

Data Encryption Standard (DES)

Abstract

After an introduction to general aspects of DES and on its history, the DES algorithm is presented, beginning with its building block, namely the Feistel cipher scheme. The modes of operation of DES and of other block encryption algorithms (ECB, CBC, CFB, OFB, CTR) are overviewed. The enhancements 2DES and 3DES to strengthen the single DES are explained, including the Meet-in-the-Middle Attack to 2DES. The application of DES in UNIX password security is presented, including the usage of salt to hinder dictionary attacks.

Outline

- **A bit of history and the basics**
 - Feistel cipher scheme and DES algorithm
 - Modes of operation
 - Double DES and Triple DES
 - Application of DES in UNIX password security

A Bit of History and Timeline

- **1973:** The National Bureau of Standards (NBS), now National Institute of Standards and Technology (NIST), issued a public request for a cipher to become a national standard
- **1974:** IBM submitted LUCIFER, based on Horst Feistel's design. The National Security Agency (NSA) worked with IBM and made some changes (S-Box changed and key shortened).
- **1977:** DES published as NBS Std. for use on all unclassified data
- **1983-1988-1993:** NBS/NIST recertified DES, in spite of concerns and practical attack demonstrations (1997, 1998: DES Cracker)
- **1999:** 3DES prescribed to overcome DES weakness
- **2002:** DES finally superseded by AES as new standard
- **2005:** DES/3DES standard officially withdrawn, but NIST approved 3DES through 2030 anyway for sensitive government information

Weakness of DES and Attacks

- Key length: 56 bit
 - ◆ $2^{56} = 7.2 \cdot 10^{16}$ keys ($3.15 \cdot 10^{16}$ ns in 1 year!)
- Trapdoor suspected for years, but never found (does not exist)
- Differential cryptanalysis (published in 1990),
 - ◆ discovered already by IBM in 1974 (but made secret by NSA)
- Linear cryptanalysis (1994)
- Distributed computing for exhaustive key search
 - ◆ DES Challenge issued by RSA Data Security won in 5 months (1997) and in 39 days (1998)
- DES Cracker: parallelized specialized computing hardware for exhaustive key search assuming a regular text is ciphered (1998)
 - ◆ average cracking time: 4.5 days (1500 chips at 40 MHz)
- 3DES extends the key size to 168 bits, but it's very slow

DES Basics

- DES is the archetypal **block cipher**
 - block size: **64 bits**
 - plain mode of operation is ECB (64-bit $P \rightarrow$ 64-bit C) but *concatenated modes of operation* are used
 - the key ostensibly consists of 64 bits, but 8 are parity bits
→ effective key length: **56 bits**
- The encryption algorithm is based on 16 cascaded processing stages (*rounds*) each based on the **Feistel scheme**
 - differential cryptanalysis was found to be useful up to 15 rounds
 - later, further differential cryptanalysis was published, which is more efficient than brute force also on 16-round DES
 - excellent *confusion* and *diffusion* properties (i.e., a strong pseudorandom permutation)

Outline

- A bit of history and the basics
- **Feistel cipher scheme and DES algorithm**
- Modes of operation
- Double DES and Triple DES
- Application of DES in UNIX password security

The Feistel Cipher Scheme

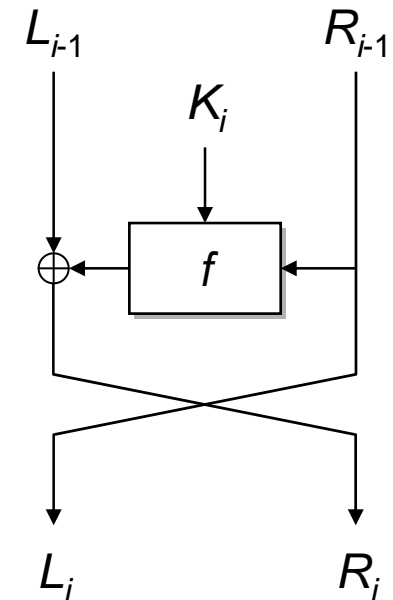
- Symmetric structure for block ciphers (Horst Feistel, IBM USA)
 - used in DES, Blowfish, RC5, and many others
 - based on swapping halves and applying a complex function $f(R_{i-1}, K_i)$
- Encryption and decryption are very similar or even identical
- Output of the i -th encryption round ($i = 1, 2, \dots, n$)

$$L_i = R_{i-1}$$

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

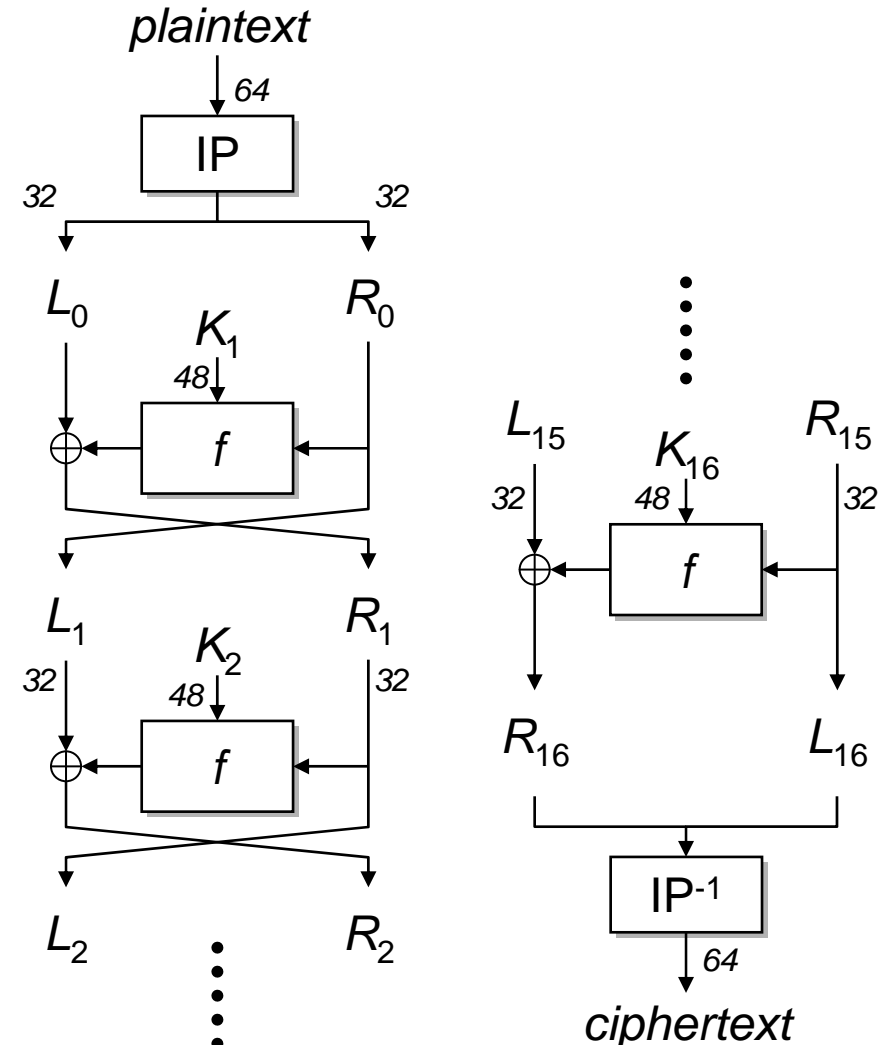
- Decryption starts from $R_n L_n$ (swapped!) and applies the same n rounds using keys in reverse order

1 st input:	R_n	L_n
1 st output:	L_n	$R_n \oplus f(L_n, K_n)$
	R_{n-1}	$L_{n-1} \oplus f(R_{n-1}, K_n) \oplus f(L_n, K_n)$
	R_{n-1}	L_{n-1}



DES Algorithm

- The *Initial Permutation* (IP) and its final inverse have no cryptographic significance
 - included to facilitate loading blocks in hardware
- $n=16$ cascaded Feistel rounds
- *Decryption* and *encryption* are identical processes with keys used in reverse order
 - may use the same hardware
 - note that the last round omits swapping
- The *F-Function* scrambles half a block with something of the key
 - highly non linear transformation
 - what achieves *confusion* and *diffusion*



DES Algorithm

Initial Permutation



- The *Initial Permutation* (IP) and its final inverse have no cryptographic significance
 - ♦ included to facilitate loading blocks in hardware
- Lookup table
 - ♦ the 58th bit of the input block becomes the 1st of the output
 - ♦ the 50th bit of the input block becomes the 2st of the output
 - ♦ ...

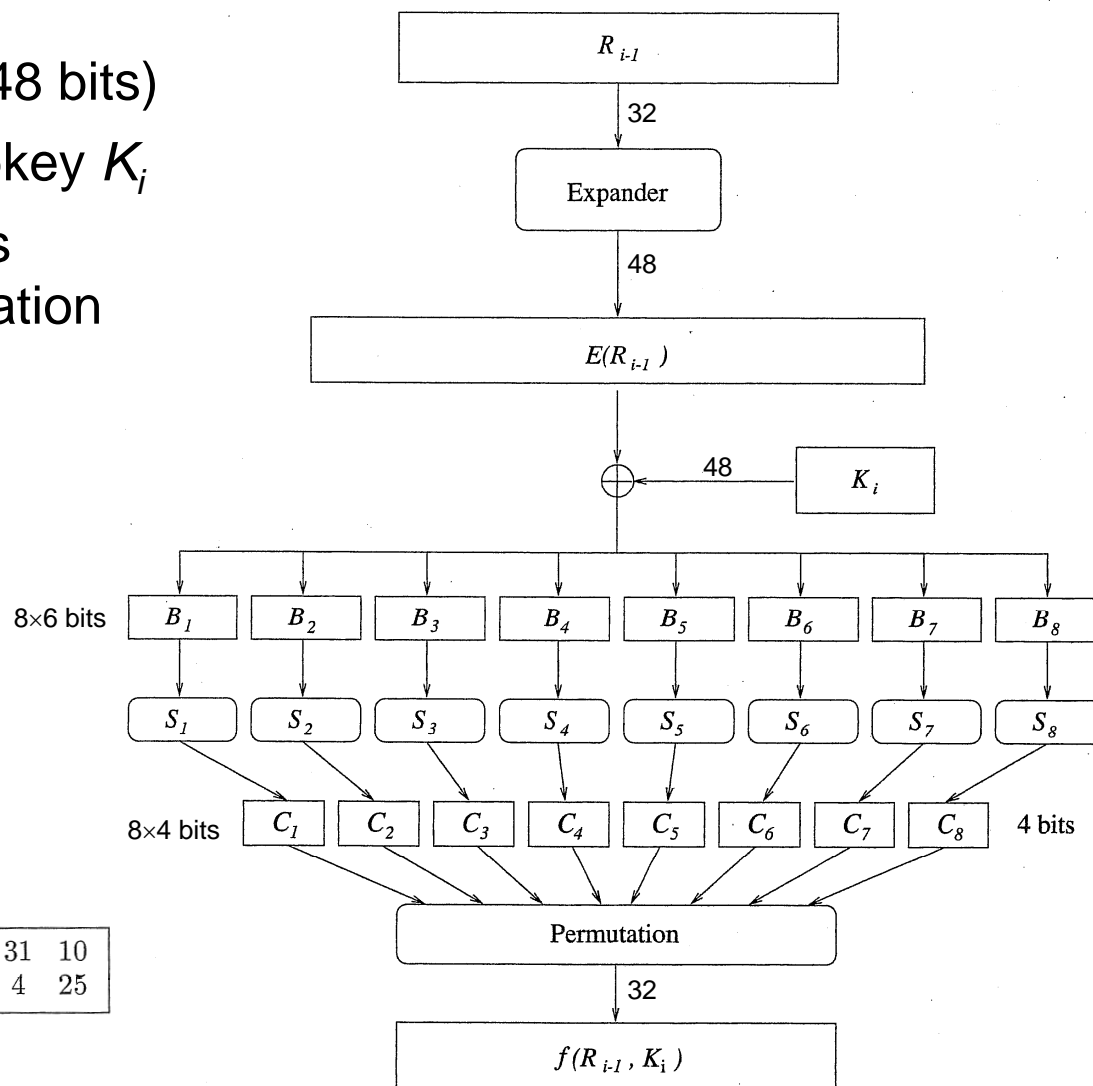
Initial Permutation															
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

The DES Function $f(R_{i-1}, K_i)$

- Expansion permutation (32→48 bits)
- Mixing with a 48-bit round subkey K_i
- Substitution with lookup tables
S-Boxes (nonlinear transformation
6→4 bits)
- Fixed permutation

Expansion Permutation											
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

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

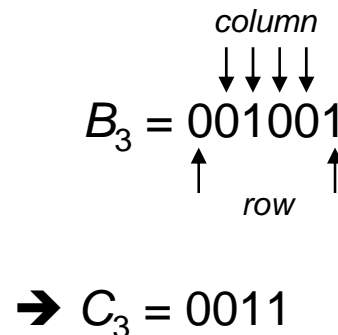


DES Algorithm

S-Boxes



- Substitution with lookup tables
6→4 bits
- Original S-Boxes by IBM replaced by NSA
- Non-linear transformation
- Concealed trapdoor suspected for years, but was never found (does not exist)



S-box 1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S-box 2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	15	4	2	11	6	7	12	0	5	14	9	
S-box 3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S-box 4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S-box 5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S-box 6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S-box 7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S-box 8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES Algorithm

Round Keys K_i



- Steps to obtain K_i ($i = 1, 2, \dots, 16$) from the master key K (56+8 bits)
 - 8 parity bits are removed → 56 bits
 - the 56 bits are permuted to get $C_0 D_0$ (28+28 bits)
 - $C_i = LS_i(C_{i-1})$, $D_i = LS_i(D_{i-1})$ based on a left-shift table
 - 48 bits are selected from $C_i D_i$ (56 bits) based on a table
 - the output is K_i

- Each bit of the key K is used in about 14 of the 16 rounds

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

Write the result as $C_0 D_0$, where C_0 and D_0 have 28 bits.

2. For $1 \leq i \leq 16$, let $C_i = LS_i(C_{i-1})$ and $D_i = LS_i(D_{i-1})$. Here LS_i means shift the input one or two places to the left, according to the following table.

Number of Key Bits Shifted per Round															
Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	1

3. 48 bits are chosen from the 56-bit string $C_i D_i$ according to the following table. The output is K_i .

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Outline

- A bit of history and the basics
- Feistel cipher scheme and DES algorithm
- **Modes of operation**
- Double DES and Triple DES
- Application of DES in UNIX password security

Modes of Operation

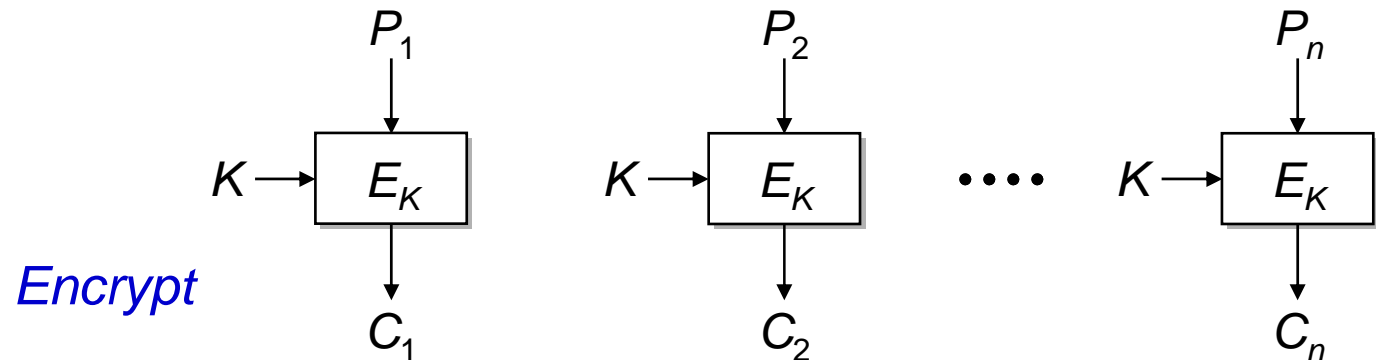
- DES is seldom used on independent 64-bit blocks, one after the other
 - ◆ Electronic Code Book (ECB)
- **Concatenated modes of operations** are common
 - ◆ Cipher Block Chaining (CBC)
 - ◆ Cipher FeedBack Mode (CFB)
 - ◆ Output FeedBack Mode (OFB)
 - ◆ CounTeR Mode (CTR)

Modes of Operation

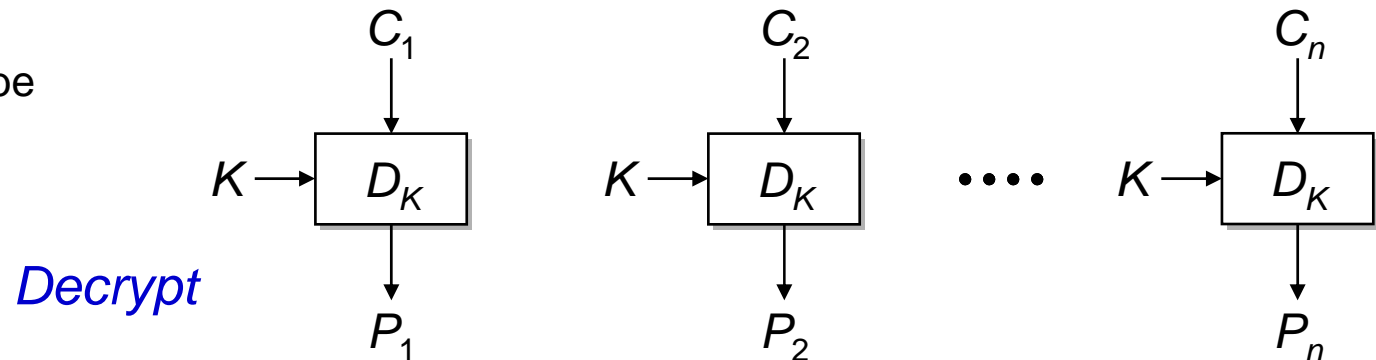
Electronic Code Book (ECB)



- $C_i = E_K(P_i)$
- $P_i = D_K(C_i)$



- Weakness
 - ◆ repeated blocks are evident
 - ◆ a *codebook* can be built over time
 - ◆ bogus blocks could be inserted



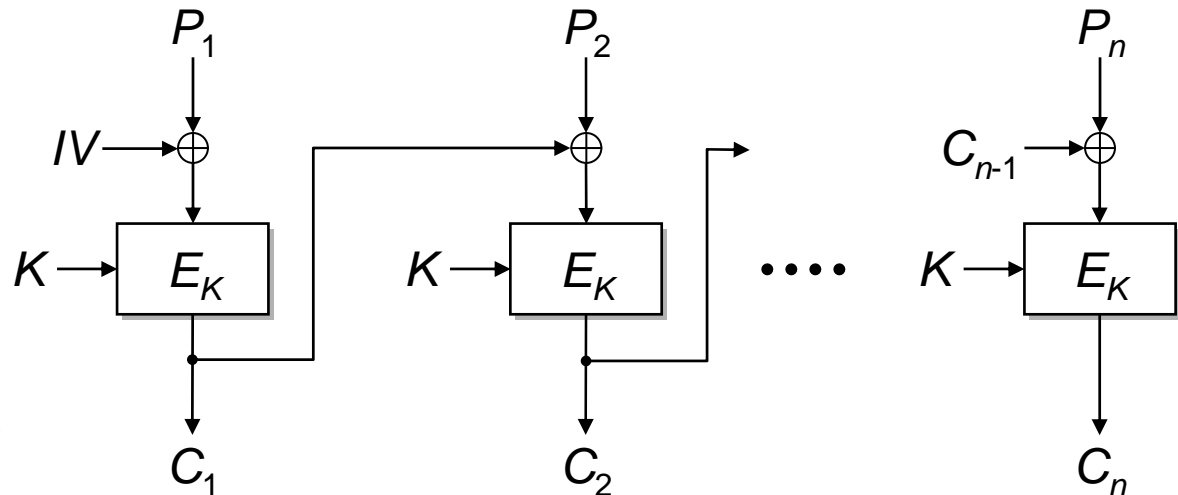
Modes of Operation

Cipher Block Chaining (CBC)



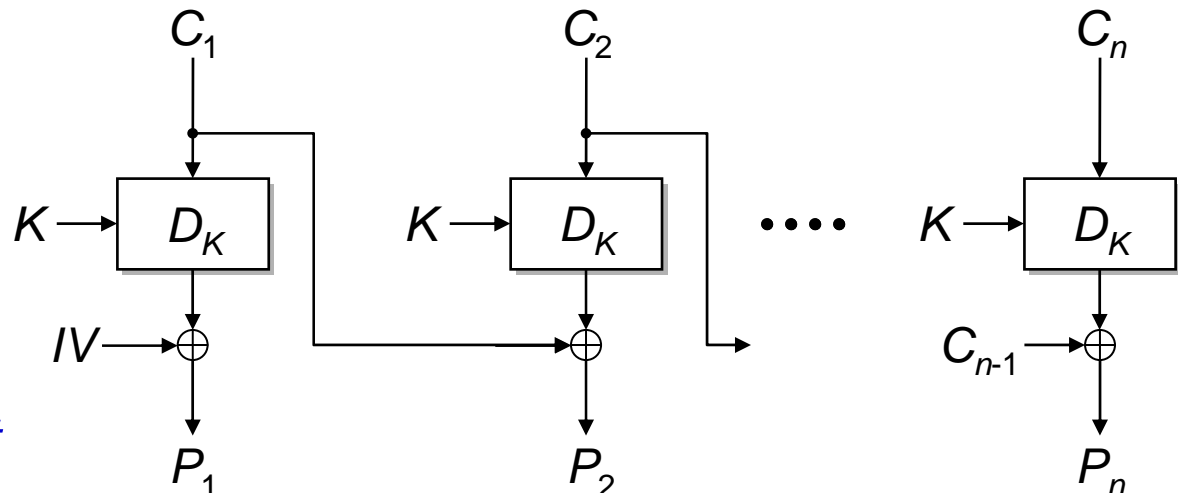
- $C_i = E_K(P_i \oplus C_{i-1})$
 $IV = C_0$
- $P_i = C_{i-1} \oplus D_K(C_i)$
 $IV = C_0$

Encrypt



- IV is random and sent in the clear
- Advantage
 - ♦ no repeated blocks

Decrypt



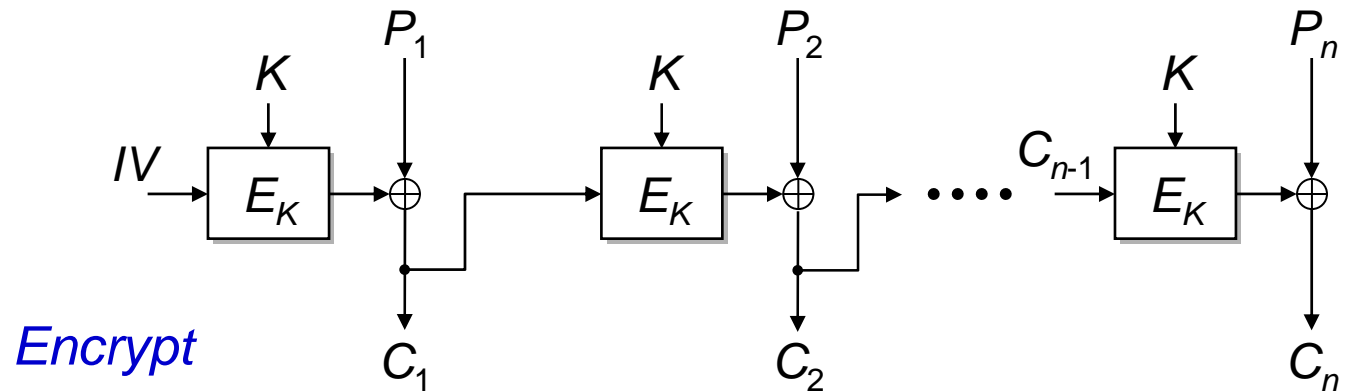
Modes of Operation

Cipher FeedBack Mode (CFB)



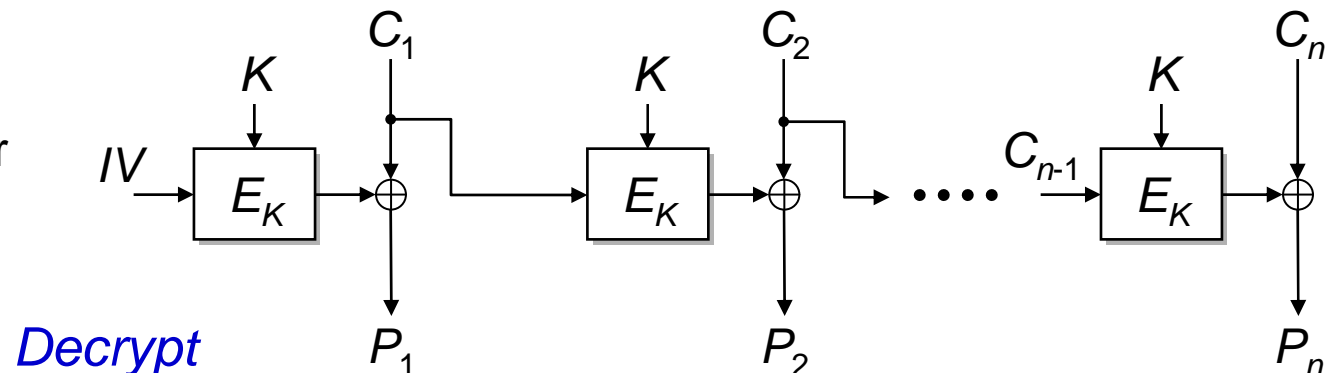
- $C_i = P_i \oplus E_K(C_{i-1})$
 $IV = C_0$

- $P_i = C_i \oplus E_K(C_{i-1})$
 $IV = C_0$



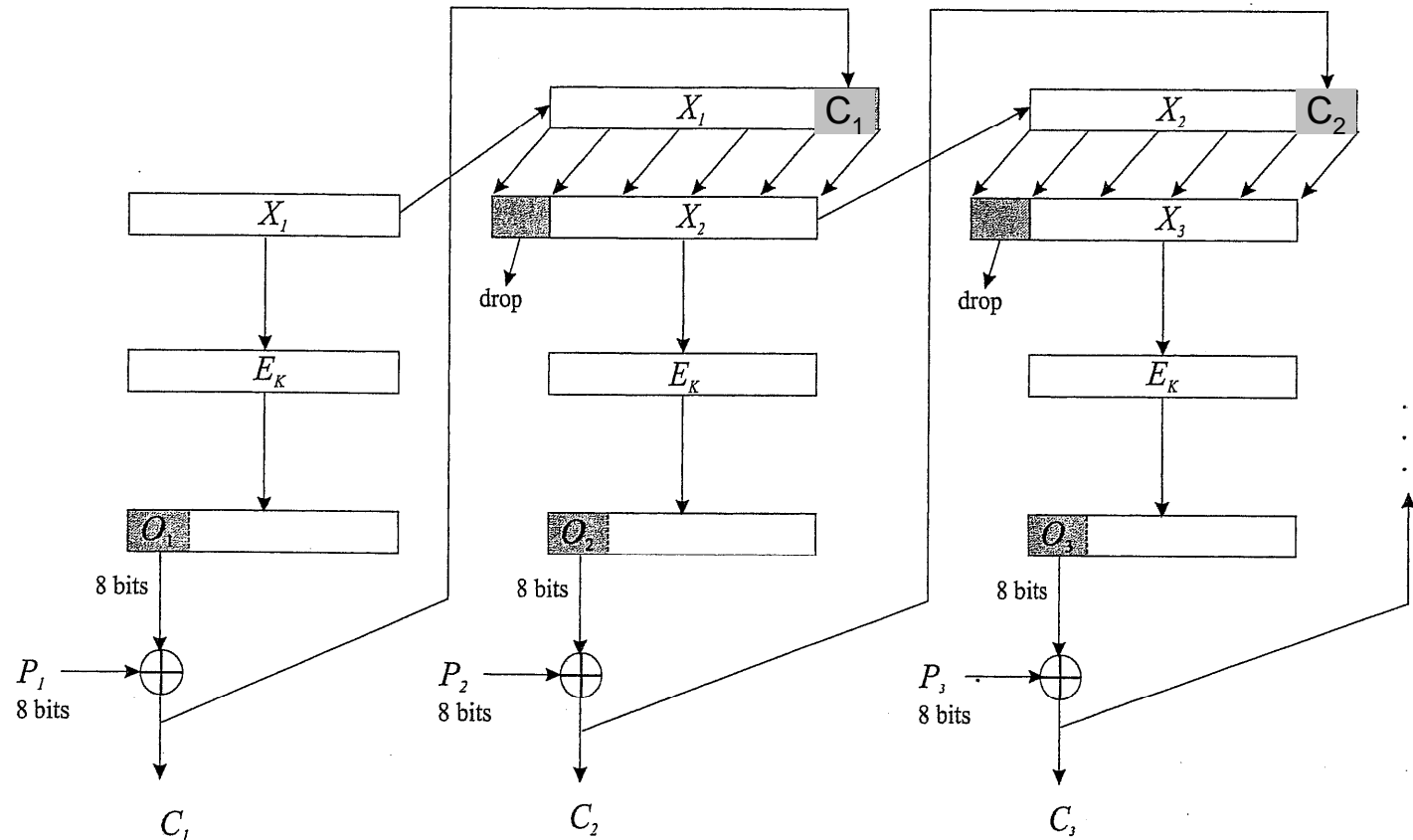
- Deciphering does not call the decrypt function D_K

- A transmission error in C_i propagates only on P_i, P_{i+1}



- $C_i = P_i \oplus E_K(C_{i-1})$
 $IV = C_0$
- $P_i = C_i \oplus E_K(C_{i-1})$
 $IV = C_0$

Encrypt

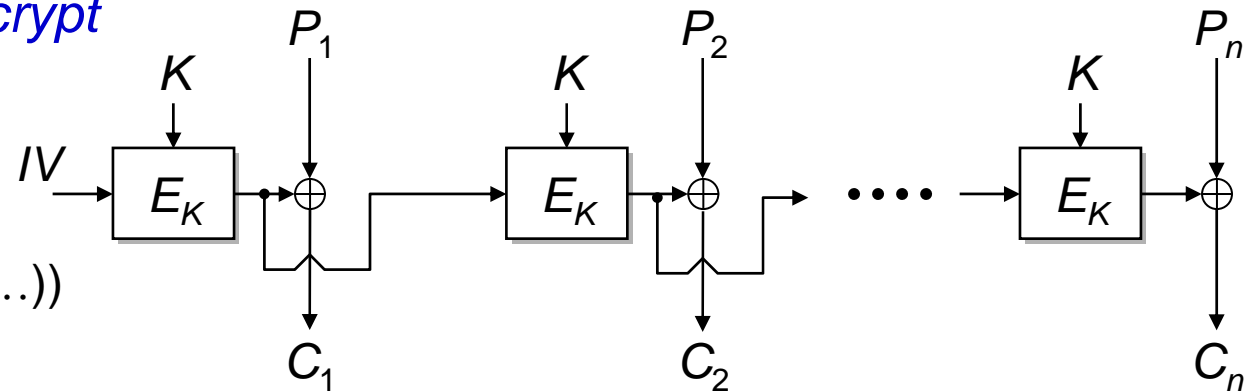


Modes of Operation

Output FeedBack Mode (OFB)



Encrypt



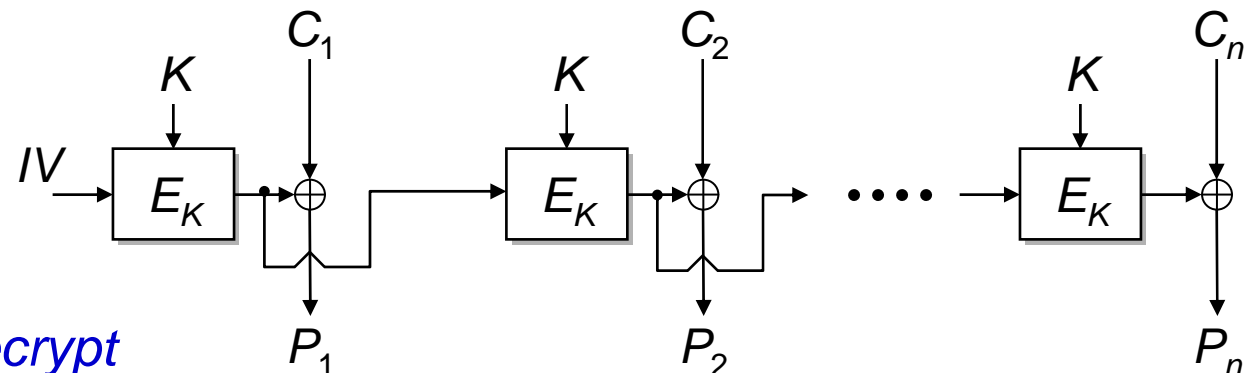
- $C_i = P_i \oplus E_K^{(i)}(IV)$
 $E_K^{(i)}(IV) = E_K(E_K(\dots E_K(IV)\dots))$

- $P_i = C_i \oplus E_K^{(i)}(IV)$
 $E_K^{(i)}(IV) = E_K(E_K(\dots E_K(IV)\dots))$

- Advantages

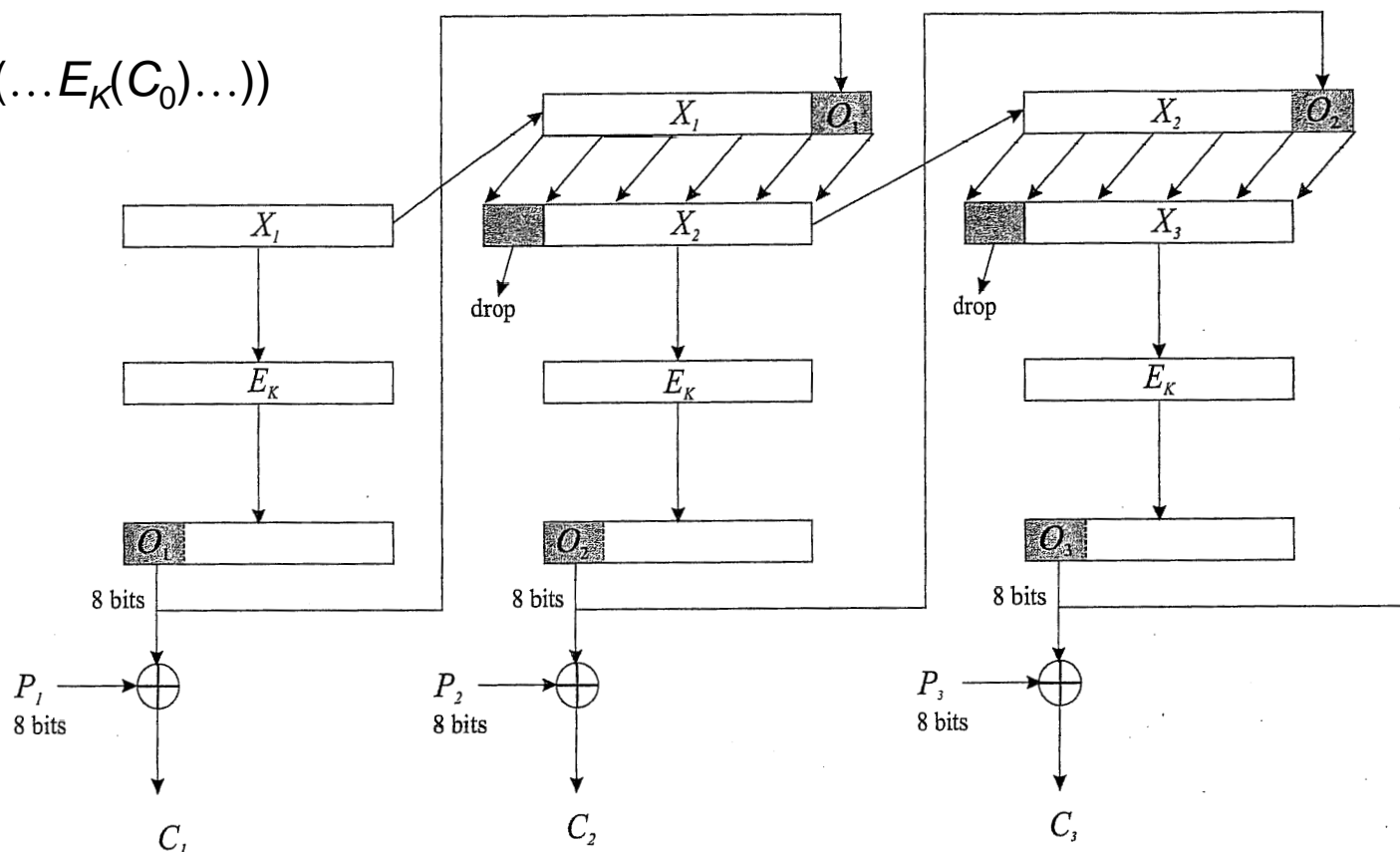
- PRBS can be computed in advance
- a transmission error in C_i does not propagate

Decrypt



- Weak to *known plaintext* attack

- $C_i = P_i \oplus E_K^{(i)}(C_0)$
 $E_K^{(i)}(C_0) = E_K(E_K(\dots E_K(C_0)\dots))$
- $P_i = C_i \oplus E_K^{(i)}(C_0)$
 $E_K^{(i)}(C_0) = E_K(E_K(\dots E_K(C_0)\dots))$



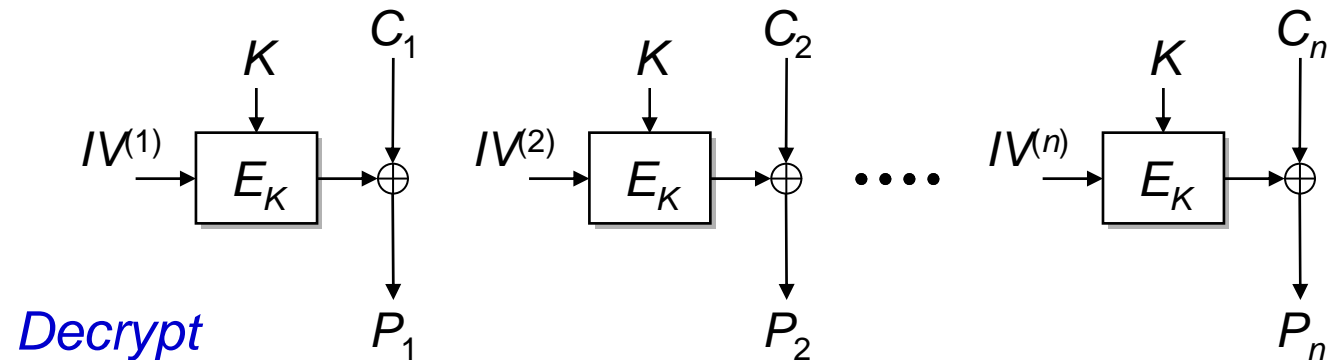
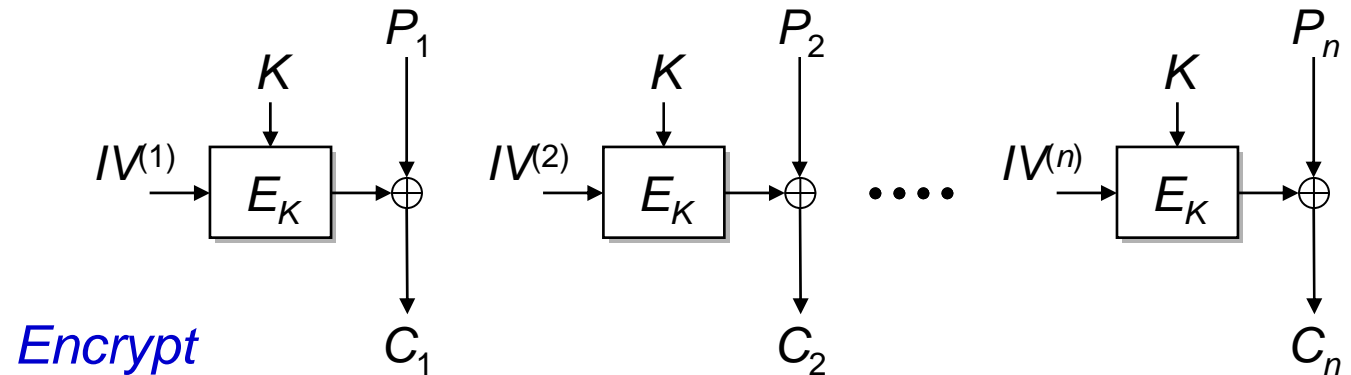
Encrypt

Modes of Operation

CounteR Mode (CTR)



- $C_i = P_i \oplus E_K(IV^{(i)})$
 $IV^{(i)} = IV^{(0)} + i$
- $P_i = C_i \oplus E_K(IV^{(i)})$
 $IV^{(i)} = IV^{(0)} + i$



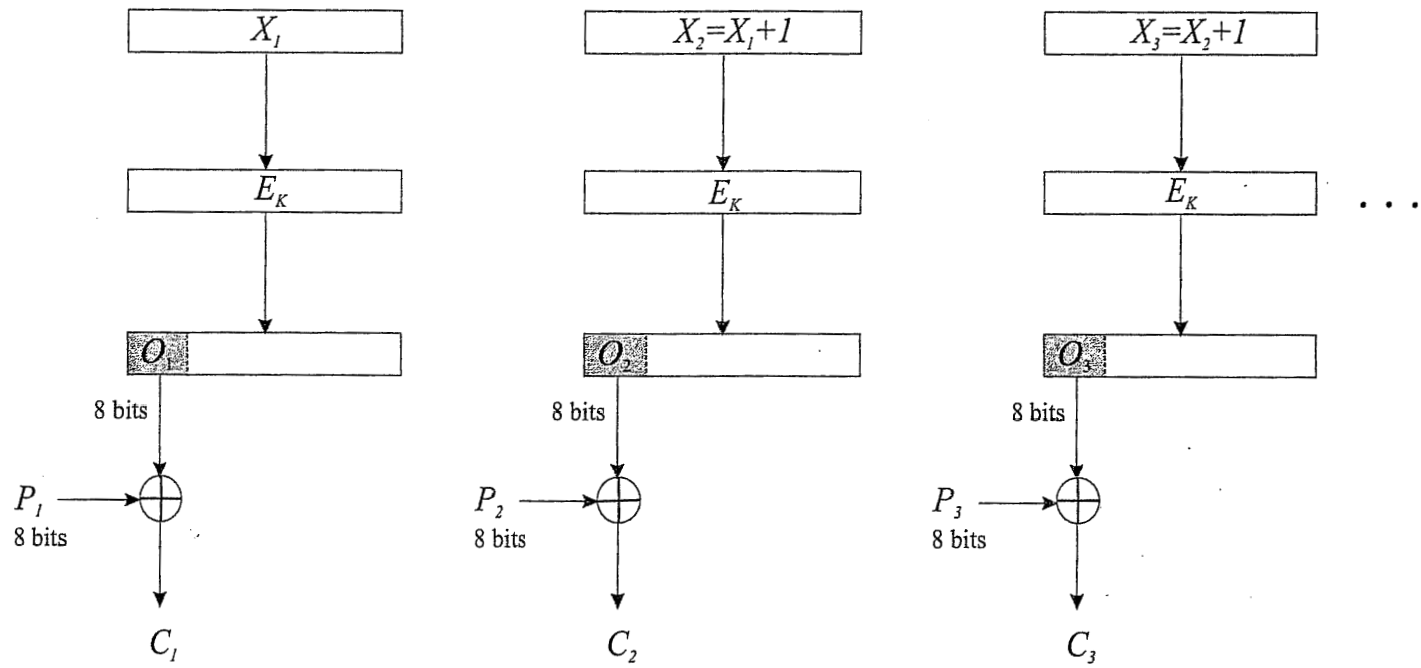
Modes of Operation

CounteR Mode (CTR): 8-Bit Mode



- $C_i = P_i \oplus E_K(IV^{(i)})$
 $IV^{(i)} = IV^{(0)} + i$
- $P_i = C_i \oplus E_K(IV^{(i)})$
 $IV^{(i)} = IV^{(0)} + i$


Encrypt



Outline

- A bit of history and the basics
- Feistel cipher scheme and DES algorithm
- Modes of operation
- **Double DES and Triple DES**
- Application of DES in UNIX password security

DES Is Not a Group: Double DES

- A possible way to increase the key length is to **double encrypt**
 - choose a double-size key $K = K_1 || K_2$ (2DES: $56+56 = 112$ bits)
 - $C = E_{K_2}(E_{K_1}(P))$
- Is there a K_3 such that $E_{K_3}(P) = E_{K_2}(E_{K_1}(P))$?
 - yes for affine and RSA ciphers \rightarrow double $E_2 E_1$ is equivalent to a single E_3
 - **NO for DES**
- **DES is not closed under composition**
DES is not a group

- 2DES brute-force cracking requires trying 2^{112} keys
- *Meet-in-the-Middle (MiM) attack* to 2DES actually reduces the key space to explore exhaustively to 2^{57} keys
- **Double DES is not secure enough**

Triple DES

- Further increase the key length by **triple encrypt**
 - triple-size key $K = K_1 || K_2 || K_3$ ($56+56+56 = 168$ bits)
- Trivial implementation of 3DES based on 3 keys K_1, K_2, K_3
 - $C = E_{K_3}(E_{K_2}(E_{K_1}(P)))$
 - the Meet-in-the-Middle attack reduces the key search space to 2^{112} keys
- Implementation of 3DES based on 2 keys K_1, K_2
 - $C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$
 - when $K_1=K_2$, this 3DES reduces to the single DES
 - using D_k instead of E_k does not change the cryptographic strength
 - resists to the Meet-in-the-Middle attack: 2^{112} keys to try
- Another strengthening of DES was proposed by Rivest (DESX)
 - choose 3 keys K_1, K_2, K_3
 - $C = K_3 \oplus E_{K_2}(K_1 \oplus P)$

Meet-in-the-Middle Attack to 2DES

- DES is not a group
 - ◆ double $E_{K_2}E_{K_1}$ is not equivalent to any single E_{K_3} for some single K_3
 - ◆ → double-size key $K = K_1 || K_2$ (112 bits)
- Meet-in-the-Middle Attack to 2DES
 - ◆ *known plaintext attack* problem:
 - P and $C = E_{K_2}(E_{K_1}(P))$ are known
 - find K_1 and K_2
 - ◆ compute and save $E_{K_1}(P)$ for all possible keys K_1 ($N=2^{56}$ values)
 - ◆ compute $D_{K_2}(C)$ for all possible keys K_2 ($N=2^{56}$ values)
 - ◆ compare each $D_{K_2}(C)$ with all $N=2^{56}$ saved values $E_{K_1}(P)$
 - ◆ there must be *at least* one correspondence
 - if more than one is found, another couple (P, C) can be examined
 - ◆ $2N$ encryptions/decryptions (+ N^2 comparisons, $N \cdot 8$ bytes storage)
 - instead of N^2 encryptions needed for brute key space search
- The key search space is reduced from 2^{112} to 2^{57}

Outline

- A bit of history and the basics
- Feistel cipher scheme and DES algorithm
- Modes of operation
- Double DES and Triple DES
- **Application of DES in UNIX password security**

Security of Stored Passwords

- User authentication for system access
 - ◆ the system prompts for a login id and password
 - ◆ if credentials entered correspond to credentials saved, access is granted
- Security of credentials storage
 - ◆ credentials must not be stored as plaintext in a data base
 - ◆ otherwise, if the system is compromised, an attacker can impersonate anyone whose credentials have been stolen
 - ◆ → a **digest** (*hash*) $y = f(x)$ of passwords x is stored
 - ◆ $y = f(x)$ must be a **one-way function**
 - for a given y , it is unfeasible to compute any x for which $f(x) = y$
- Example: UNIX systems
 - ◆ password digests are saved in a file (readable only by root)
 - ◆ the password digest is computed by a *slow* one-way function $f(x)$
 - ◆ anyone who has the file cannot obtain the passwords x

Dictionary Attack to the Password File

- Dictionary attack
 - ◆ all words in a **dictionary** (adding slight variations, combinations, enhancements with numbers) are fed into $f(x)$
 - ◆ resulting digests are searched in the password file (assumed public)
 - ◆ the passwords of many users will be found
- Possible ways to prevent dictionary attacks
 - ◆ making the one-way digest function $f(x)$ *slow*
 - ◆ using a random **salt** to add variety to plain hashing and compute $f(x,s)$

Salting in UNIX: Traditional DES-Based Scheme

- The dictionary attack on the UNIX file is hindered by using a **salt**
 - ◆ each password x is padded with random additional 12 bits s (*salt*)
 - ◆ each salt s is stored in clear with the user name and the $f(x,s)$ digest
 - ◆ $f(x)$ (`crypt()`) is based on DES, but the salt s is used to perturb the standard DES algorithm, to make impossible using regular DES chips
 - the user's password is truncated to 8 characters, and those are coerced down to 7 bits each, forming a 56-bit DES key
 - that key is used to DES encrypt an all-0s 64-bit block, and then encrypt it again with the same key for a total of 25 DES encryptions
 - the 12-bit salt is used to perturb the encryption algorithm, so standard DES implementations can't be used to implement `crypt()`
 - the salt and the final ciphertext are base64-encoded into a printable string
 - ◆ recovering the password x implies solving a *known plaintext attack*
- User authentication
 - ◆ the entered password x is completed with the saved salt s and the digest $f(x,s)$ is compared to the stored digest

Salting Hinders Dictionary Attacks

- The dictionary attack targets *random* users, not a *specific* user
- Salting hinders attacks to random users in the file, not to specific ones
 - ◆ if I want to find the password x of Bob, I must compute $f(x, S)$ for all dictionary words padded with the salt S saved for Bob
 - ◆ if I want to try all dictionary words, hoping that some resulting digest matches with values stored in the file for some users, then I must compute $f(x, s)$ for all possible 2^{12} values of salt for each word x
- Salting has been not considered sufficient to block attacks
- Newer versions of UNIX crypt() extended its strength
 - ◆ longer salt (24 bits)
 - ◆ more than 8 characters used from the user's password
 - ◆ slower functions, based on various encryption algorithms (MD5, Blowfish, SHA, etc.)