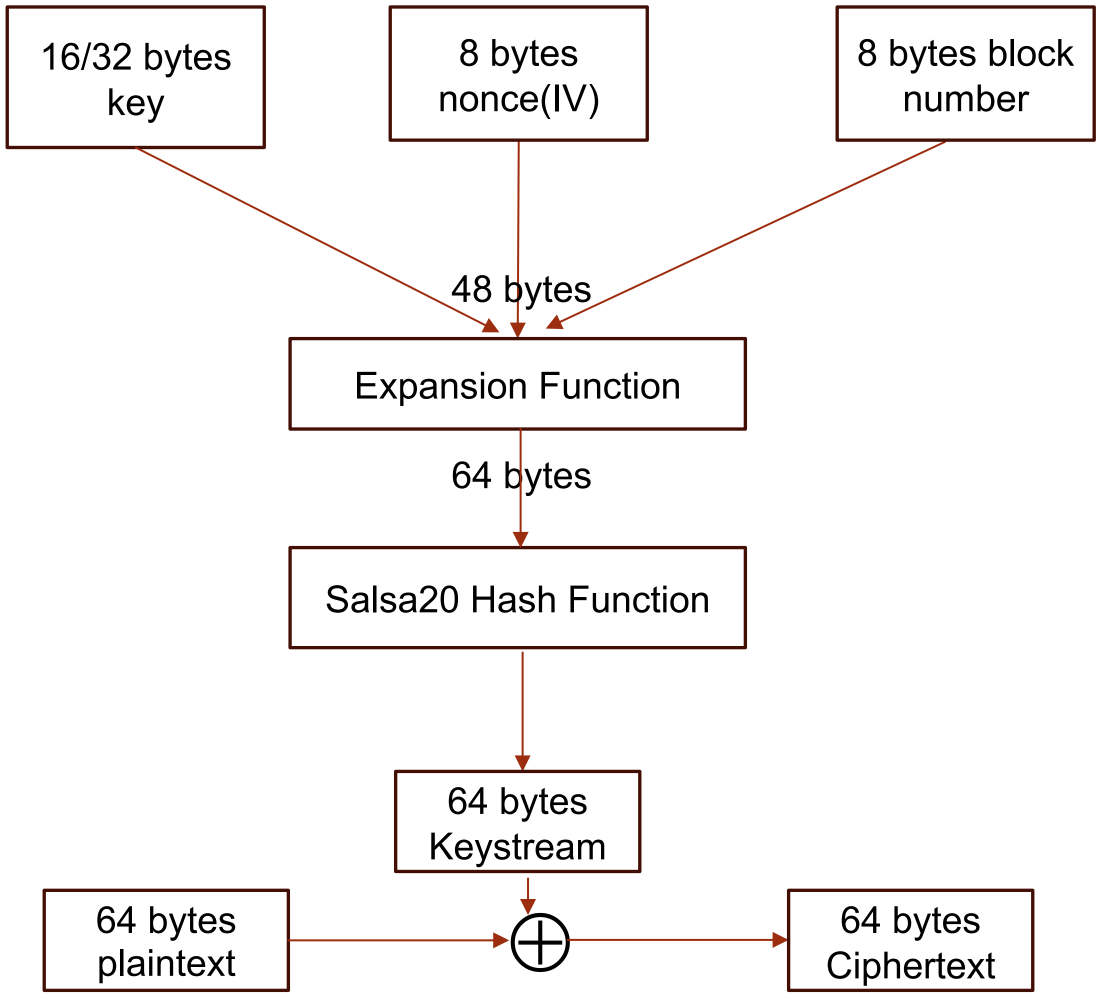
Salsa20 - History

- Stream cipher is dead?
- Designed by Daniel Bernstein in 2005
- Simple, high performance, and strong security
- Candidate of eSTREAM project
  - part of the European Network of Excellence in Cryptology (ECRYPT) initiative to identify new stream ciphers suitable for widespread adoption
- Chacha - Sucessor of Salsa20
  - Integrated into security protocols

Salsa20

- Input: 16/32 bytes(128bit/256bit) key, 8-byte nonce(IV), 8-byte block number
- Output: 64 bytes keystream
- Encryption: plaintext $\oplus$ keystream = ciphertext(64 bytes)

Salsa20 Encryption

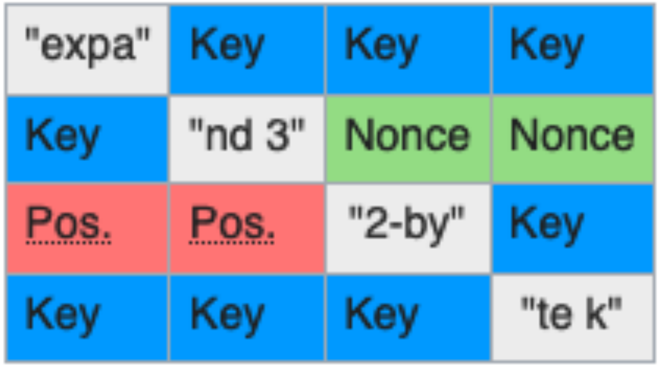


Salsa20 Encryption – Expansion

32 bytes key

4 X 4 matrix

Initial state of Salsa20



"expa"	Key	Key	Key
Key	"nd 3"	Nonce	Nonce
Pos.	Pos.	"2-by"	Key
Key	Key	Key	"te k"

Each word is **4 bytes**

Eight words of key
Two words of nonce (IV)
Two words of stream position(block number)
Four fixed words

Constant word "expand 32-byte k” in ASCII

Salsa20 Encryption – Hash function

- **Little Endian Function**
  - Starting from `x = (x[0], x[1], . . . , x[63])`, define
  - `x0 = littleendian(x[0], x[1], x[2], x[3]),`
  - `x1 = littleendian(x[4], x[5], x[6], x[7]),`
  - `x2 = littleendian(x[8], x[9], x[10], x[11]),`
  - …
  - `x15 = littleendian(x[60], x[61], x[62], x[63])`
- If b = (b0, b1, b2, b3) then
  - **littleendian**(b) = b0 + 2^{8}b1 + 2^{16}b2 + 2^{24}b3.

Salsa20 Encryption – Hash function

- **Double Round Function**
  - Input: 4 X 4 matrix(16-word sequence)
  - Output: 4 X 4 matrix(16-word sequence)
  - A double round is a column round followed by a row round
  - `doubleround(x) = rowround(columnround(x))`
  - $doubleround(x)^{10}$ `iterate doubleround for 10 times`
    - **Iterate: using previous round result as input for next round**
    - **10 times double round -> 20 rounds**
      - **Also called Salsa20/20**
      - **Other popular variants: Salsa20/8, Salsa20/12**
- **Salsa20(x) = x +** $doubleround(x)^{10}$

Addition is mod 2^32

Salsa20 Encryption – Hash function

- **Columnround Function**
  - Input: 4 X 4 matrix(16-word sequence) – (x0, x1, .. x15)
  - Output: 4 X 4 matrix(16-word sequence) - (y0, y1 .. y15)
  - If `x = (x0, x1, x2, x3, . . . , x15)` then **`columnround(x) = (y0, y1, y2, y3, . . . , y15)`**
  - where

$$(y_0, y_4, y_8, y_{12}) = \mathrm{quarterround}(x_0, x_4, x_8, x_{12}),$$

Not in order $$\boxed{(y_5, y_9, y_{13}, y_1)} = \mathrm{quarterround}\boxed{(x_5, x_9, x_{13}, x_1)},$$

$$(y_{10}, y_{14}, y_2, y_6) = \mathrm{quarterround}(x_{10}, x_{14}, x_2, x_6),$$

$$(y_{15}, y_3, y_7, y_{11}) = \mathrm{quarterround}(x_{15}, x_3, x_7, x_{11}).$$

Y0	**Y1**	**Y2**	**Y3**
Y4	Y5	Y6	Y7
Y8	Y9	Y10	Y11
Y12	Y13	Y14	y15

Output matrix

Salsa20 Encryption – Hash function

- **Rowround Function**
  - Input: 16-word sequence(4X4 matrix) – y0, y1 … y15
  - Output: 16-word sequence(4X4 matrix) - z0, z1 … z15
  - If `y = (y0, y1, y2, y3, . . . , y15)` then **`rowround(y) = (z0, z1, z2, z3, . . . , z15)`** where

$$(z_0, z_1, z_2, z_3) = \text{quarterround}(y_0, y_1, y_2, y_3),$$

Not in order

$$\boxed{(z_5, z_6, z_7, z_4)} = \text{quarterround}\boxed{(y_5, y_6, y_7, y_4)},$$

$$(z_{10}, z_{11}, z_8, z_9) = \text{quarterround}(y_{10}, y_{11}, y_8, y_9),$$

$$(z_{15}, z_{12}, z_{13}, z_{14}) = \text{quarterround}(y_{15}, y_{12}, y_{13}, y_{14}).$$

z0	**z1**	**z2**	**z3**
z4	z5	z6	z7
z8	z9	z10	z11
z12	z13	z14	y15

Output matrix

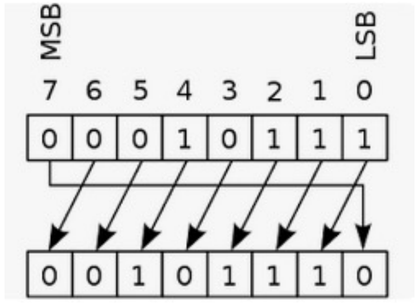
Salsa20 Encryption – Hash function

- **Quaterround Function(core)**
  - Input: a 4-word(16 bytes) sequence
  - Output: a 4-word(16 bytes) sequence
  - If `y = (y0, y1, y2, y3)` then **`quarterround(y) = (z0, z1, z2, z3)`** where

Addition is mod 2^32

$$\boxed{\begin{aligned} z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7), \\ z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9), \\ z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13), \\ z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18). \end{aligned}}$$

<<< left circular shift(left rotation)



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
  - Trade number of round for complexity of each round

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])
end for
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
end for
plaintext = (L,R)
```

TEA Comments

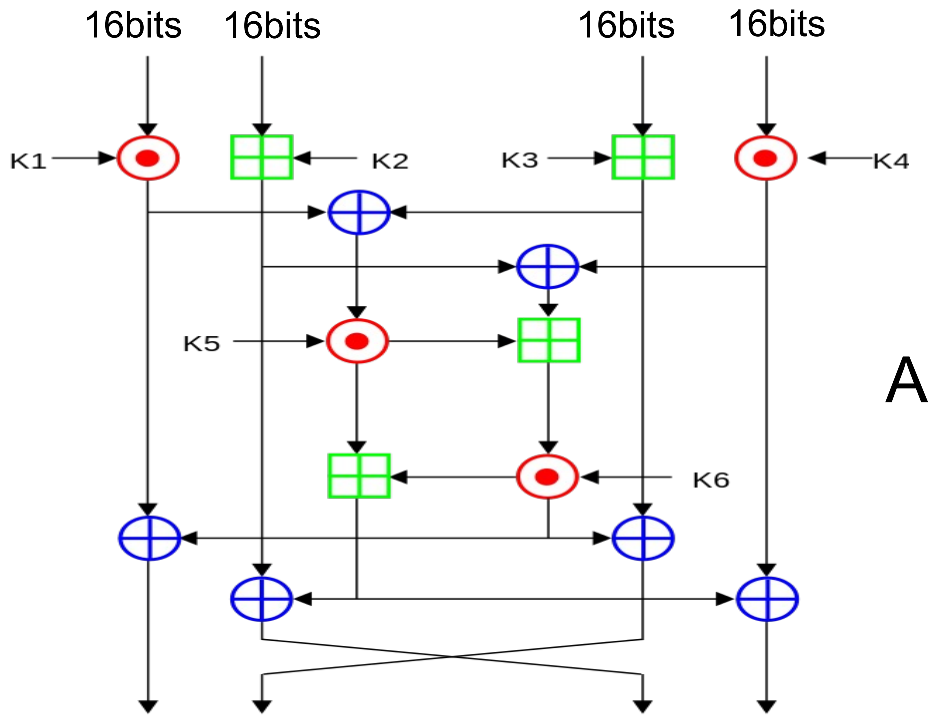
- **"Almost"** a Feistel cipher
  - Uses + and – (mod 2^{32})
- Simple, easy to implement, fast, low memory requirement, etc
- eXtended TEA (XTEA) eliminates related key attack (slightly more complex)
- Simplified TEA (STEA) — insecure version used as an example for cryptanalysis

A Few Other Block Ciphers

- Briefly…
  - IDEA
  - Blowfish
  - RC6

IDEA (International Data Encryption Algorithm)

- Invented by James Massey
- IDEA has 64-bit block, 128-bit key
- IDEA has 8.5 rounds
- IDEA uses **mixed-mode arithmetic** to get rid of the S-box explicitly
  - XOR, addition, multiplication
- Key schedule
  - Shifts and transformation
- Combine different math operations
  - IDEA the first to use this approach
  - Frequently used today



A full round

⊕: bitwise XOR

⊞: addition modulo 2^{16}

⊙: Multiplication modulo $2^{16}+1$, where the all-zero word (0x0000) in inputs is interpreted as 2^{16} and 2^{16} in output is interpreted as the all-zero word (0x0000)

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Blowfish

- Blowfish encrypts 64-bit blocks
- Key is variable length, up to 448 bits
- **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: 128bits
  - Key size: 128, 192, or 256 bit
  - Number of rounds: 20
- An AES finalist
- Uses **data dependent rotations**
  - Unusual for algorithm to depend on plaintext

Data Integrity with Symmetric Key Crypto

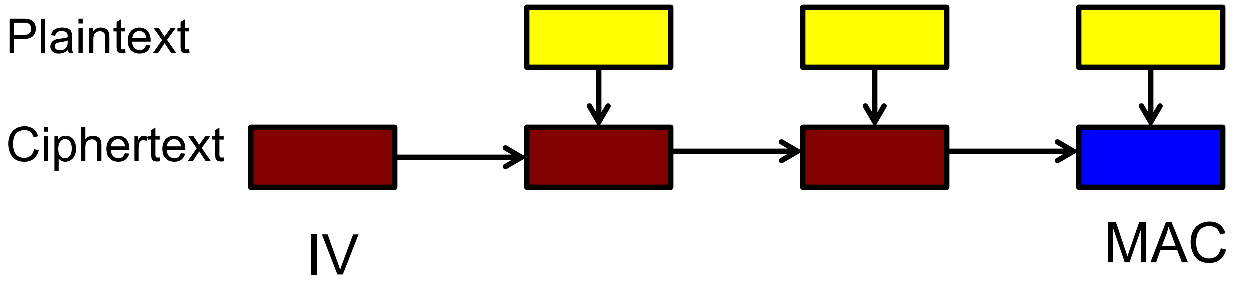
Data Integrity

- **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)
- Can encryption alone provide integrity?
    - ECB cut-and-paste attack

Answer is no!

MAC for integrity

- 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)

$$\left.\begin{aligned}
&C_0 = E(IV \oplus P_0, K),\\
&C_1 = E(C_0 \oplus P_1, K),\\
&C_2 = E(C_1 \oplus P_2, K),\ldots\\
&C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = \textbf{MAC}
\end{aligned}\right\}\text{CBC}$$

- MAC sent with IV and plaintext
- Receiver does same computation and verifies that result agrees with MAC
- Note: receiver must know the key K

How does MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $\mathbf{C_0} = E(IV \oplus P_0, K)$, $\mathbf{C_1} = E(\mathbf{C_0} \oplus P_1, K)$,

  $\mathbf{C_2} = E(\mathbf{C_1} \oplus P_2, K)$, $\mathbf{C_3} = E(\mathbf{C_2} \oplus P_3, K) = \mathbf{MAC}$
- Alice sends IV, P_0, P_1, P_2, P_3 and $\mathbf{MAC}$ to Bob
- Suppose Trudy changes P_1 to X
- Bob computes

  $\mathbf{C_0} = E(IV \oplus P_0, K)$, $\boldsymbol{C_1} = E(\mathbf{C_0} \oplus X, K)$,

  $\boldsymbol{C_2} = E(\boldsymbol{C_1} \oplus P_2, K)$, $\boldsymbol{C_3} = E(\boldsymbol{C_2} \oplus P_3, K) = \boldsymbol{MAC} \neq \mathbf{MAC}$
- That is, error propagates into MAC

Wait a minute,

- We know that CBC (Cipher Block Chaining) achieves automatic error recovery. But now we say that the error propagates to the last ciphertext block. What's the difference?

Wait a minute,

- We know that CBC (Cipher Block Chaining) achieves automatic error recovery. But now we say that the error propagates to the last ciphertext block. What's the difference?

The former is done through **decryption**;

- Encryption finished; Error is from garbled data
- Remaining data is not affected

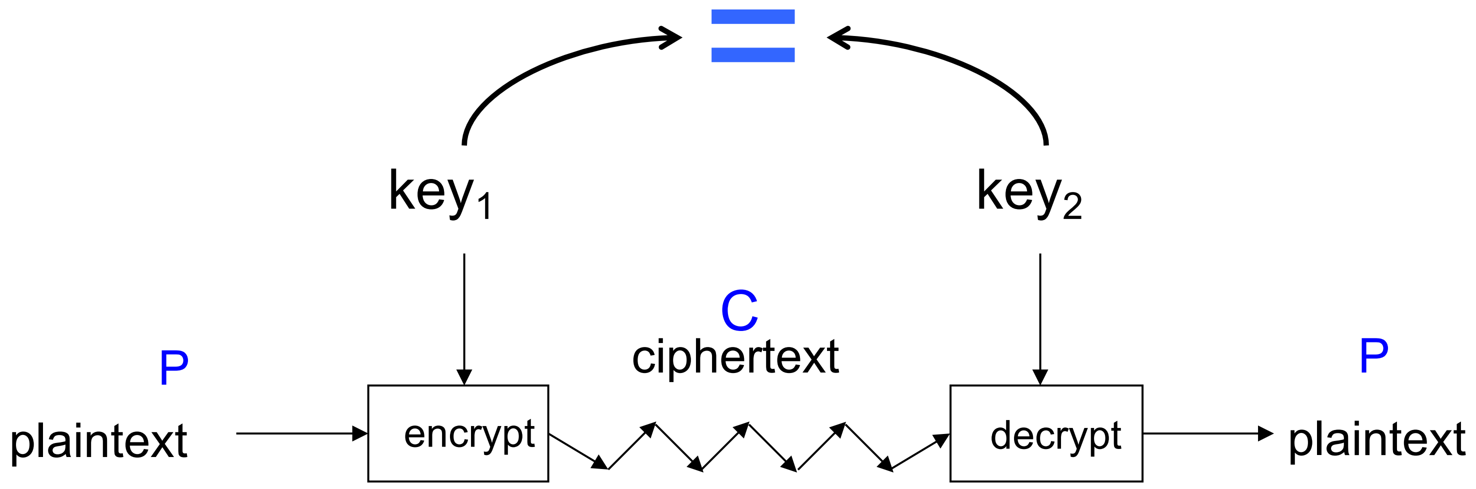
The latter is done through **encryption**.

- Change in the plaintext affect the final result

Confidentiality and Integrity with CBC

- Encrypt with one key, MAC with another key
- Why not use the same key?
  - Suppose modify the ciphertext
  - Decrypt this modified ciphertext won't result the original plaintext
  - CBC MAC calculation will be based on the "decrypted" plaintext with same key
  - MAC will be the same, cannot detect the integrity violation
- 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

Symmetric Key Cryptography

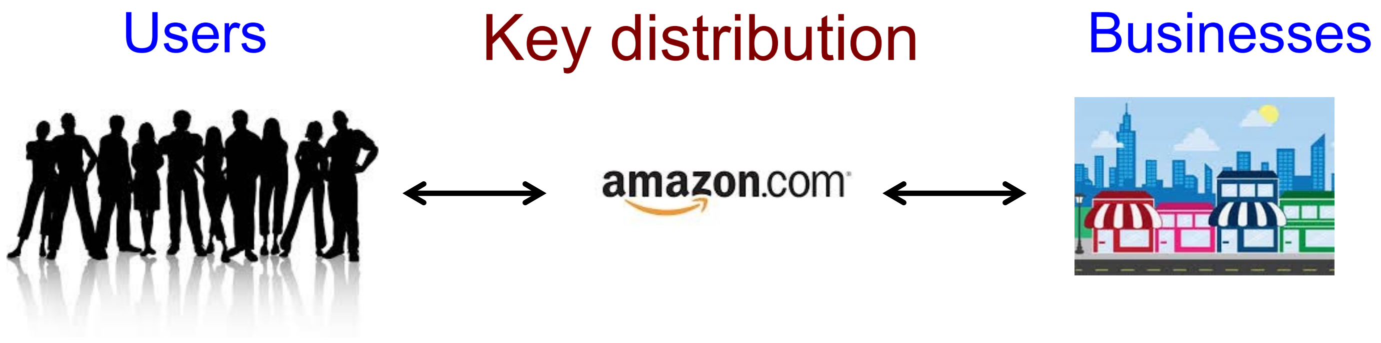


Summary on Symmetric Key Crypto

- **Stream cipher** — based on one-time pad
  - **A5/1:** based on shift registers; in GSM mobile phone system
  - **RC4:** Based on a changing lookup table
- **Block cipher** — based on codebook concept
  - **Feistel cipher**: a type of block ciphers
  - **DES**: a specific Feistel cipher; used to be a strong cipher but its 56 bit keys are too short
  - **3DES**: encrypt-decrypt-encrypt with 2 keys
  - **Others**: AES, IDEA, Blowfish, RC6
  - **Block cipher modes**: ECB, CBC (cipher block chaining), CTR

Uses for Symmetric Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC, or Message Authentication Code)
- Question: what is the key challenging of using symmetric key crypto?


Users
Key distribution
Businesses
amazon.com