04 Cryptography II

@Andrew: assign groups!

Engr 399/599: Hardware Security

Andrew Lukefahr

Indiana University

Adapted from: Mark Tehranipoor of University of Florida

Course Website

engr599.github.io

4111 Room Code:
2-3-5

Write that down!

Last Time: Caesar Cypher Example

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

H E L L O W O R L D ← plaintext

K H O O R . . . ← cypher text

1-grams (Unigrams) for English

a	0.080	h	0.060	n	0.070	t	0.090
b	0.015	i	0.065	o	0.080	u	0.030
c	0.030	j	0.005	p	0.020	v	0.010
d	0.040	k	0.005	q	0.002	w	0.015
e	0.130	l	0.035	r	0.065	x	0.005
f	0.020	m	0.030	s	0.060	y	0.020
g	0.015					z	0.002

[cf. Barbara Endicott-Popovsky, U. Washington]

Polyalphabetic Substitution - Examples

– Example:

	A	B	C	D	E	F	G	H	I	J	K	L	M
Key1:	a	d	g	j	m	p	s	v	y	b	e	h	k
Key2:	n	s	x	c	h	m	r	w	b	g	l	q	v

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Key1:	n	q	t	w	z	c	f	i	l	o	r	u	x
Key2:	a	f	k	p	u	z	e	j	o	t	y	d	i

– **Plaintext: TOUGH STUFF**

– **Ciphertext: ffirv zfjpm**

use n (=2) keys in turn for consecutive P chars in P

- Note:
  - Different chars mapped into the same one: **T, O → f**
  - Same char mapped into different ones: **F → p, m**
  - '**f**' most frequent in C (0.30); in English: f(**f**) = 0.02 << f(**e**) = 0.13

[cf. J. Leiwo, VU, NL]

Vigenere Tableaux (1)

Note: Row A – shift 0 (a->a)
Row B – shift 1 (a->b)
Row C – shift 2 (a->c)
…
Row Z – shift 25 (a->z)

[cf. J. Leiwo, VU, NL]

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	p
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	1
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	2
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	3
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	4
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	5
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	6
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	7
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	8
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	9
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	10
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	11
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	12
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	13
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	14
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	15
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	16
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	17
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	18
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	19
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	20
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	21
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	22
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	23
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	24
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	25

Vigenère Tableaux (2)

- Example

  Key:

  EXODUS

  Plaintext P:

  YELLOW SUBMARINE FROM YELLOW RIVER

  Extended keyword (re-applied to mimic words in P):

  YELLOW SUBMARINE FROM YELLOW RIVER

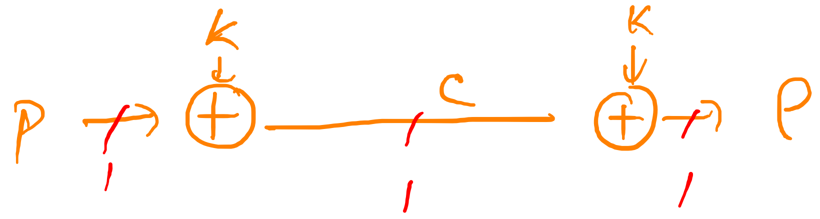
  EXODUS EXODUSEXO DUSE XODUSE XODUS

  Ciphertext:

  cbxoio wlppujmks ilgq vsofhb owyyj

[cf. J. Leiwo, VU, NL]

Stream and Block Ciphers (1)



a. Stream ciphers

b. Problems with stream ciphers

c. Block ciphers

d. Pros / cons for stream and block ciphers

Stream Ciphers (1)

- **Stream cipher**: 1 char from P → 1 char for C
  - Example: polyalphabetic cipher
    - P and K (repeated 'EXODUS'):
      YELLOWSUBMARINEFROMYELLOWRIVER
      EXODUSEXODUSEXODUSEXODUSEXODUS
    - Encryption (char after char, using Vigenère Tableaux):
      (1) E(**Y**, **E**) → **c**   (2) E(**E**, **X**) → **b**   (3) E(**L**, **O**) → **z** . . .
    - C: **cbzoiowlppujmksilgqvsofhbowyyj**
    - C as sent (in the right-to-left order):

Stream Ciphers (2)

– Example: polyalphabetic cipher - cont.

- C as received (in the right-to-left order):

Sender S → **jyywobhfosvqgliskmjupplwoiozbc** → Receiver R

- C and K for decryption:

  cbzoiowlppujmksilgqvsofhbowyyj

  EXODUSEXODUSEXODUSEXODUSEXODUS

- Decryption:

  (1) D(**c**, **E**) → **Y** (2) D(**b**, **X**) → **E** (3) D(**z**, **O**) → **L** . . .

- Decrypted P:

  YEL. . .

Q: Do you know how D uses Vigenère Tableaux?

A: Finds c under column e → Y

Problems with Stream Ciphers (1)

- Problems with stream ciphers
  - Dropping a char from key K results in wrong decryption
  - Example:
    - P and K (repeated 'EXODUS') with a char in K missing:

YELLOWSUBMARINEFROMYELLOWRIVER
EODUSEXODUSEXODUSEXODUSEXODUSE

missing X in K ! (no errors in repeated K later)

- Encryption

(using VT):

1) E(**Y**,**E**) → **c**
2) E(**E**,**O**) → **s**
3) E(**L**,**D**) → **o**

...

- Ciphertext: **cso...**

C in the order as sent (right-to-left):

...osc

Problems with Stream Ciphers (2)

- C as received (in the right-to-left order):

  ...osc →

- C and correct K ('EXODUS') for decryption:

  cso...

  EXO...

- Decryption (using VT, applying correct key):

  1) D(c, E) → Y

  2) D(s, X) → V

  3) D(o, O) → A

  ...

- Decrypted P:

  YVA... - Wrong!

  – We know it's wrong, Receiver might not know it *yet*!

What if message is corrupted in a noisy area?

Problems with Stream Ciphers (3)

- The problem might be recoverable
  - Example:

    If R had more characters decoded, R might be able to detect that S dropped a key char, and R could recover
    - E.g., suppose that R decoded:

      YELLOW SUBMAZGTR

      - R could guess, that the 2nd word should really be:

        SUBMARINE

      - => R would know that S dropped a char from K after sending "**SUBMA**"
      - => R could go back 4 chars, drop a char from K ("recalibrate K with C"), and get "resynchronized" with S

Block Ciphers (1)

- We can do better than using recovery for stream ciphers
  - Solution: use block ciphers

- Block cipher:
  1 *block* of chars from P → 1 *block* of chars for C
  - Example of block cipher: columnar transposition
  - Block size = "o(message length)" (informally)

Block Ciphers (2)

- Why block size = "o(message length)" ?
  - Because R must wait for "almost" the entire C before R can decode some characters near beginning of P
  - E.g., for P = '**HELLO WORLD**', block size is "o(10)"
  - Suppose that Key = 3 (3 columns):

```
HEL
LOW
ORL
DXX
```

  - C as sent (in the right-to-left order):

Sender S → **xlwlxroedolh** → Receiver R

Block Ciphers (3)

– C as received (in the right-to-left order): **xlwlxroedolh**

– R knows: K = 3, block size = 12 (=> 4 rows)

```
123
456
789
abc
```

a=10
b=11
c=12

=> R knows that characters wil be sent in the order:
1st-4th-7th-10th--2nd-5th-8th-11th--3rd-6th-9th-12th

– R must wait for at least:
  - 1 char of C to decode 1st char of P ('**h**')
  - 5 chars of C to decode 2nd char of P ('**he**')
  - 9 chars of C to decode 3rd, 4th, and 5th chars of P ('**hello**')
  - 10 chars of C to decode 6th, 7th, and 8th chars of P ('**hello wor**')
  - etc.

Block Ciphers (4)

- *Informally*, we might call ciphers like the above example columnar transposition cipher "weak-block" ciphers
  - R can get some (even most) but not all chars of P before entire C is received
    - R can get one char of P immediately
      - » the 1st-after 1 of C (delay of 1 - 1 = 0)
    - R can get some chars of P with "small" delay
      - » e.g., 2nd-after 5 of C (delay of 5 - 2 = 3)
    - R can get some chars of P with "large" delay
      - » e.g., 3rd-after 9 of C (delay of 9 – 3 = 6)
- There are block ciphers when R cannot even start decoding C before receiving the entire C
  - *Informally*, we might call them "strong-block" ciphers

Pros / Cons for Stream and Block Ciphers (1)

- Pros / cons for stream ciphers
  - + Low delay for decoding individual symbols
    - Can decode as soon as received
  - + Low error propagation
    - Error in $E(c_1)$ does not affect $E(c_2)$
  - - Low diffusion
    - Each char separately encoded => carries over its frequency info
  - - Susceptibility to malicious insertion / modification
    - Adversary can fabricate a new msg from pieces of broken msgs, even if he doesn't know E (just broke a few msgs)

Pros / Cons for Stream and Block Ciphers (2)

- Pros / cons for block ciphers
  - \+ High diffusion
    - Frequency of a *char* from P diffused over (a few chars of) a *block* of C
  - \+ Immune to insertion
    - Impossible to insert a char into a block without easy detection (block size would change)
    - Impossible to modify a char in a block without easy detection (if checksums are used)

Pros / Cons for Stream and Block Ciphers (3)

- Pros / cons for block ciphers — Part 2
  - \- High delay for decoding individual chars
    - See example for '`hello worldxx`' above
      - For some E can't decode even the 1st char before whole k chars of a block are received
  - \- High error propagation
    - It affects the block, not just a single char

Cryptanalysis (1)

- What cryptanalysts do when confronted with unknown?

  Four possible situations w.r.t. available info:

  1) C available
  2) Full P available
  3) Partial P available
  4) E available (or D available)

- (1) – (4) suggest 5 different approaches

Cryptanalysis (2)

- Cryptanalyst approaches

  1) Ciphertext-only attack
     - We have shown examples for such attacks
       - E.g., for Caesar’s cipher, columnar transposition cipher

  2) Known plaintext attack
     - Analyst have C and P
       - Needs to deduce E such that C=E(P), then finds D

  3) Probable plaintext attack
     - Partial decryption provides partial match to C
       - This provides more clues

Cryptanalysis (3)

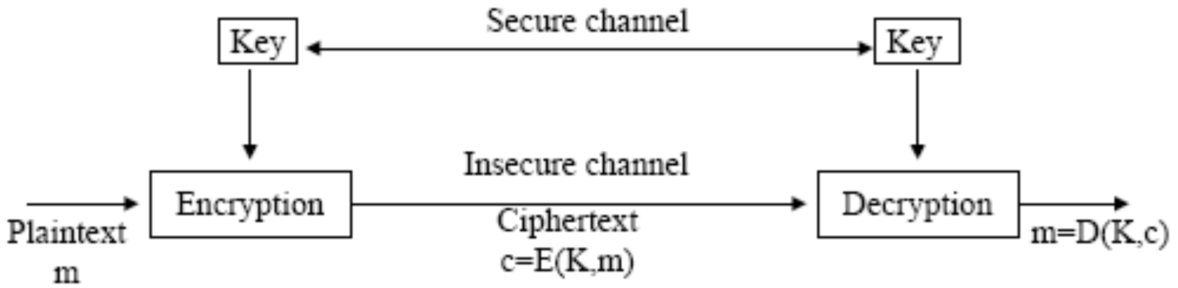
- Cryptanalyst approaches – cont.

4) Chosen plaintext attack
    - Analyst able to fabricate encrypted msgs
        - Then observe effects of msgs on adversary's actions
            - » This provides further hints

5) Chosen ciphertext attack
    - Analyst has both E and C
    - Run E for many candidate plaintexts to find P for which E(P) = C
        - Purpose: to find K_E

Symmetric and Asymmetric Cryptosystems (1)

- Symmetric encryption = secret key encryption
  - $K_E = K_D$ — called a secret key or a private key
  - Only sender S and receiver R know the key



[cf. J. Leiwo]

  - As long as the key remains secret, it also provides authentication (= proof of sender's identity)

Symmetric and Asymmetric Cryptosystems (3)

- Asymmetric encryption = public key encryption (PKE)
  - $K_E \neq K_D$ — public and private keys
- PKE systems eliminate symmetric encryption problems
  - Need no secure key distribution channel
    - => easy key distribution

Symmetric and Asymmetric Cryptosystems (4)

- One PKE approach:
  - R keeps her private key K_D
  - R can distribute the correspoding public key K_E to anybody who wants to send encrypted msgs to her
    - No need for secure channel to send K_E
    - Can even post the key on an open Web site — it is public!
  - Only private K_D can decode msgs encoded with public K_E!
    - Anybody (K_E is public) can encode
    - Only owner of K_D can decode

DES (Data Encryption Standard)

Background and History of DES (1)

- Early 1970's - NBS (Nat'l Bureau of Standards) recognized general public's need for a secure crypto system

  NBS – part of US gov't / Now: NIST – Nat'l Inst. of Stand's & Technology

  - "Encryption for the masses" [A. Striegel]
  - Existing US gov't crypto systems were not meant to be made public
    - E.g. DoD, State Dept.
  - Problems with proliferation of commercial encryption devices
    - Incompatible
    - Not extensively tested by independent body

Background and History of DES (2)

- 1972 - NBS calls for proposals for a *public* crypto system
  - Criteria:
    - Highly secure / easy to understand / publishable / available to all / adaptable to diverse app's / economical / efficient to use / able to be validated / exportable
    - In truth: Not *too* strong (for NSA, etc.)
- 1974 – IBM proposed its Lucifer
  - DES *based* on it
  - Tested by NSA (Nat'l Security Agency) and the general public
- Nov. 1976 – DES adopted as US standard for *sensitive but unclassified* data / communication
  - Later adopted by ISO (Int'l Standards Organization)
  - Official name: DEA - Data Encryption Algorithm / DEA-1 abroad

Overview of DES

@ Andrew: port DES verilog

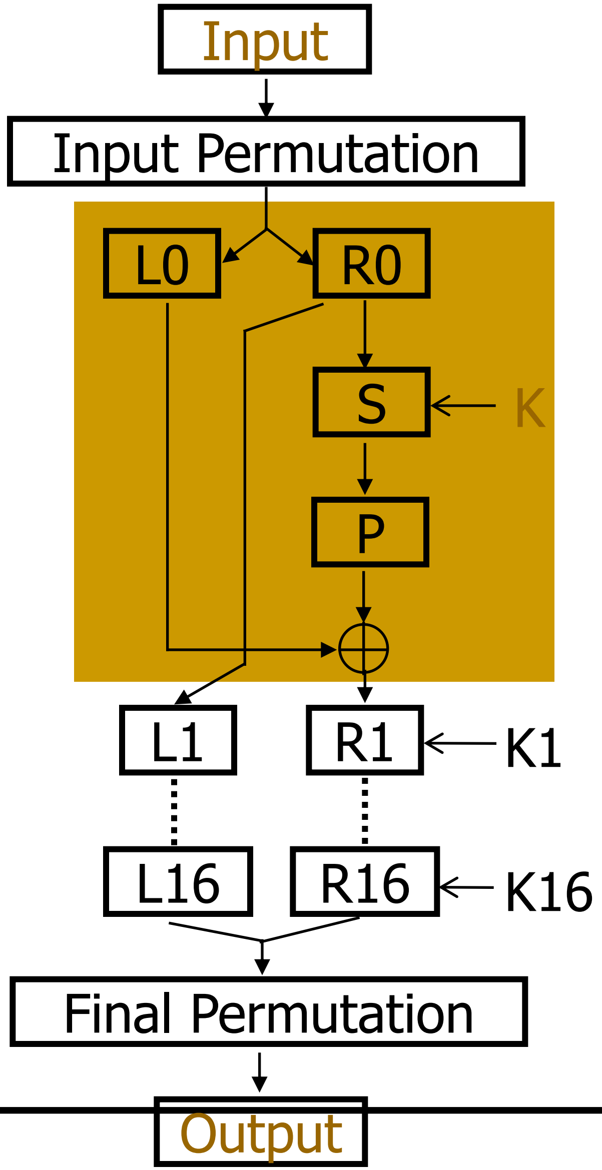
- DES - a block cipher
  - a product cipher
  - 16 rounds (iterations) on the input bits (of P)
    - substitutions (for confusion) and permutations (for diffusion)
  - Each round with a *round key*
    - Generated from the user-supplied key
- Easy to implement in S/W or H/W

- There are 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys that can be used.
- For each given message, the key can be chosen at random from among this enormous number of keys.

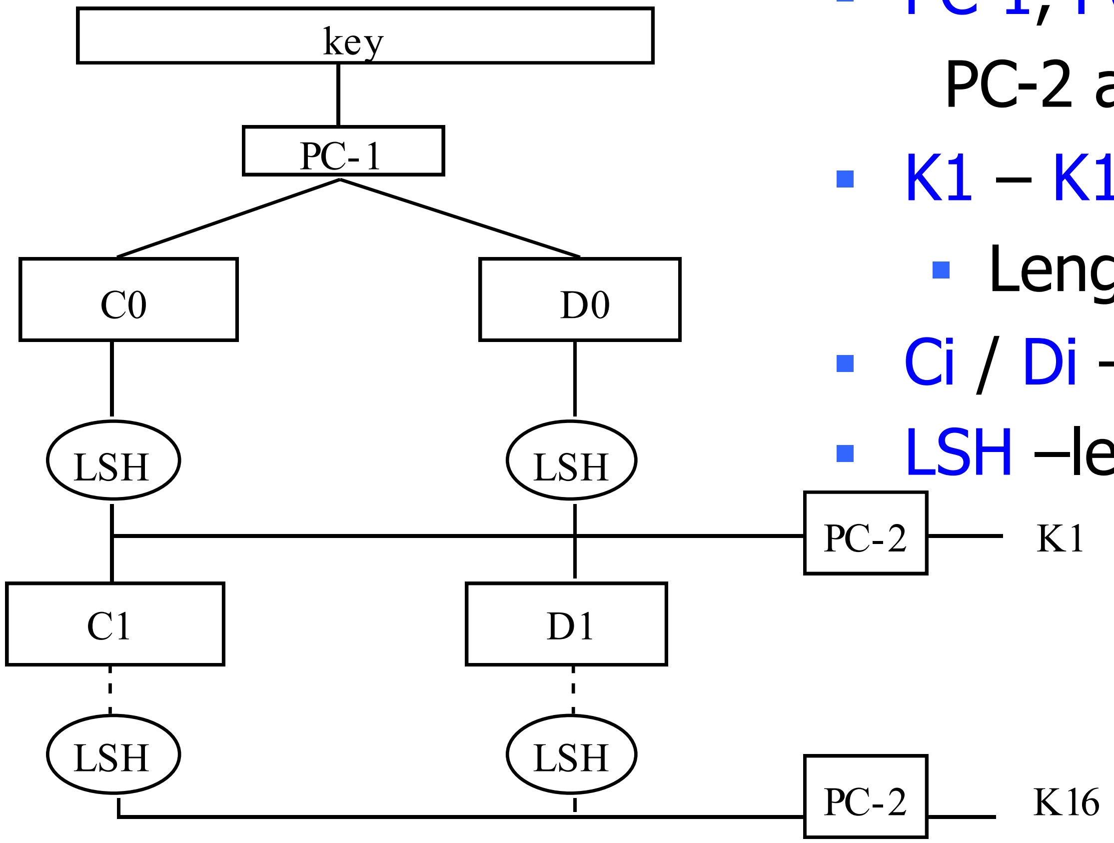
Basic Structure

[Fig. – cf. J. Leiwo]

- Input: 64 bits (a block)
- Li/Ri– left/right half of the input block for iteration i (32 bits) – subject to substitution S and permutation P
- K - user-supplied key
- Ki - round key:
  - 56 bits used +8 unused
    (unused for E but often used for error checking)
- Output: 64 bits (a block)
- Note: Ri becomes L(i+1)
- All basic op's are simple logical ops
  - Left shift / XOR



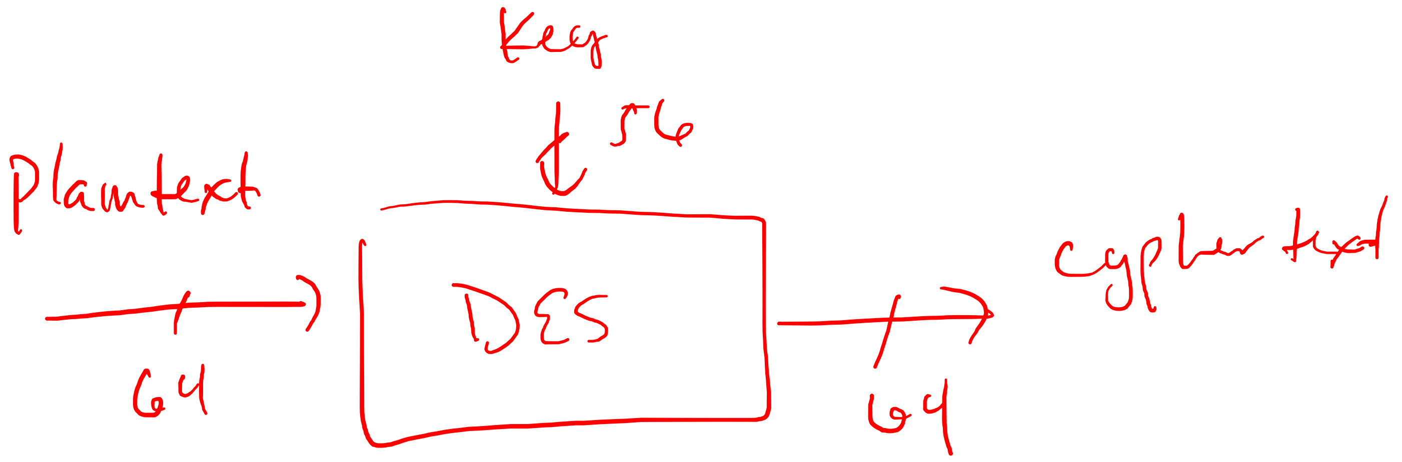
Generation of Round Keys

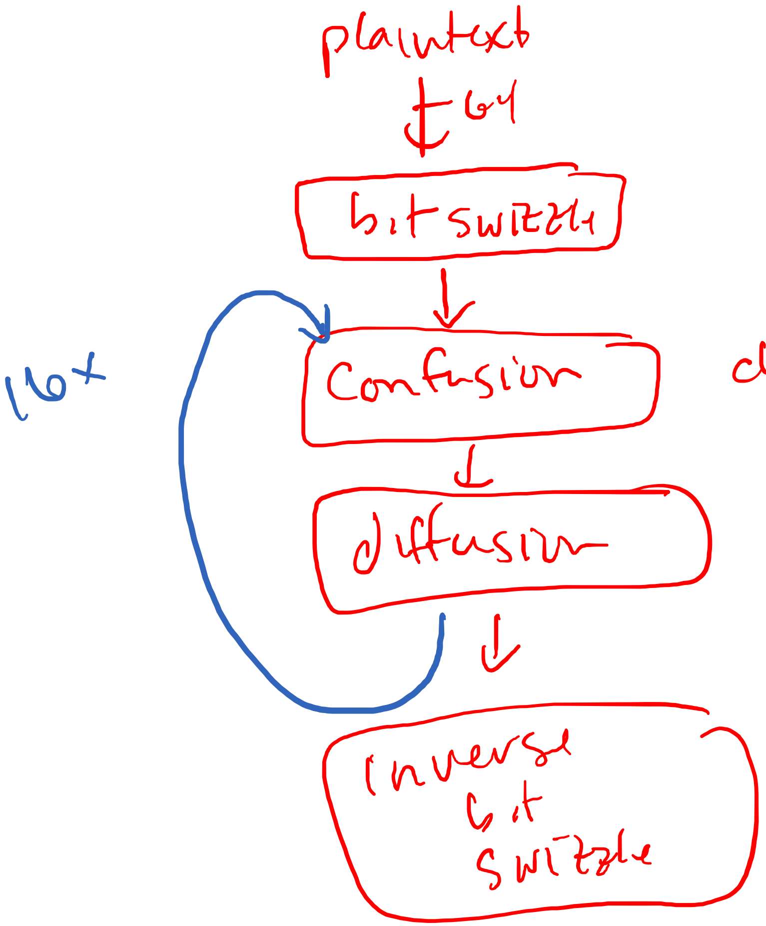


- key – user-supplied key (input)
- PC-1, PC-2 – permutation tables

  PC-2 also extracts 48 of 56 bits
- K1 – K16 – round keys (outputs)
  - Length(Ki) = 48
- Ci / Di – confusion / diffusion (?)
- LSH –left shift (rotation) tables

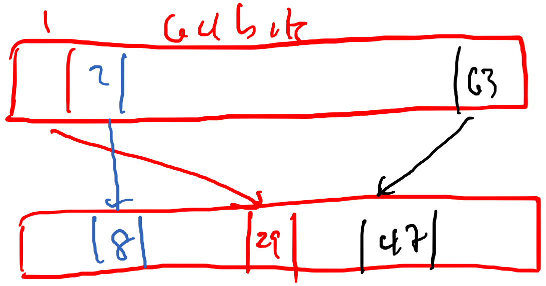
[Fig: cf. Barbara Endicott-Popovsky, U. Washington]


Key
56
Plaintext
64
DES
cypher text
64



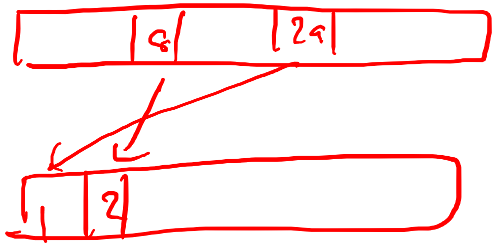
char switch table

(XOR)

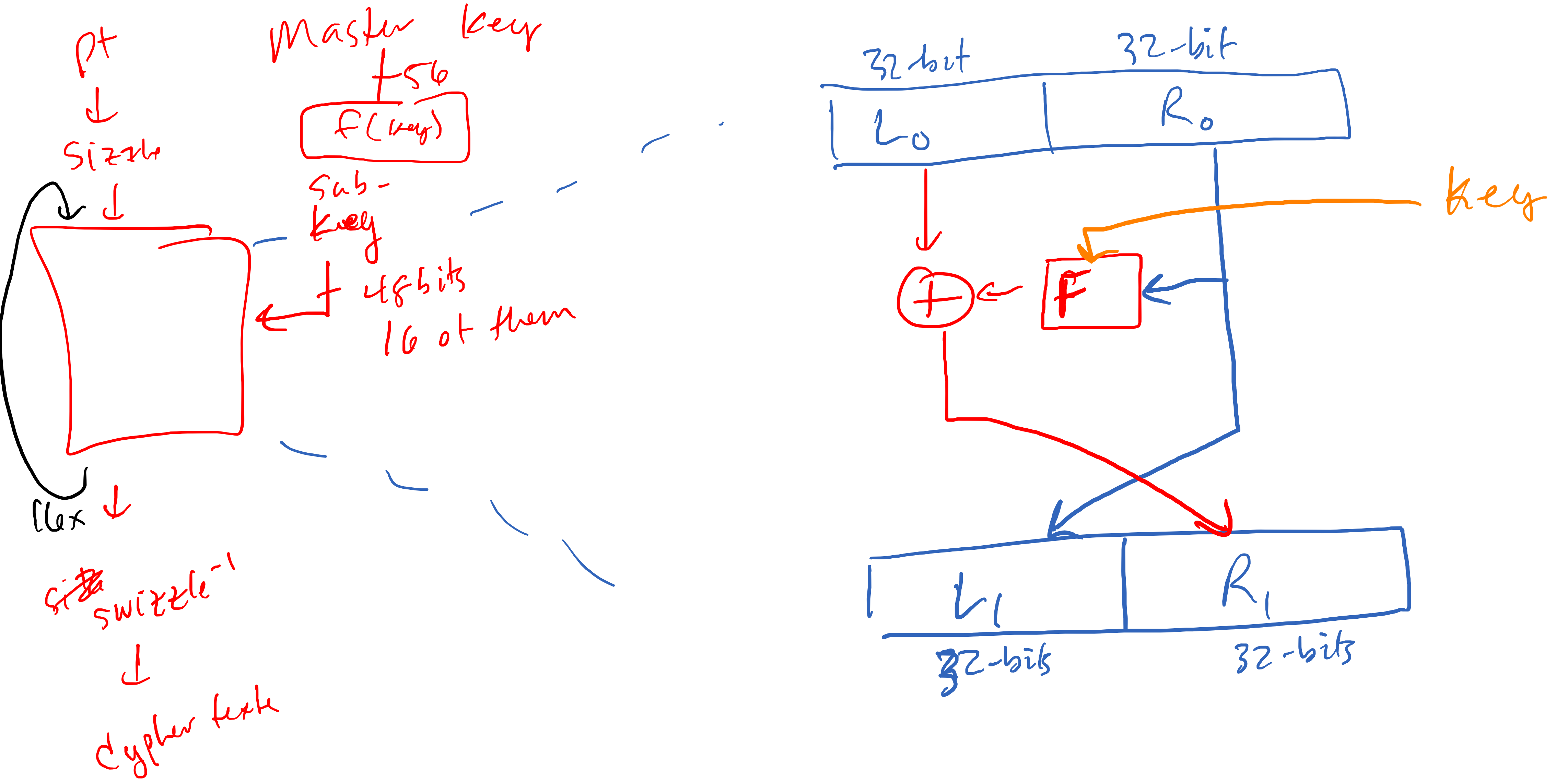


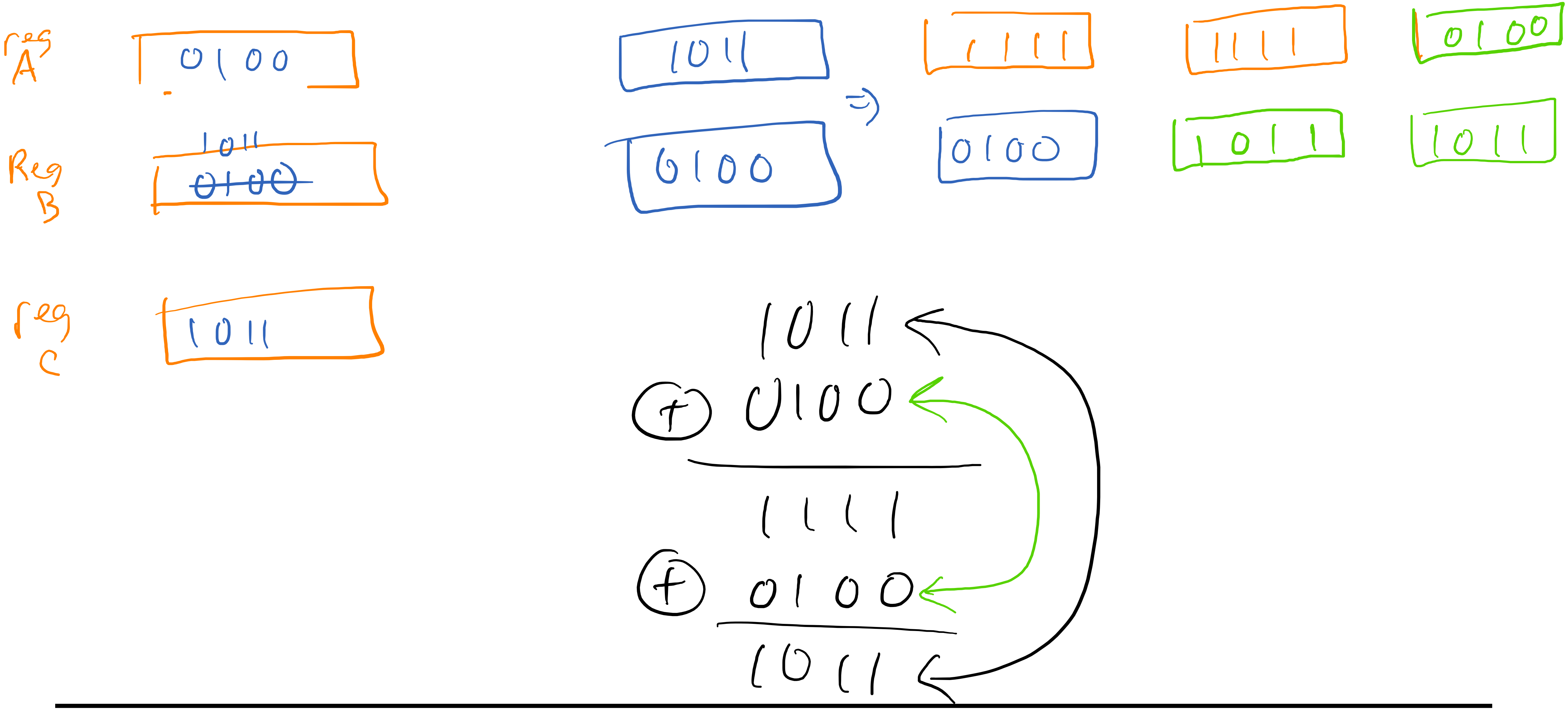
10010001 → 1010 1010

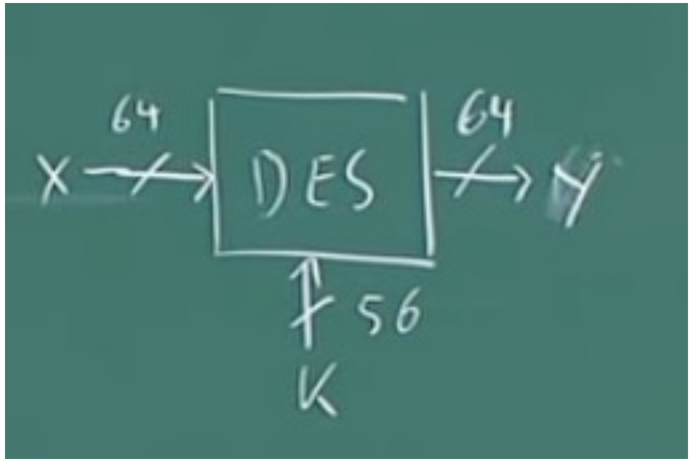
10011001 → 1100 1100



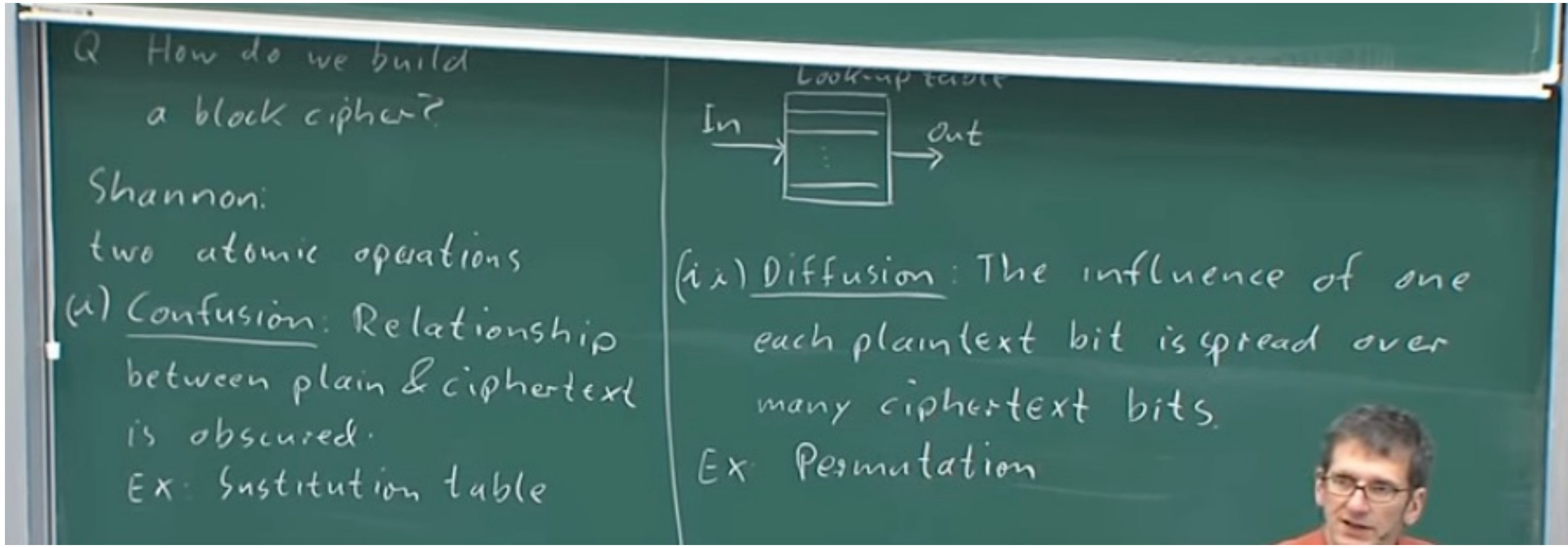
$$A \oplus B \oplus B = A$$



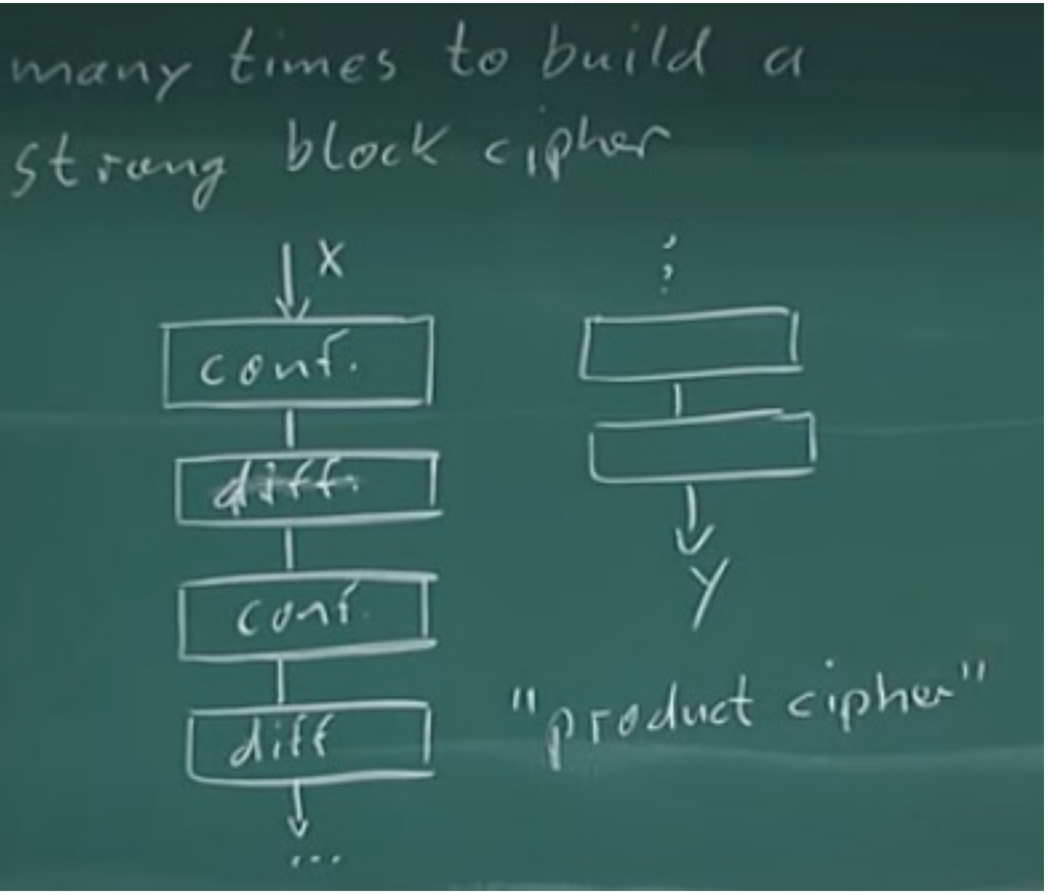

reg A
0100
Reg B
1011
0100
reg C
1011
1011
0100
1111
0100
1111
1111
0100
1011
0100
1011
1011
+ 0100
1111
+ 0100
1011

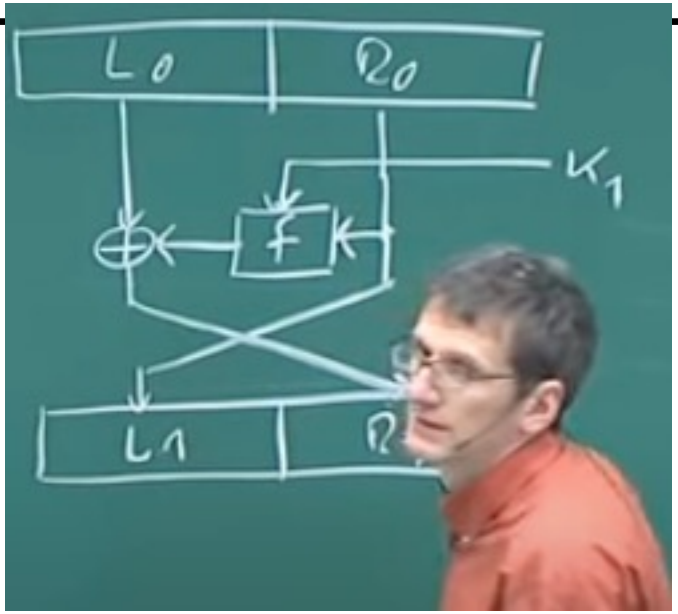


64
X
DES
64
Y
56
K



Q How do we build a block cipher?
Shannon:
two atomic operations
(i) Confusion: Relationship between plain & ciphertext is obscured.
Ex: Sustitution table
In
out
(ii) Diffusion: The influence of one each plaintext bit is spread over many ciphertext bits.
Ex Permutation


many times to build a
strong block cipher
x
conf.
diff.
conf.
diff
y
"product cipher"


L0
R0
K1
f
L1

Problems with DES

- Diffie, Hellman 1977 prediction: "In a few years, technology would allow DES to be broken in days."
- Key length is fixed (= 56)
  - 2^{56} keys ~ 10^{15} keys
  - "Becoming" too short for faster computers
    - 1997: 3,500 machines – 4 months
    - 1998: special "DES cracker" h/w – 4 days
- Design decisions not public
  - Suspected of having backdoors
    - Speculation: To facilitate government access?

Double and Triple DES

- Double DES:
  - Use double DES encryption

    C = E(k2, E(k1, P))

  - Expected to multiply difficulty of breaking the encryption
    - Not true!
      - In general, 2 encryptions are not better than one

        [Merkle, Hellman, 1981]
      - Only doubles the attacker's work

Double and Triple DES (2)

- Triple DES:
  - Is it C = E(k3, E(k2, E(k1, P)) ?
  - Not soooo simple!

Double and Triple DES (3)

- Triple DES: *Is it C=E(k3, E(k2, E(k1, P))?*
  - Tricks used:

    D not E in the 2nd step, k1 used twice (in steps 1 & 3)
  - It is:

    C = E(k1, D(k2, E(k1, P))

    and

    P = D(k1, E(k2, D(k1, C))
  - Doubles the effective key length
    - 112-bit key is quite strong
      - Even for today's computers
      - For all feasible known attacks

Security of DES

- So, is DES insecure?
- No, not yet
  - 1997 attack required a lot of cooperation
  - The 1998 special-purpose machine is still very expensive
  - Triple DES still beyond the reach of these 2 attacks
- But ...
  - In 1995, NIST (formerly NBS) began search for new strong encryption standard

Groups for Project 1

Adewale Adeniran
Annabel Brinker
Joey Brewington
Sotaro Kaneda

Joseph Bellahcen
Alec Blanton
Will Brenneke

Nicholas (Nicky) Goh
James (Dave) Ingalls
Joseph Patus

Nicole Miller
Bryan Sauter
Jack Tyndall

Owen Vogelgesang
Caleb Vrydaghs
Jason Zheng

Caleb Cook
Vishal Dung Dung
Omair Alhajeri

Project 1 Overview / Demo