



Symmetric Encryption

DAT159 – Basic Cryptography Module

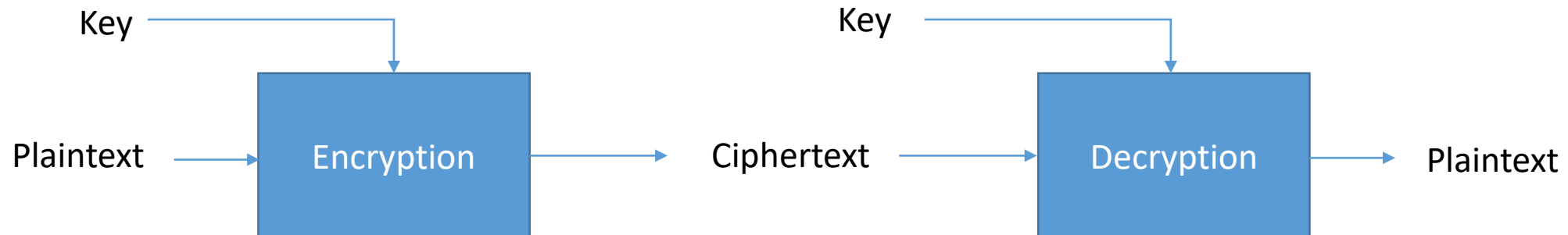
Tosin Daniel Oyetoyan

Previously

- We looked at historical ciphers
 - Shift, Substitution, Vigenere, Hill
- Popular war era cryptography machines such as the Enigma cipher used by the German army was based on a more sophisticated substitution cipher
- Relies on secret sharing
- 1950 –
 - Modern ciphers

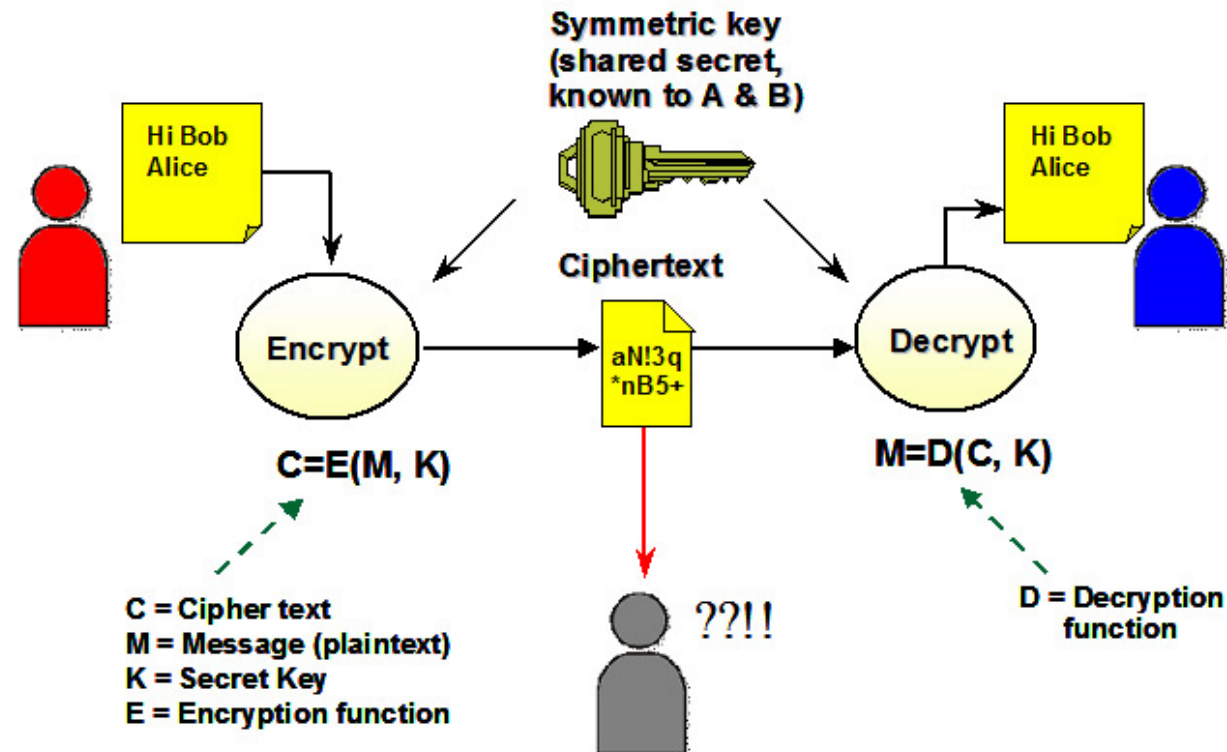
Symmetric Encryption

- Private key encryption
- Shared secret
- Secret key
- The secret key has to be shared securely between Bob and Alice
- A locked briefcase
 - You need to know the combination to unlock



Symmetric Encryption Security Application

- Confidentiality
- Very fast compared to asymmetric encryption
- Because of this, a hybrid approach is often used in practice



Challenges with symmetric cryptography

- The 2 parties must agree on secret key
- Key management problem
 - Not scalable: Users require many keys to manage. Users need unique pair of keys
- Key distribution problem
 - Requires a secure mechanism to deliver keys properly
- Does not provide non-repudiation
- Limited authentication (e.g. MAC)

A little about Boolean Algebra

We are particularly concerned with the Exclusive-OR (**XOR**) operator in cryptography

The XOR Truth table: XOR is equivalent to modulo 2 system

A	B	A XOR B	A mod B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

e.g.

$$1000001_2 \oplus 0110110_2 = 1110111_2$$

$$\begin{array}{rccccccc} & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ + & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$


Conversion between common number systems

- Binary
- Decimal
- Hexadecimal

Decimal to Binary


$$65_{10} = 1000001_2$$

Divisor	Dividend	Remainder
2	65	
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
2	1	0
	0	1



$$54_{10} = 110110_2$$

Divisor	Dividend	Remainder
2	54	
2	27	0
2	13	1
2	6	1
2	3	0
2	1	1
2	0	1



Binary to Decimal

Convert 1000001_2 to decimal number

$N_{10} = a_n \times 2^{n-1} + a_{n-1} \times 2^{n-2} + \dots + a_0 \times 2^0$ where n = number of bits and a is the bit at n location

$$\begin{aligned} N_{10} &= 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 64 + 0 + 0 + 0 + 0 + 0 + 1 = 65_{10} \end{aligned}$$

Binary to Hexadecimal

Break down the bits into nibbles (4-bit) group from right to left.

If the last bits are not up to 4, pad with zeros from left (MSB)

Convert each nibble to decimal

If decimal is greater than 9, encode as A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

Convert 1000001_2 to Hexadecimal number

$1000001 = 0100\ 0001$

$0100 = 4$

$0001 = 1$

$= 41_{16}$

Hexadecimal to Binary

- Obtain the binary digits (nibble) for each number from the hexadecimal number
- Put all the binary together into one binary sequence.
- Convert back to decimal

Convert 41_{16} to binary

$$41_{16} = 0100\ 0001_2$$

Exercise (5mins)

- Convert 76 to binary
- Convert A5 to Decimal
- Convert 1000100101_2 to hex

Solution

- $76 = 1001100_2$
- $A5 = 1010\ 0101_2 = 165$
- $1000100101_2 = 0010\ 0010\ 0101 = 225_{\text{hex}}$

Character encoding

ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol		
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Plaintext: BERGEN IS BEAUTIFUL

ASCII: 6669827169781277383127666965858473708576

HEX: 42455247454E7F49537F42454155544946554C

Binary: 10000100100010101010010010001110100010101001110011111110100100101010011011111110100001001000101010000010101010101010001001001010001100101010101001100

Block vs. Stream Ciphers

- Block Ciphers: Successive plaintext elements are obtained using the same Key K

$$y = y_1 y_2 \dots = e_K(x_1) e_K(x_2) \dots$$

- Stream Ciphers: Generate a key stream $Z = z_1 z_2 \dots$ to encrypt a plaintext $x = x_1 x_2 \dots$

$$y = y_1 y_2 \dots = e_{z_1}(x_1) e_{z_2}(x_2) \dots$$

Stream Ciphers

- The plaintext, the ciphertext and the key stream consist of individual bits.
- $x_i, y_i, k_i \in \{0, 1\}$

$$\text{Encryption: } y_i = e_{k_i}(x_i) \equiv x_i + k_i \text{ mod } 2$$

$$\text{Decryption: } x_i = d_{k_i}(y_i) \equiv y_i + k_i \text{ mod } 2$$

This is an XOR operation

And both the encryption and decryption are the same function

Example

Alice wants to encrypt the letter A using the keystream $k = 0101100$ and send it to Bob

ASCII value for A = 65_{10}

$$65_{10} = 1000001_2$$



$$A = 1000001_2$$

1 0 0 0 0 0 1

\oplus

0 1 0 1 1 0 0

1 1 0 1 1 0 1

1101101 = m



1 1 0 1 1 0 1

\oplus

0 1 0 1 1 0 0

1 0 0 0 0 0 1

$$A = 1000001_2$$

Security in cryptography

Kerckhoffs' Principle: The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key.

Security by obscurity

- Computational security
- Unconditional security

Unconditional security

- A crypto system is unconditionally secured or information-theoretically secure if it cannot be broken even with infinite computational resources
 - Assumes no limit on computational resources
 - e.g. One-Time Pad
 - Key stream is generated by a true random number
 - Encryption key length is the same as the length of the plaintext
 - and each key is used only once (not repeated)
 - All known practical algorithms are not unconditionally secure
 - We only think about computational security

Computational Security

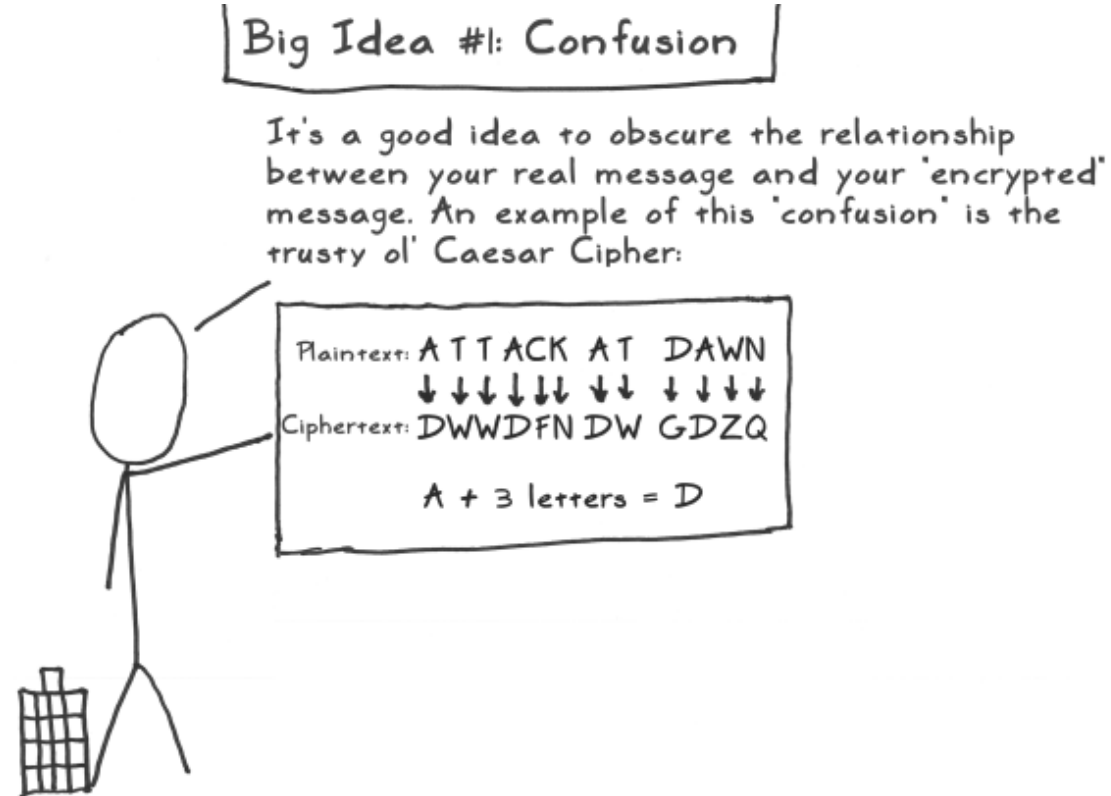
- A crypto system is computationally secure if the best known algorithm for breaking it requires at least t operations
 - But we don't know what the best algorithm is for a given attack
 - And if we know the lower bound of the complexity of one attack, we do not know whether any other more powerful attacks exist (e.g. substitution cipher)
- For a symmetric cipher, we can only hope that there is no attack method with a complexity better than an exhaustive key search

Basics of cryptography

- Confusion
- Diffusion
- Key secrecy

Confusion

An encryption operation where the relationship between key and ciphertext is obscured. A common approach for achieving confusion is through **substitution**



Diffusion

An encryption operation where the influence of one plaintext symbol is spread over many ciphertext symbols in order to hide the statistical properties of the plaintext. Bit permutation is one approach to achieve diffusion.

Big Idea #2: Diffusion

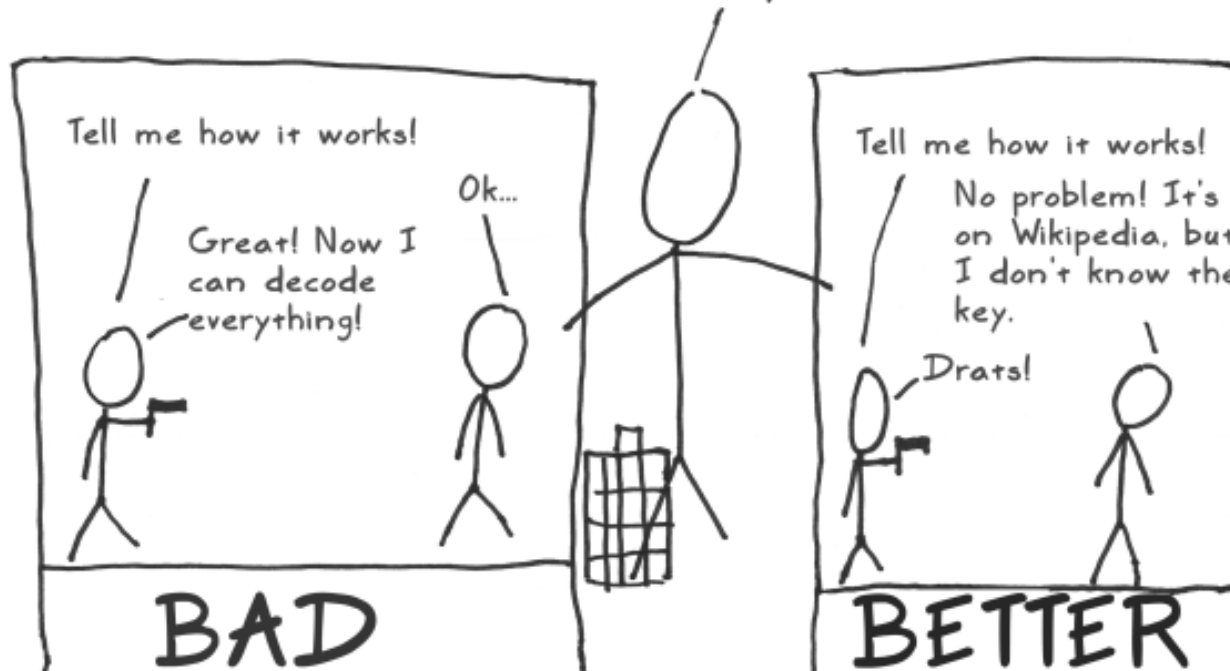
It's also a good idea to spread out the message. An example of this 'diffusion' is a simple column transposition:



Key secrecy

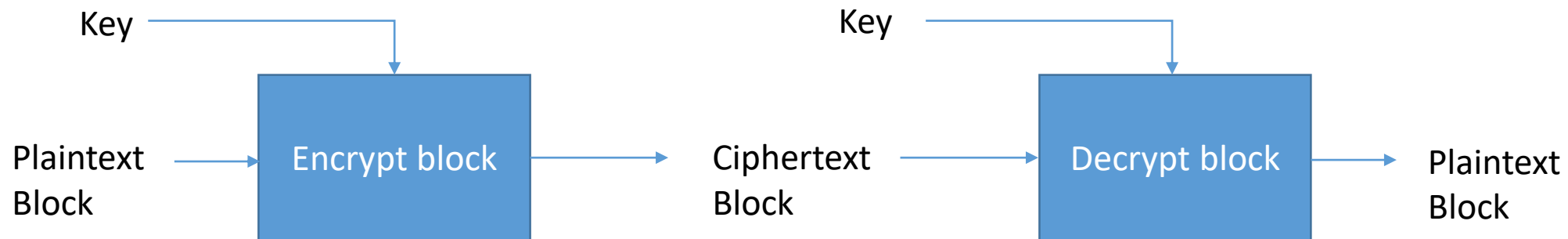
Big Idea #3: Secrecy Only in the Key

After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.

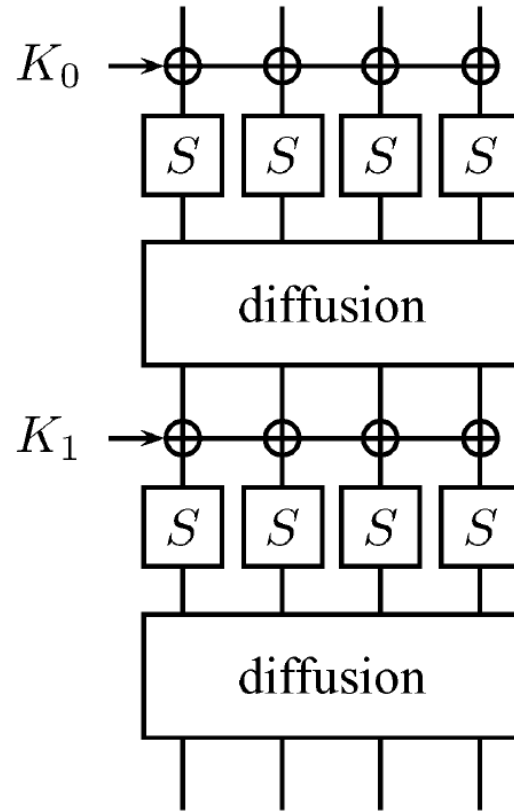
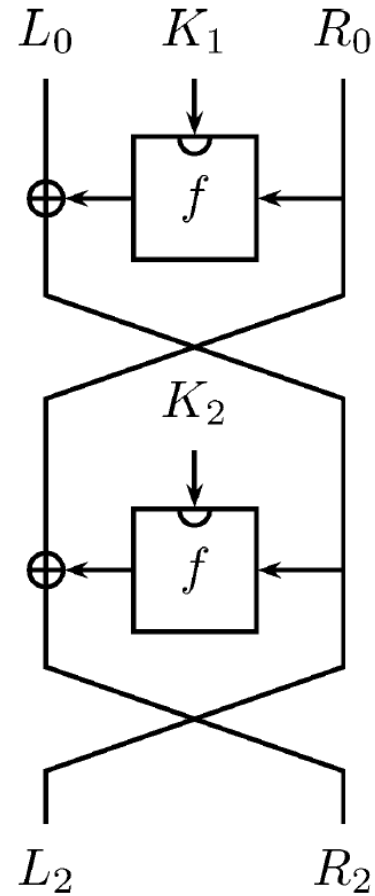


Block ciphers – basic idea

- Operates on a fixed size block of data (e.g. 128bits blocks)
- Uses a secret key (relatively same length)
- If the final block size is less than the fixed size, it must be padded
- Based on Feistel and SP networks



Feistel Cipher vs. SP Network



Feistel Cipher

- Examples of Block Ciphers using a Feistel structure:
- DES
 - Published 1977
 - Designed by IBM
- Blowfish
 - Published 1992
 - Designed by Bruce Schneier
- RC5
 - Published 1994
 - Designed by Ron Rivest

SP Network

Examples of Block Ciphers using a SP Network structure:

- AES (Rijndael)
 - Published 1998
 - Designed by Vincent Rijmen and Joan Daemen
- CAST-128
 - Published 1996
 - Designed by Carlisle Adams and Stafford Tavares
- IDEA - International Data Encryption Algorithm
 - Published 1991
 - Designed by Xuejia Lai and James Massey

Modes of Operation

- How to encrypt messages with arbitrary-length using a block cipher
 - We divide message into blocks
 - Then, we encrypt each block independently
- The last block may need to be extended to fit the block size
 - Padding
- Some modes need an additional input value
 - Initialization vector

Padding

- Different padding schemes
 - Zero
 - ... | DD DD DD DD DD DD DD DD | DD DD DD DD **00 00 00 00** | (add zero bytes)
 - OneAndZeroes Padding
 - ... | 1011 1001 1101 0100 0010 0111 **0000 0000** | (padded with 0x80 byte and add zeros to make the desired bytes)
 - ANSI X923
 - ... | DD DD DD DD DD DD DD DD | DD DD DD DD **00 00 00 04** | (last byte specifies the number of padded bytes with the other padding bytes as zero)
 - ISO 10126
 - ... | DD DD DD DD DD DD DD DD | DD DD DD DD **81 A6 23 04** | (last byte specifies the number of padded bytes with other bytes randomly chosen)
 - PKCS#5
 - ... | DD DD DD DD DD DD DD DD | DD DD DD DD **04 04 04 04** | (append required byte length repeatedly)

Padding security

- Good padding scheme
 - Padded bit/bytes are random
- Encryption does not conceal the length of the message being encrypted
- Choice of padding scheme affects message security

Initialization Vector (IV) - Nonce

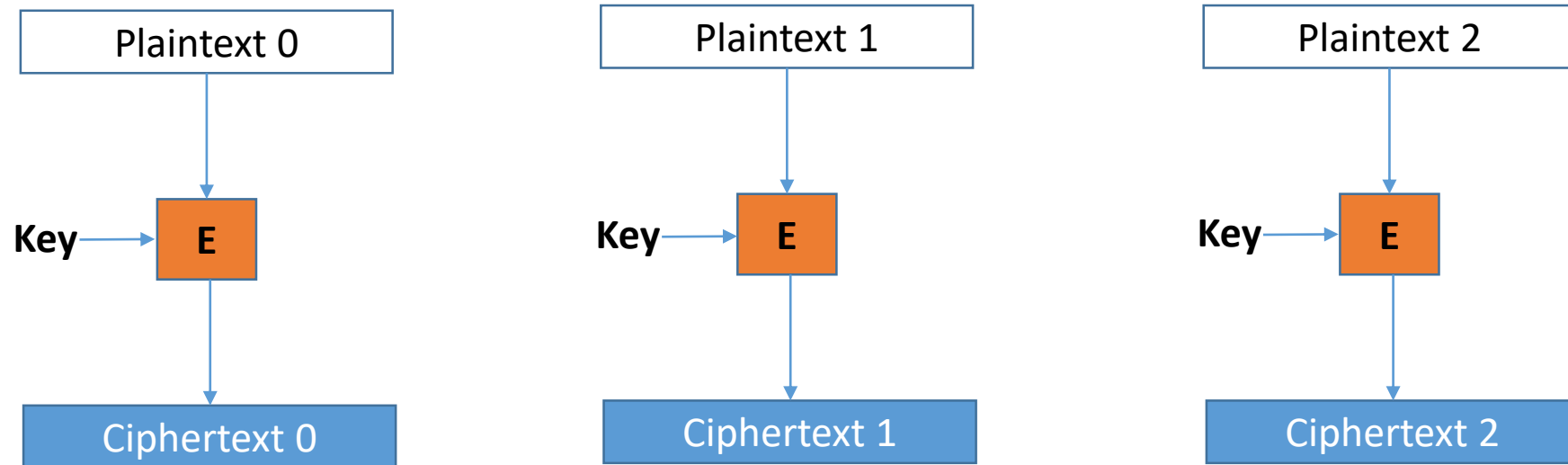
- It is a fixed-sized input value
- Should be random
- Unique
- Unpredictable

Modes of Operation

- Electronic Code Book (ECB) Mode
- Cipher Block Chaining (CBC) mode
- Cipher Feedback mode (CFB)
- Output Feedback mode (OFB)
- Counter mode (CTR)

Electronic Code Book (ECB) Mode

- Split the message into blocks of n bytes of data (e.g. 16bytes)
- Pass each block through the encryption algorithm
- Uses the same key each time

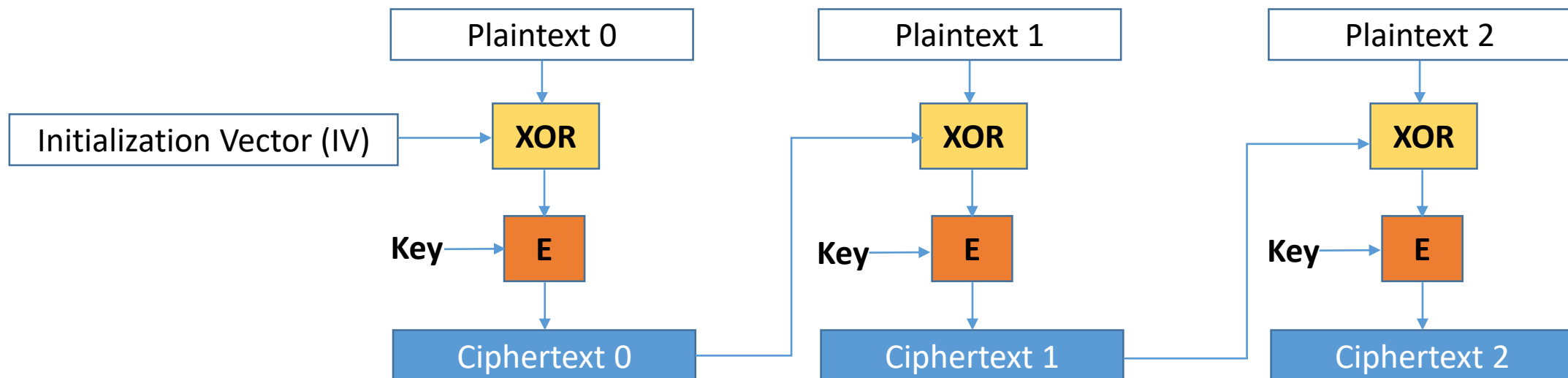


ECB Mode

- Strengths
 - No need of block synchronisation
 - If there is transmission problems or bit errors, it is still possible to decrypt received blocks
 - Block ciphers in ECB mode can be parallelized
- Weaknesses
 - Encryption is highly deterministic (i.e. Identical plaintext block results in identical ciphertext block)
 - Plaintext blocks are encrypted independently of previous blocks
 - Plaintext patterns are still visible after encryption

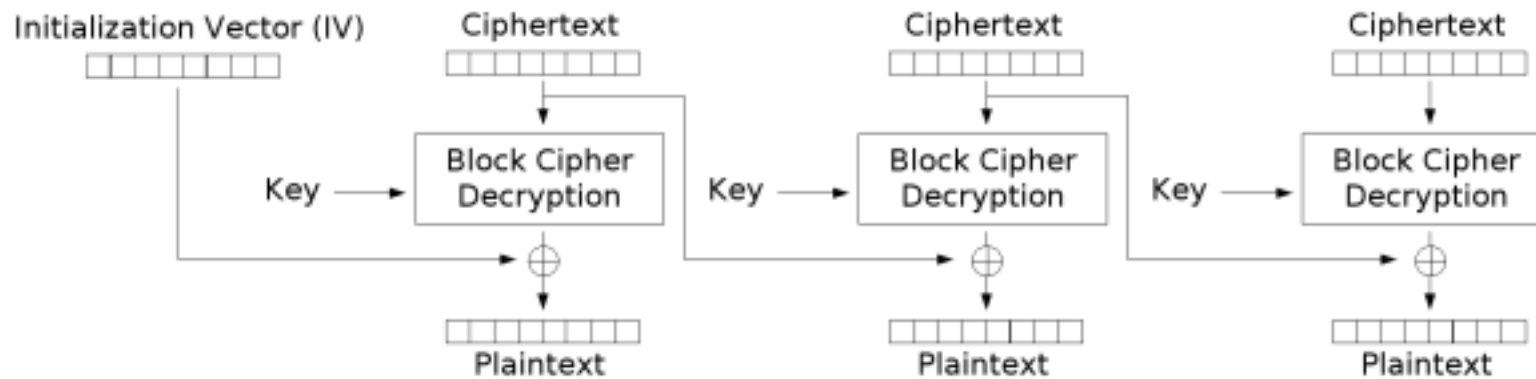
Cipher Block Chaining (CBC) mode

- Encryption is dependent on values from the previously encrypted block
- The encryption of all blocks are chained together
 - Each plaintext block is XOR'ed with the previous ciphertext block before encryption
- The encryption is randomized using an initialization vector (IV) (nonce)



CBC Mode

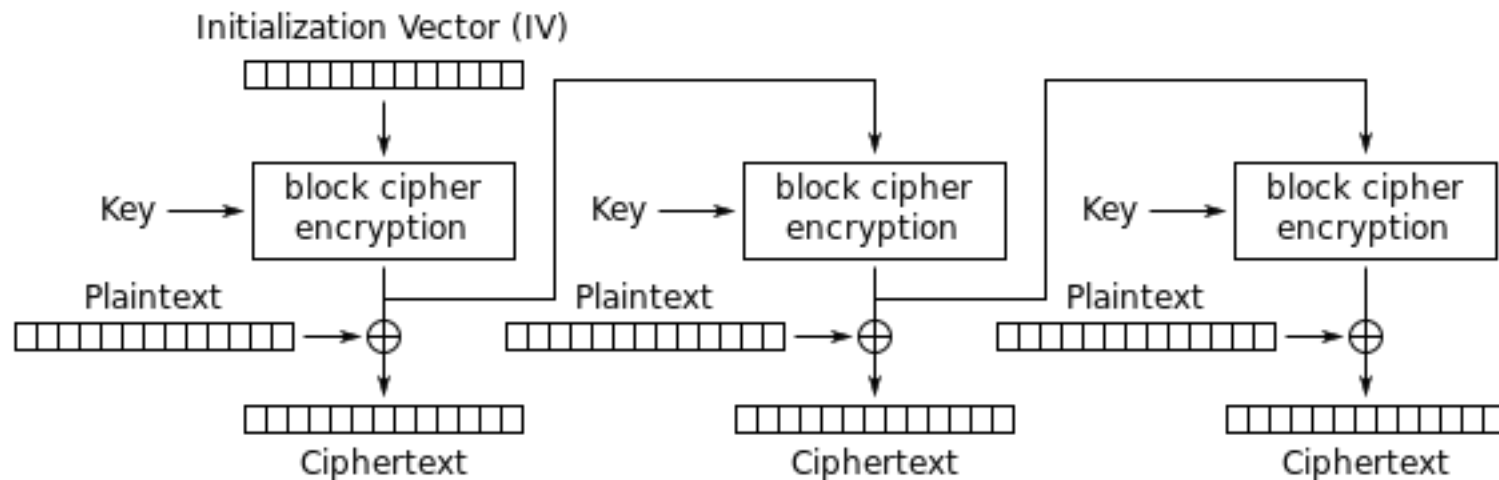
- Strengths
 - Same messages using the same keys will be encrypted to different ciphertexts
 - Message can be decrypted from any part and decryption function can therefore be parallelized
 - Plaintext patterns are not visible (blurred)
- Weaknesses
 - Encryption has to be performed sequentially
 - Bit error in one block affects preceding blocks



Cipher Block Chaining (CBC) mode decryption

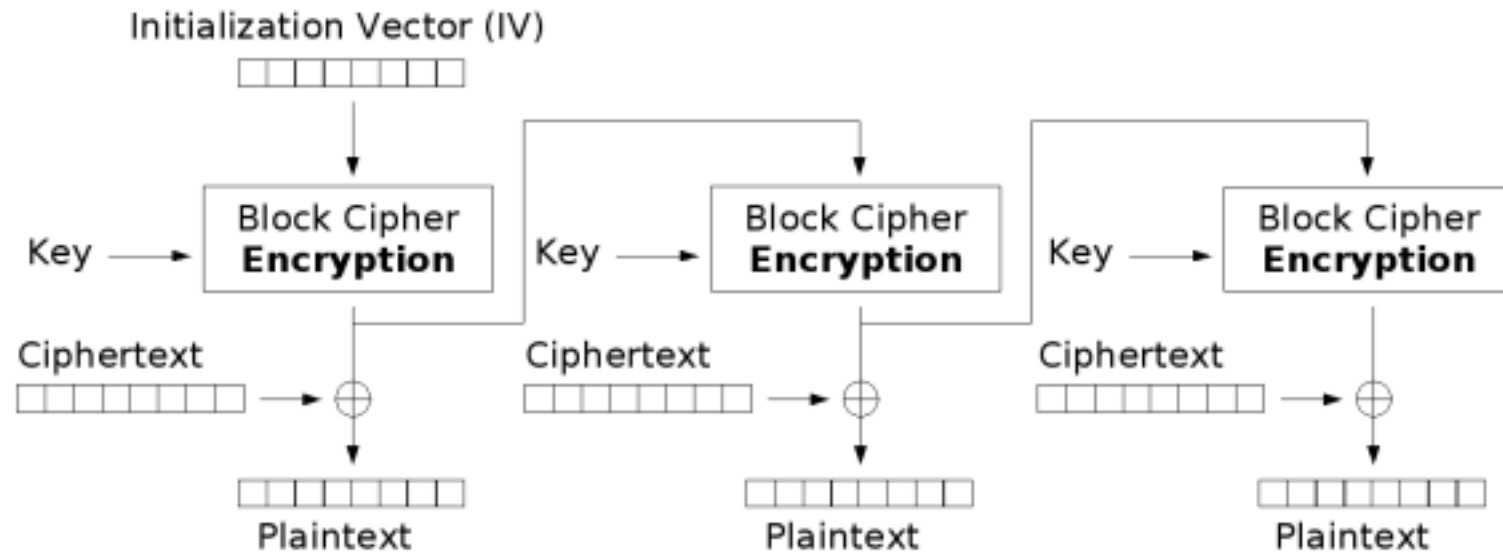
Output Feedback Mode (OFB)

- The whole output of the previous block's calculation is used as input for the next block's encryption
- Uses the block cipher as a building block to build a stream cipher encryption scheme
- The output of the cipher gives key stream bits that can be used to encrypt plaintext bits using the XOR operation



OFB Mode

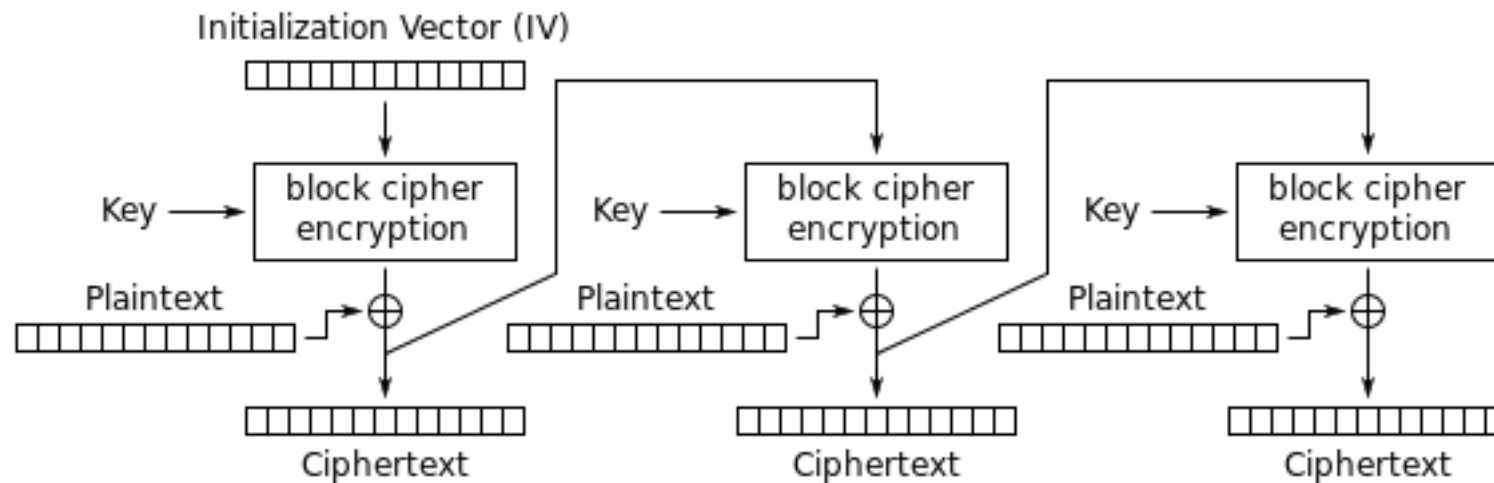
- Strengths
 - Keystream can be pre-computed
 - No padding
- Weaknesses
 - Keystream computation cannot be parallelized
 - Susceptible to bit-flipping attacks



Output Feedback (OFB) mode decryption

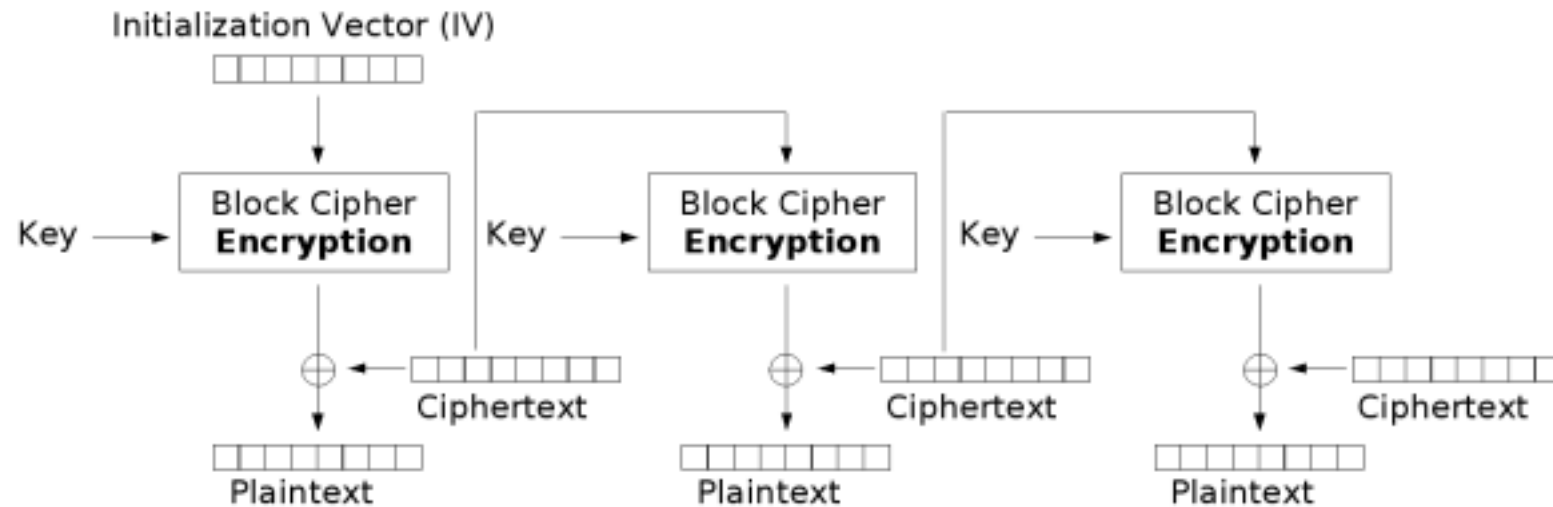
Cipher Feedback (CFB) Mode

- Previous ciphertext is used to encrypt the next block of data
 - Similar to OFB but uses only the ciphertext as feedback



CFB Mode

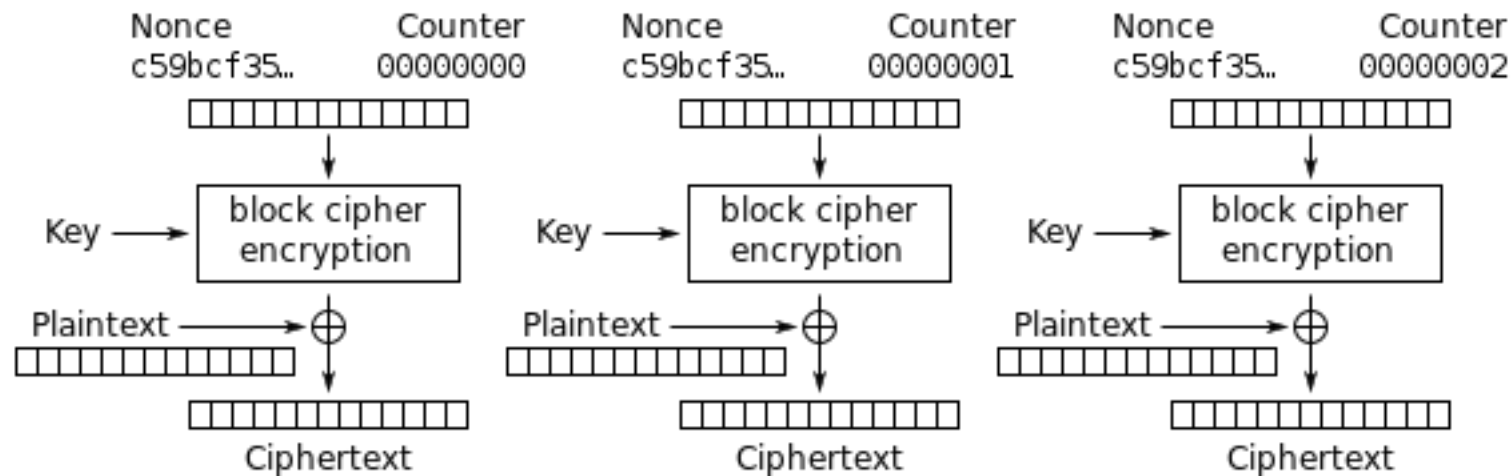
- Strengths
 - No padding
 - Decryption can be parallelized
- Weaknesses
 - Encryption cannot be parallelized
 - No pre-computation of the keystream
 - Susceptible to bit-flipping attacks



Cipher Feedback (CFB) mode decryption

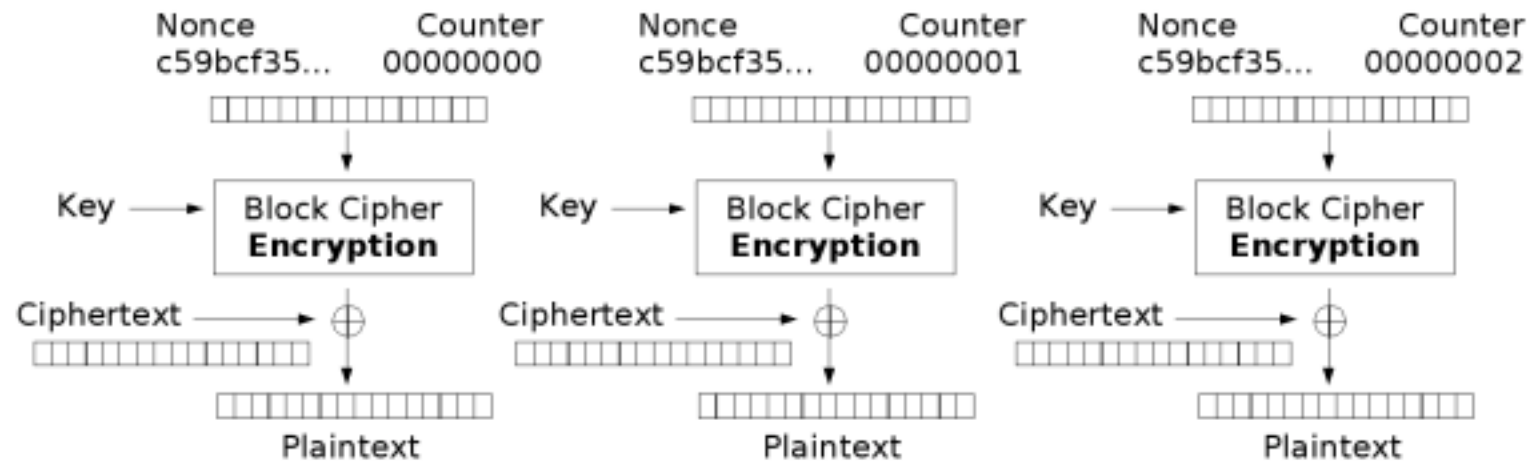
Counter (CTR) Mode

- Also uses block cipher as a stream cipher
- The input to the block cipher is a counter that takes new value each time a new key stream is computed
- Encryption/decryption of a block does not depend on the previous blocks



CTR Mode

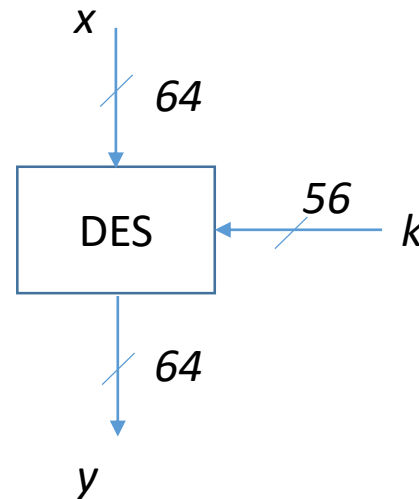
- Strengths
 - Encryption and decryption can be parallelized
 - No padding
 - Keystream can be pre-computed (in parallel)
- Weaknesses
 - Dangerous when key and nonce/counter are reused
 - Susceptible to bit-flipping attacks



Counter (CTR) mode decryption

Data Encryption Standard (DES)

- Published by National Institute of Standards and Technology (NIST)
- Replaced in 1999 with Advanced Encryption Standard (AES)
- DES encrypt 64 bits length of blocks using 56 bits key size
- Uses the Feistel network



Feistel Network

- Symmetric structure named after the German IBM cryptographer Horst Feistel
- Product ciphers
 - Permutation (bit-shuffling) - produces diffusion
 - Non-linear functions (substitution boxes or S-boxes) – produces confusion
 - Linear mixing using XOR to perform confusion and diffusion

If f is the round function and k_1, k_2, \dots, k_n are the subkeys for the rounds $1, 2, \dots, n$ respectively

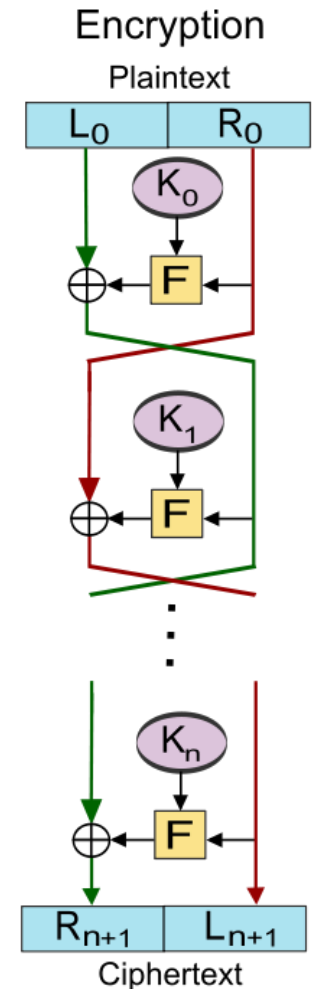
Split the plaintext x into 2 equal pieces (L_1, R_1)

For each round $i = 1, 2, \dots, n$, we compute:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus f(R_i, k_i)$$

The final ciphertext $y = (R_{n+1}, L_{n+1})$



Feistel Network

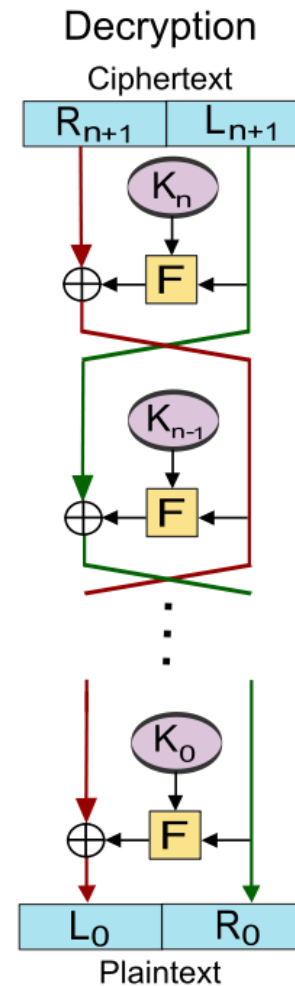
To decrypt a ciphertext $y = (R_n, L_n)$ we basically reverse the process:

For each round $i = n, n - 1, \dots, 1$, we compute:

$$R_i = L_{i+1}$$

$$L_i = R_{i+1} \oplus f(L_{i+1}, k_i)$$

The final plaintext $x = (L_1, R_1)$



Data Encryption Standard (DES)

1. Initial Permutation (IP) of 64-bit plaintext x
2. Split x into 2 equal parts (32-bit each) i.e. L_0 and R_0
3. Create 16 subkeys (k_1, \dots, k_{16}) from k
4. 16 rounds of encryption (Feistel structure)

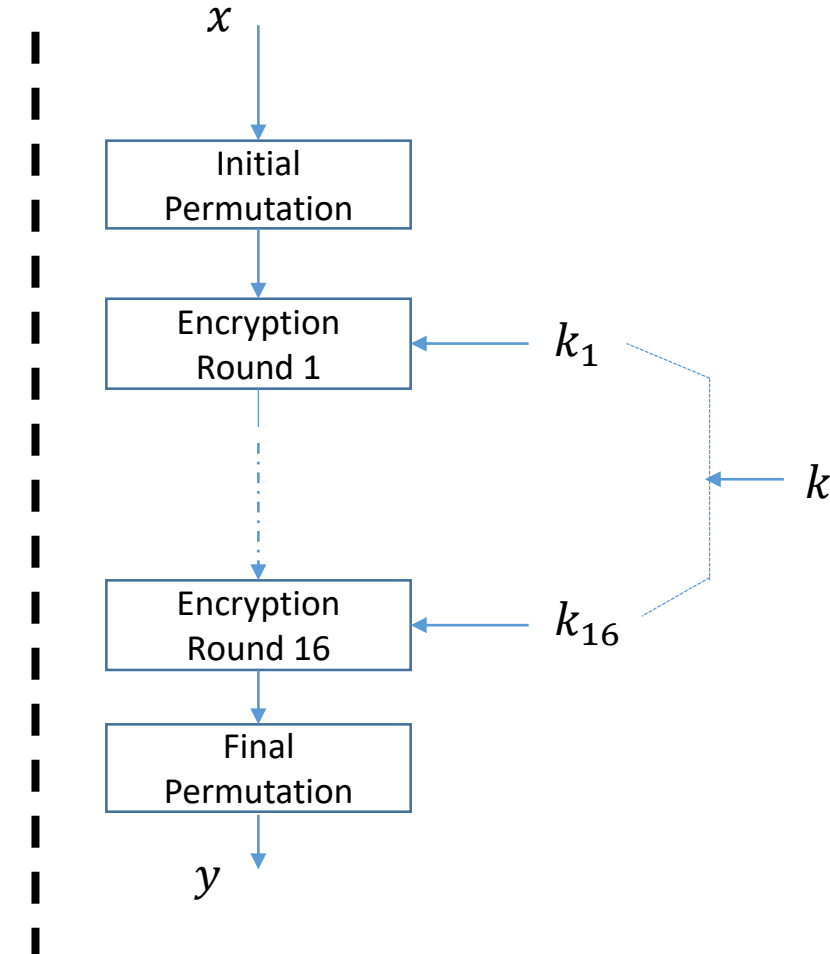
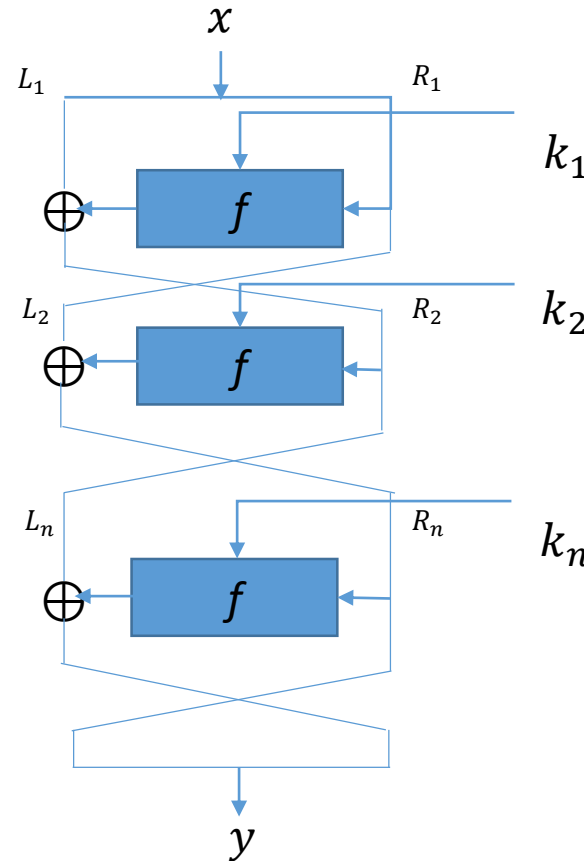
For each round $i = 1, 2, \dots, n, (n = 16)$
we compute:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus f(R_i, k_i)$$

$$\text{Output} = (R_{n+1}, L_{n+1})$$

5. Final permutation
Inverse of IP (IP^{-1}) is performed on (R_{n+1}, L_{n+1})



Nearly working Example

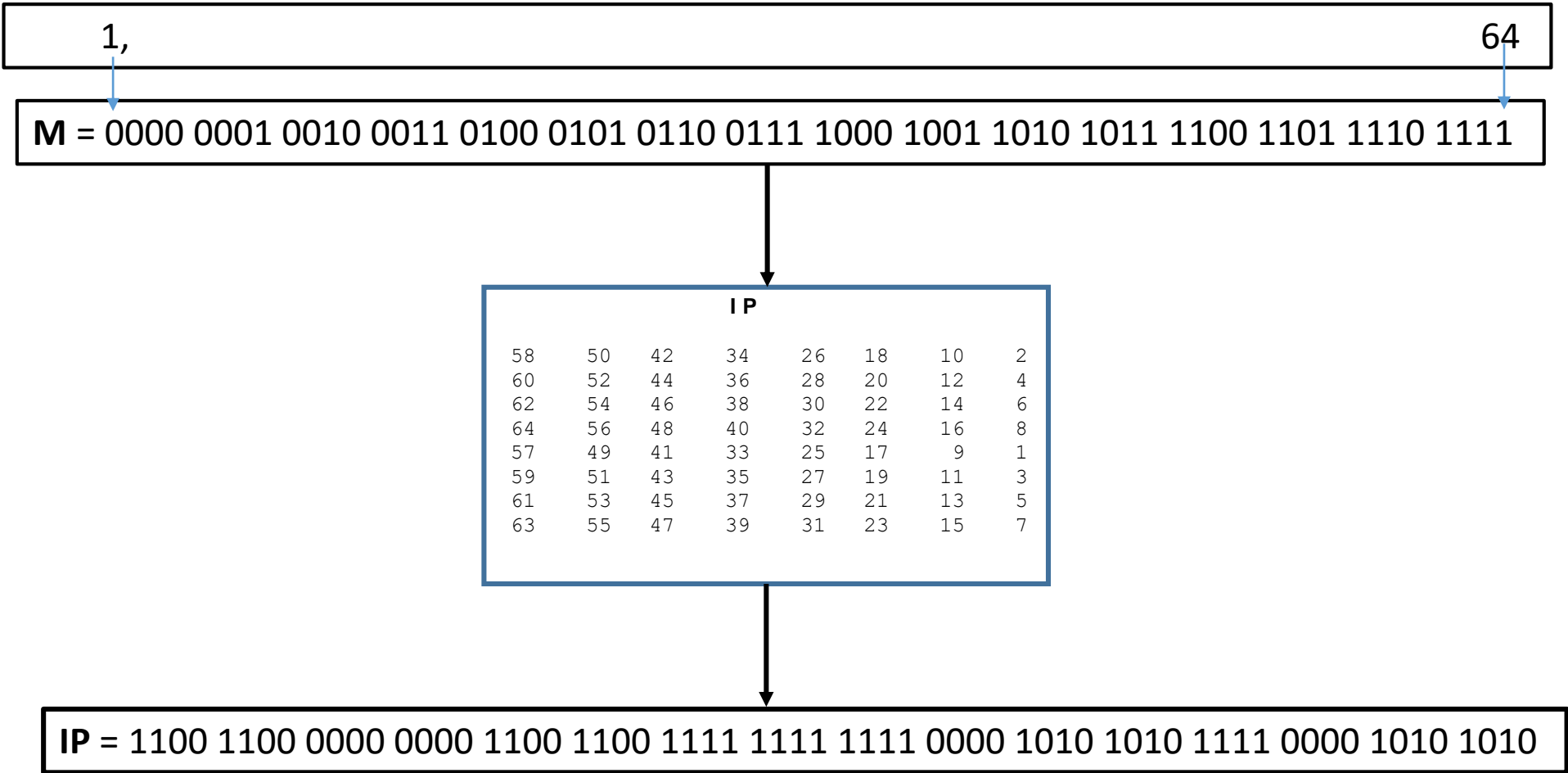
- Plaintext Message M = 0123456789ABCDEF
- K = I33457799BBCDFFI

In binary format

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

1. Initial Permutation (IP) of 64-bit plaintext x



Inverse of permutation table (revisited)

IP \rightarrow IP⁻¹

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

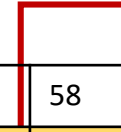
	58	50	42	34	26	18	10	2
i								
	60	52	44	36	28	20	12	4
i								
	62	54	46	38	30	22	14	6
i								
	64	56	48	40	32	24	16	8
i								
	57	49	41	33	25	17	9	1
i								
	59	51	43	35	27	19	11	3
i								
	61	53	45	37	29	21	13	5
i								
	63	55	47	39	31	23	15	7
i								

	58	50	42	34	26	18	10	2
i	1	2	3	4	5	6	7	8
	60	52	44	36	28	20	12	4
i	9	10	11	12	13	14	15	16
	62	54	46	38	30	22	14	6
i	17	18	19	20	21	22	23	24
	64	56	48	40	32	24	16	8
i	25	26	27	28	29	30	31	32
	57	49	41	33	25	17	9	1
i	33	34	35	36	37	38	39	40
	59	51	43	35	27	19	11	3
i	41	42	43	44	45	46	47	48
	61	53	45	37	29	21	13	5
i	49	50	51	52	53	54	55	56
	63	55	47	39	31	23	15	7
i	57	58	59	60	61	62	63	64



We need to find the inverse of IP (IP^{-1})

	58	50	42	34	26	18	10	2
IP^{-1}	55	53	51	49	56	54	52	50
	60	52	44	36	28	20	12	4
IP^{-1}	39	37	35	33	40	38	36	34
	62	54	46	38	30	22	14	6
IP^{-1}	23	21	19	17	24	22	20	18
	64	56	48	40	32	24	16	8
IP^{-1}	7	5	3	1	8	6	4	2
	57	49	41	33	25	17	9	1
IP^{-1}	63	61	59	57	64	62	60	58
	59	51	43	35	27	19	11	3
IP^{-1}	47	45	43	41	48	46	44	42
	61	53	45	37	29	21	13	5
IP^{-1}	31	29	27	25	32	30	28	26
	63	55	47	39	31	23	15	7
IP^{-1}	15	13	11	9	16	14	12	10



	58	50	42	34	26	18	10	2
IP ⁻¹	55	53	51	49	56	54	52	50
	60	52	44	36	28	20	12	4
IP ⁻¹	39	37	35	33	40	38	36	34
	62	54	46	38	30	22	14	6
IP ⁻¹	23	21	19	17	24	22	20	18
	64	56	48	40	32	24	16	8
IP ⁻¹	7	5	3	1	8	6	4	2
	57	49	41	33	25	17	9	1
IP ⁻¹	63	61	59	57	64	62	60	58
	59	51	43	35	27	19	11	3
IP ⁻¹	47	45	43	41	48	46	44	42
	61	53	45	37	29	21	13	5
IP ⁻¹	31	29	27	25	32	30	28	26
	63	55	47	39	31	23	15	7
IP ⁻¹	15	13	11	9	16	14	12	10

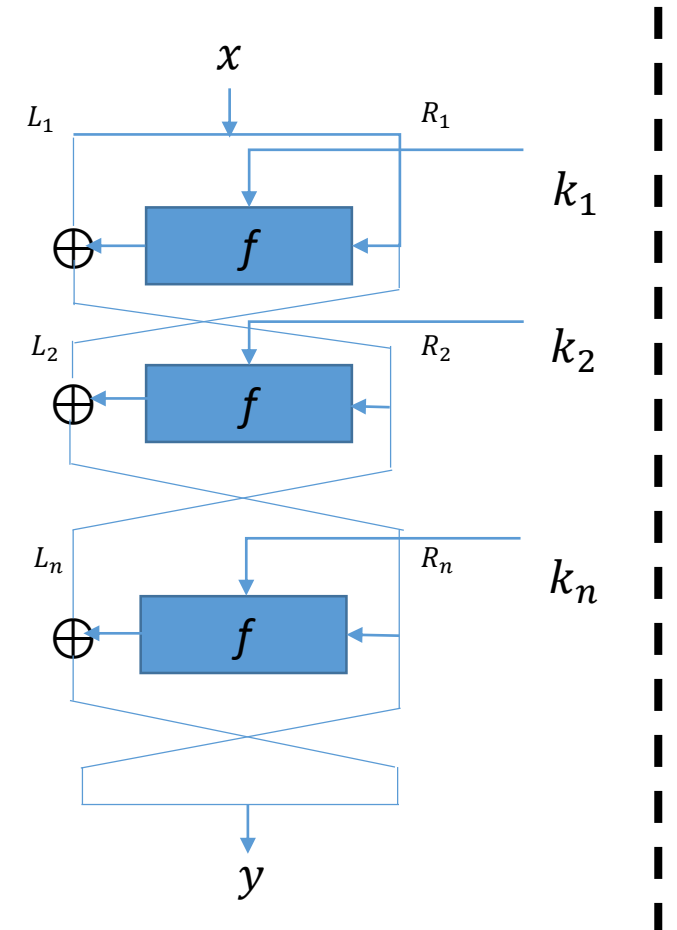
IP ⁻¹							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

2. Split x into 2 equal parts (32-bit each) i.e. L_0 and R_0

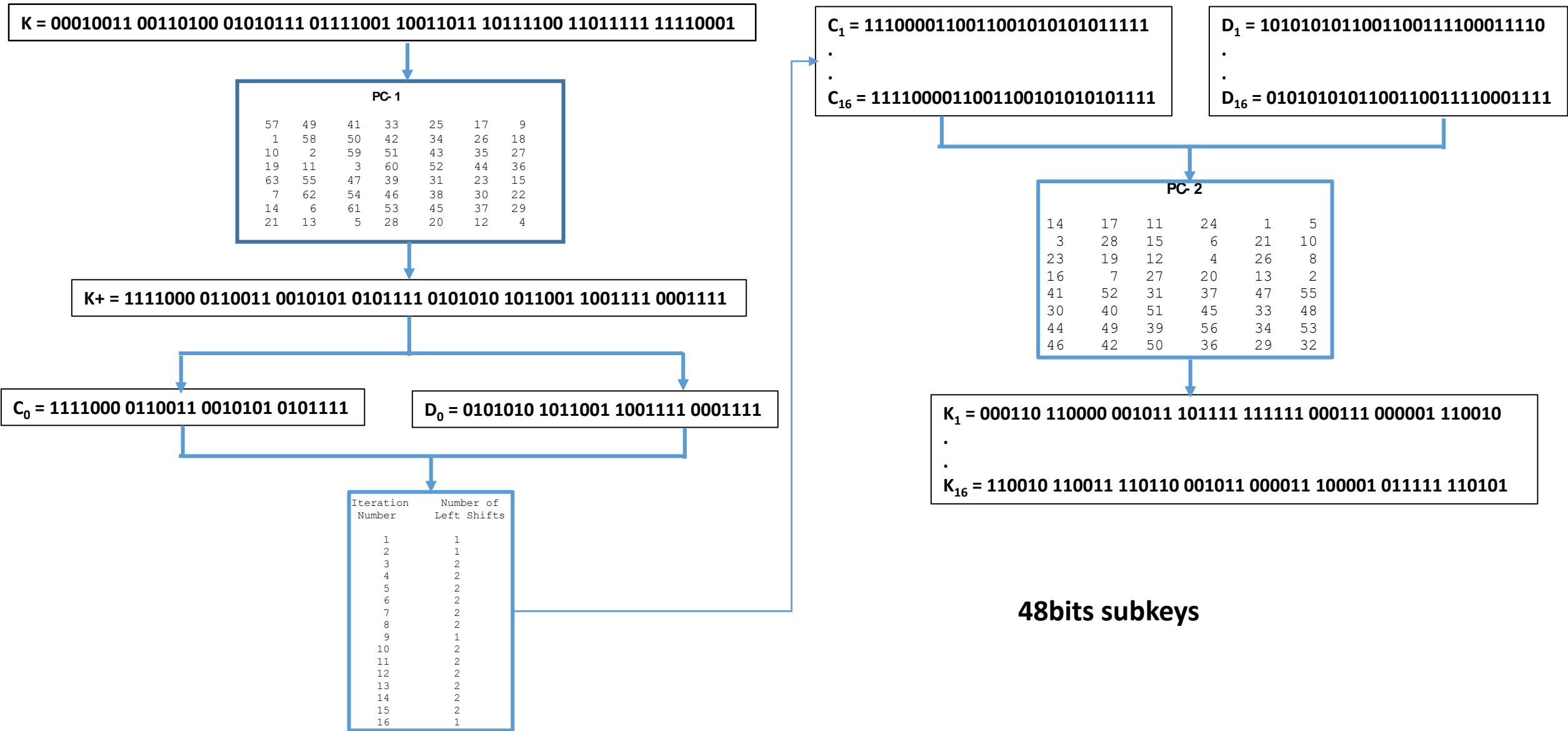
IP
1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

L_1
1100 1100 0000 0000 1100 1100 1111 1111

R_1
1111 0000 1010 1010 1111 0000 1010 1010



3. Create 16 subkeys (k_1, \dots, k_{16}) of 56bits length from k



For each round $i = 1, 2, \dots, n, (n = 16)$
we compute:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus f(R_i, k_i)$$

$$\text{Output} = (R_{n+1}, L_{n+1})$$

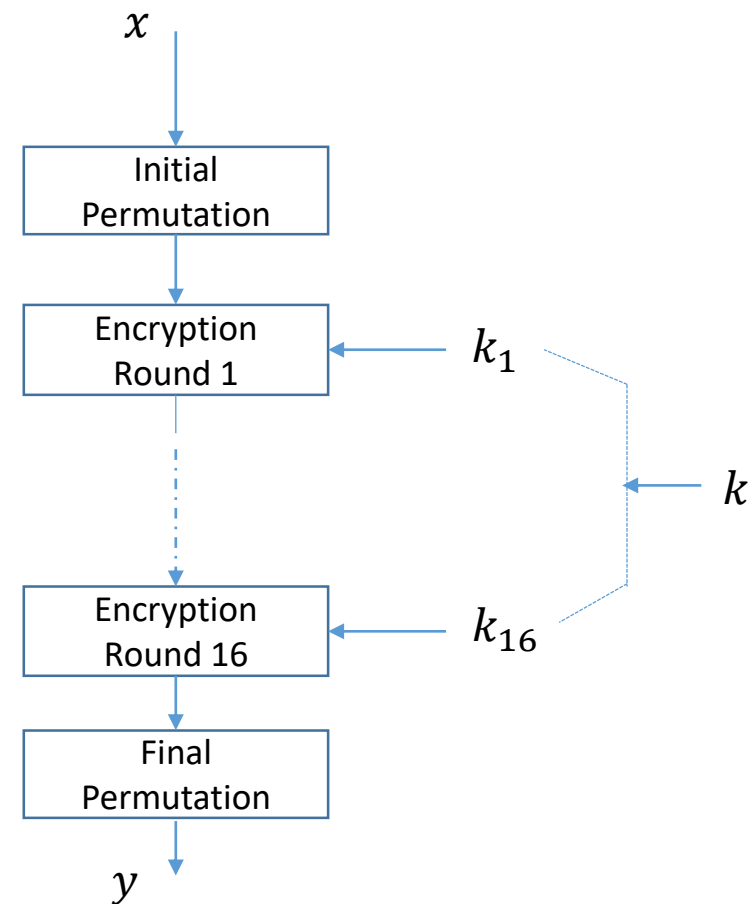
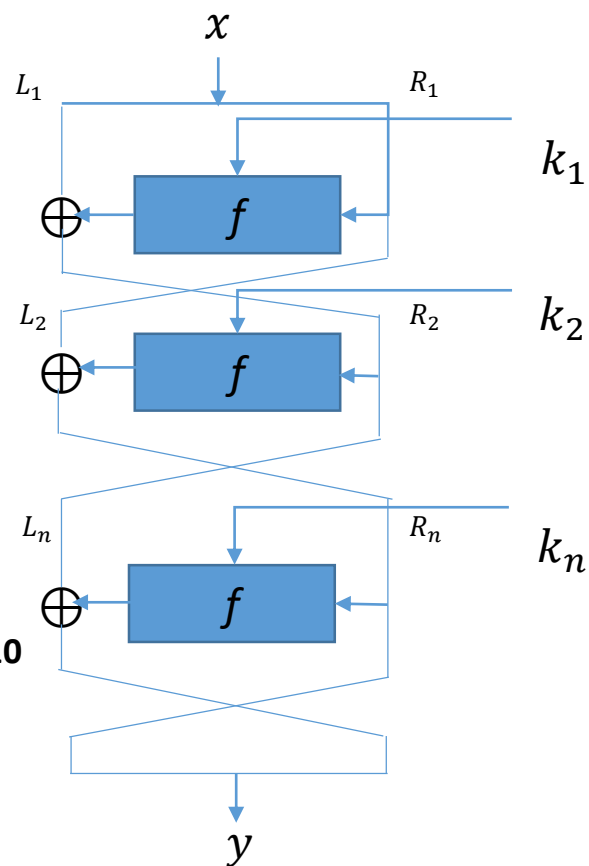
For $i = 1$

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

$$L_2 = R_1 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

$$R_2 = L_1 \oplus f(R_1, k_1)$$

$$f(R_1, k_1)??$$



How $f(R_1, k_1)$ works

$R_1 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$

E BIT- SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

$E(R_1) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

\oplus

$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$

$K_1 \oplus E(R_1) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111$

S1

		Col umn Number															
Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$
 $= 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$

6bits used to determine the address in the S boxes as follows:

Bit 1 and **6** are combined and converted to decimal to know the column.

Bit 2 – 5 are converted to decimal to know the row
 e.g. $S_1(011000) = (00, 1100) = (0, 12) = 5\ (0101)$

Putting it together

For $i = 1$

$L_1 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$

$f_1 = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$

$R_2 = L_1 \oplus f(R_1, k_1)$

$= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$
 $+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$
 $= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100$

For $i = 16$

$L_{16} = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$
 $R_{16} = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$

Reverse the order $L_{16}R_{16} \rightarrow R_{16}L_{16}$

$R_{16}L_{16} = 00001010\ 01001100\ 11011001\ 10010101\ 01000011\ 01000010\ 00110010\ 00110100$

IP⁻¹

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

IP⁻¹ = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

85E813540F0AB405 (Hexadecimal)

Brute-force attack on DES

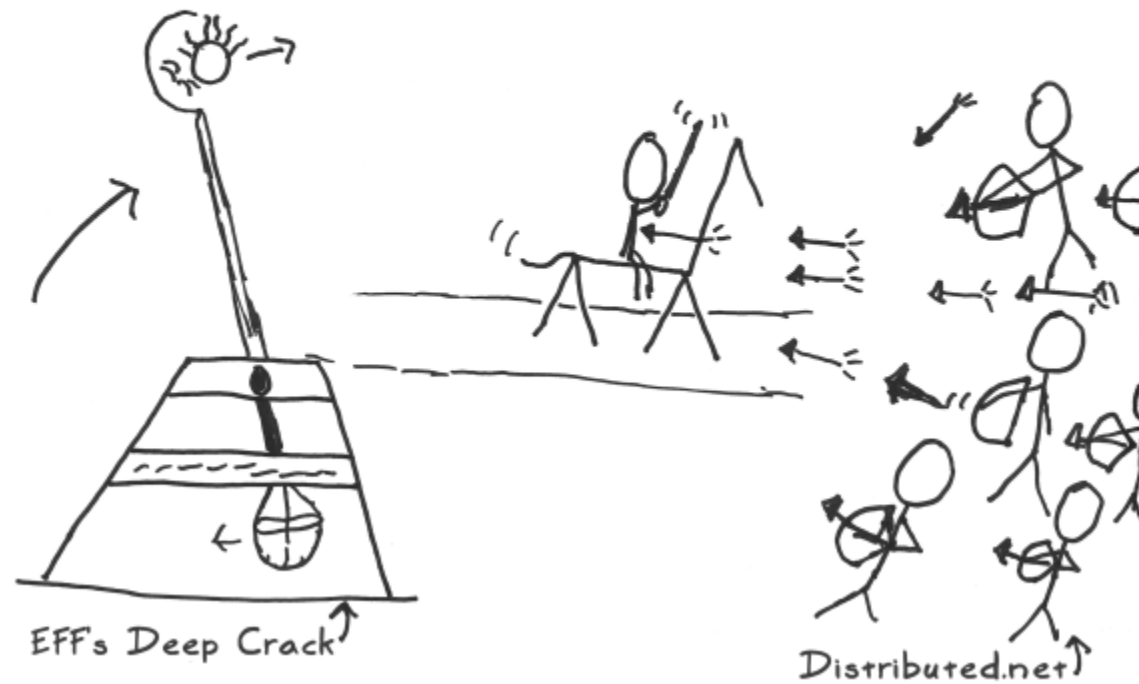
- 56-bit key size
- 2^{56}
- On a dedicated hardware
 - Crack.sh could exhaust the key space in ~ 26 hours
 - Using:
 - 48 Xilinx Virtex-6 LX240T FPGAs
 - Each FPGA contains a design with 40 fully pipelined DES cores running at 400MHz
 - $= 400,000,000 * 40 = 16,000,000,000$ keys/sec
 - With 48 FPGAs $= 768,000,000,000$ keys/sec
 - $2^{56} / 768,000,000,000 = \sim 26$ hours

Exercise (1min)

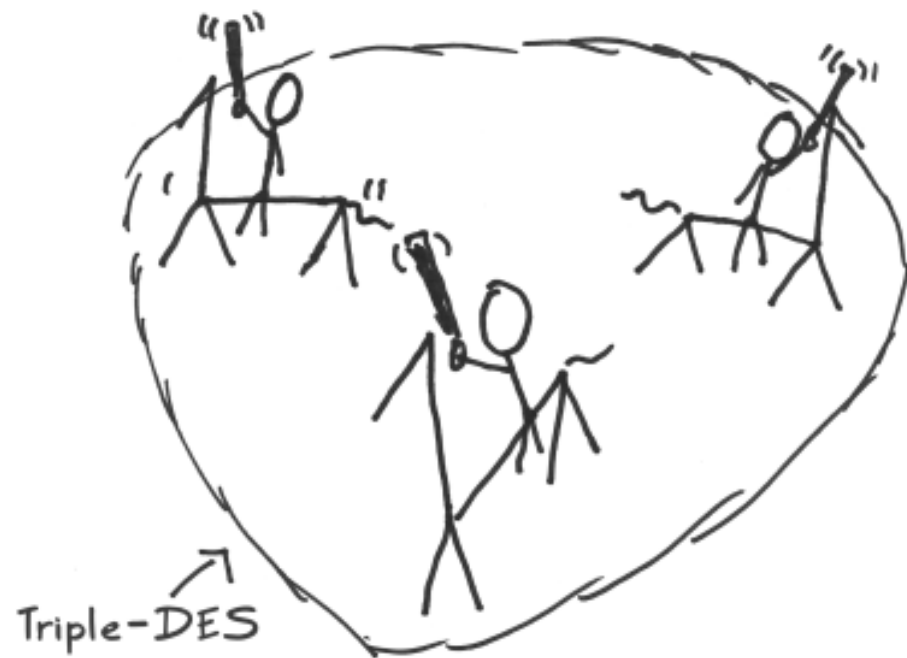
- How long will it take to brute-Force a 128-bit DES using the crack.sh dedicated hardware?

$\sim 1.4 * 10^{19}$ years

Over the years, many attackers challenged DES. He was defeated in several battles.



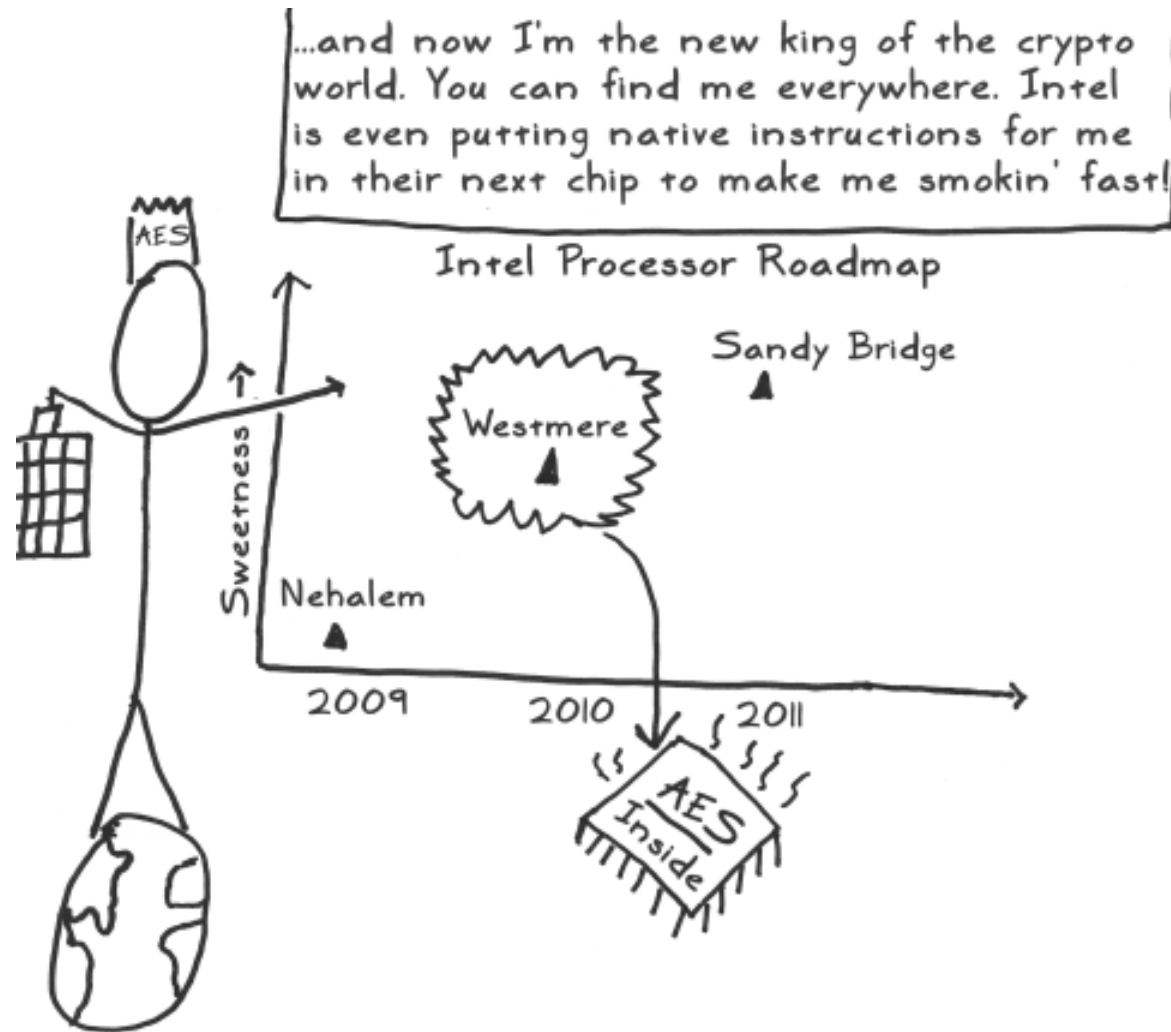
The only way to stop the attacks was to use DES 3 times in row to form 'Triple-DES.' This worked, but it was awfully slow.



- 3DES was born due to the short key length of DES (56 bits)
 - Requires that DES is performed 3 times
 - Main challenge: Very slow
- NIST gave another call for another symmetric cipher with the strength of 3DES but faster to implement even on hardware

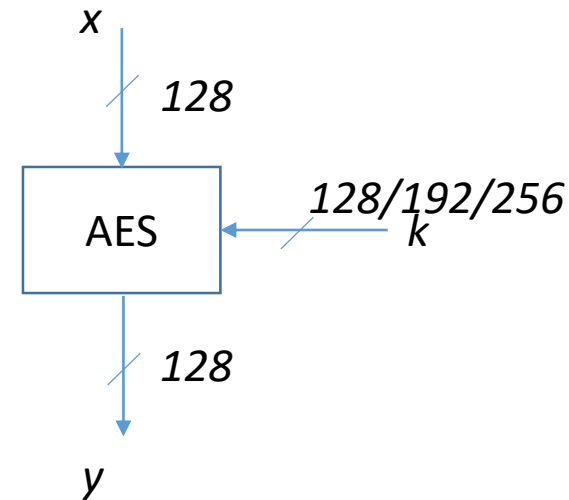
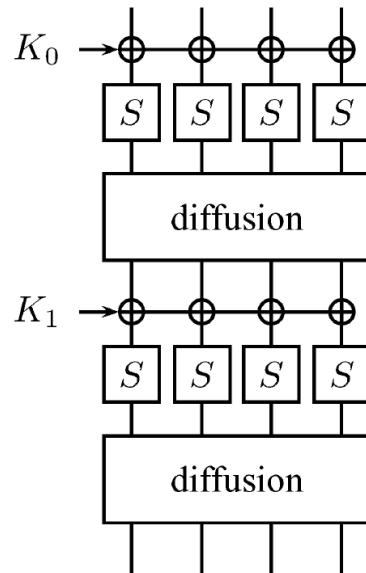


Advanced Encryption Standard (AES)

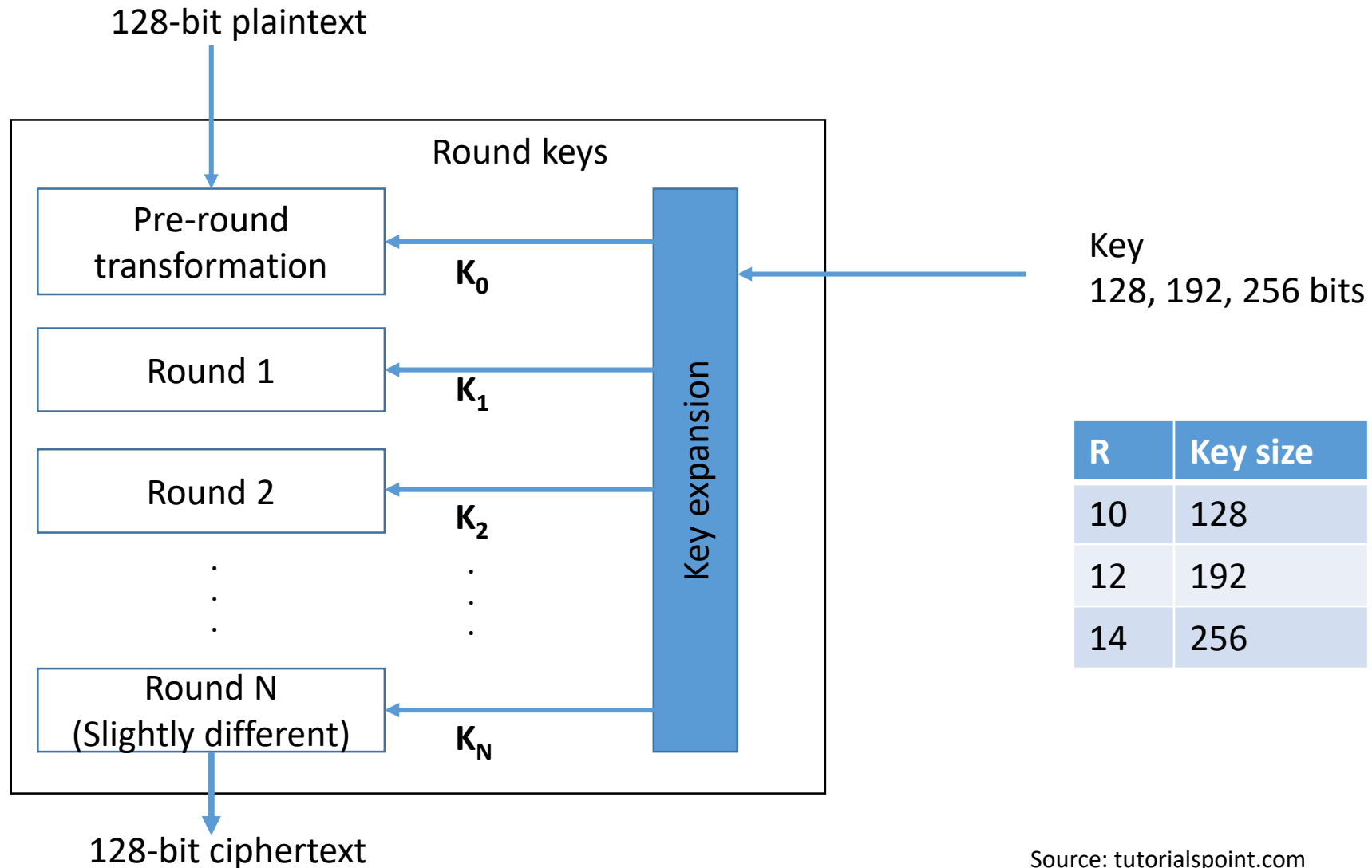


Advanced Encryption Standard (AES)

- Block cipher with 128 bit block size
- Key size varies between 128, 192 and 256 bits
- Unlike DES, AES does not use Feistel structure
- Based on substitution-permutation network

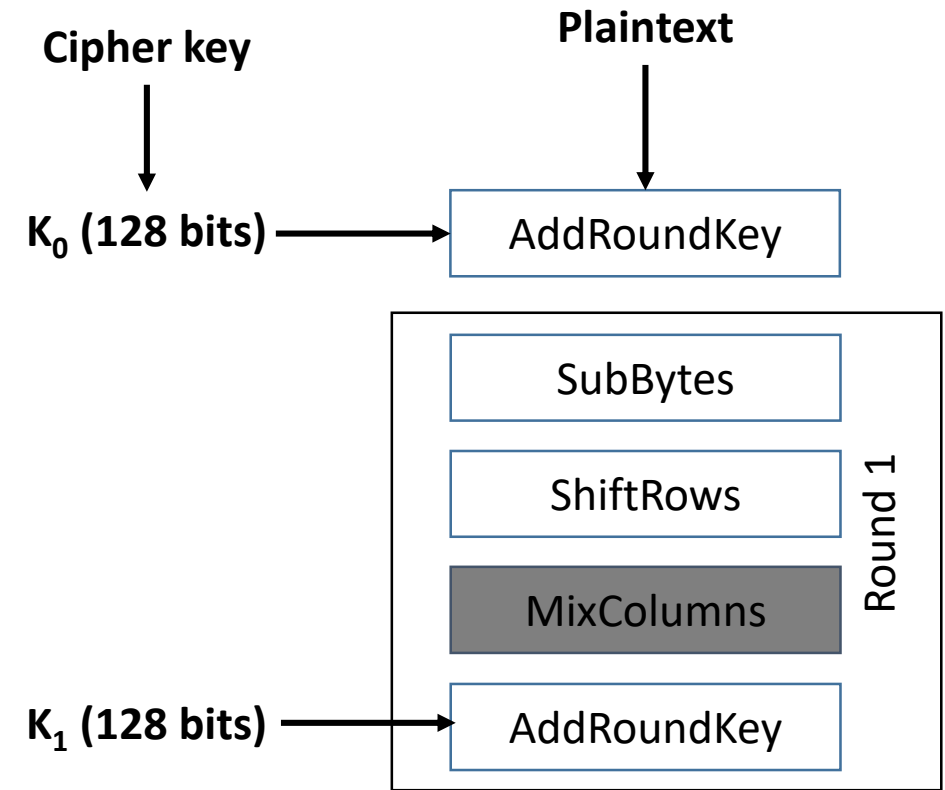


AES Description



Encryption process

- Four sub-processes
- SubBytes
 - 16 input bytes (128 bits) are substituted by looking up a fixed table (S-box) resulting in a 4 X 4 matrix
- ShiftRows
 - Each of the 4 rows of the matrix is shifted to the left. Any entry that fall off are re-inserted on the right side
- MixColumns
- AddRoundKey
 - The 16 bytes (128 bits) are XORed with the 128 bits of the round key. The output is fed back into the next round otherwise it is the ciphertext



Decryption Process

- Reverse process of encryption
- Add round keys
- Inverse Mix columns
- Inverse Shift rows
- Inverse Byte substitution

Quiz (5mins)

1. The basic elements of a sound cryptography are:
 - a) Diffusion, Key secrecy and Confusion
 - b) Encryption, Key secrecy, confusion
 - c) Encryption, decryption, key secrecy
2. In a block cipher:
 - a) Messages are encrypted at once
 - b) Messages are split into blocks before encryption
 - c) Messages are split into two before encryption
3. Symmetric encryption can provide:
 - a) Non-repudiation
 - b) Confidentiality
4. The following are challenges with symmetric encryption except:
 - a) Key distribution
 - b) Key management
 - c) Speed of execution
5. In stream ciphers:
 - a) Successive keys are used to encrypt plaintext message
 - b) The same key is used to encrypt plaintext message
6. The security of cryptography must rely on
 - a) keeping the cryptography algorithm secret
 - b) Keeping both the crypto algorithm and the key secret
 - c) Keeping only the key secret