Computer Security lecture notes

Symmetric-key cryptography

In symmetric-key cryptography, we encode our plain text by mangling it with a secret key. Decryption requires knowledge of the same key, and reverses the mangling.

ciphertext = encrypt(plaintext, key)
plaintext = decrypt(ciphertext, key)

Symmetric key cryptography is useful if you want to encrypt files on your computer, and you intend to decrypt them yourself. It is less useful if you intend to send them to someone else to be decrypted, because in that case you have a "key distribution problem".) In security, we assume the encryption *algorithms* that we have chosen to use are publically known; only the key is secret to the participants. Slogan: "*obscurity is no security*".

Caesar cipher

The key is a number between 1 and 25. Define code('a')=0, code('b')=1, ..., code('z')=25.

$$\text{encryption}(c, \text{key}) = \text{code}^{-1}(\text{code}(c)+\text{key} \bmod 26)$$

Pros: simple.
Cons: crap.

Compression-then-substitution

Compress the text first (in an attempt to avoid the *frequency-of-letters* attack), and then do a substitution such as: 'a' goes to 'x', '¶' goes to 'Æ', etc. This map will be the key.
Pros: still simple, but much better.
Cons: may be some regularity in compressor output (header info, etc) which would aid cryptanalysis.

Data Encryption Standard (DES)

The Data Encryption Standard (DES) was a standard for encryption from 1976 to about 2000, and has been widely used during (and since) that period internationally. (Since 2001, it has been superceded as a standard by AES; see later.) The DES algorithm has been intensely studied and has motivated the modern understanding of block ciphers and their cryptanalysis.

Symmetric key encryption: simplified DES

S-DES is a simplified version of DES algorithm (see [2]). It closely resembles the real thing, but it has smaller parameters, to facilitate operation by hand for pedagogical purposes. It was designed by Edward Schaefer as a teaching tool to understand DES. S-DES (and DES) are examples of a block cipher: the plain text is split into blocks of a certain size, in this case 8 bits.

$$\text{plaintext} = b_1b_2b_3b_4b_5b_6b_7b_8$$
$$\text{key} = k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$$

Subkey generation

First, produce two subkeys K_1 and K_2:

$$K_1 = P8(LS_1(P10(key)))$$
$$K_2 = P8(LS_2(LS1(P10(key))))$$

where $P10(k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}) = k_3k_5k_2k_7k_4k_{10}k_1k_9k_8k_6$.

It's convenient to write such *bit substitution operators* in this notation:

P10	3	5	2	7	4	10	1	9	8	6	10 bits to 10 bits
P8	6	3	7	4	8	5	10	9			10 bits to 8 bits
LS_1 ("left shift 1 bit" on 5 bit words)	2	3	4	5	1	7	8	9	10	6	10 bits to 10 bits
LS_2 ("left shift 2 bit" on 5 bit words)	3	4	5	1	2	8	9	10	6	7	10 bits to 10 bits

Encryption

The plain text is split into 8-bit blocks; each block is encrypted separately. Given a plaintext block, the cipher text is defined using the two subkeys K_1 and K_2, as follows:

$$\text{ciphertext} = IP^{-1}(\ f_{K_2}(\ SW(\ f_{K_1}(\ IP(\ \text{plaintext}\ )\ )\ )\ )\ )$$

where:

IP ("initial permutation")	2	6	3	1	4	8	5	7	8 bits to 8 bits
IP^{-1}	4	1	3	5	7	2	8	6	8 bits to 8 bits
SW ("switch")	5	6	7	8	1	2	3	4	8 bits to 8 bits

and $f_K(\ )$ is computed as follows. We write exclusive-or (XOR) as +.

$f_K(\ L, R\ ) = (\ L + F_K(R)\ ,\ R\ )$

$F_K(R) = P4\ (\ \ S0(\ lhs(\ EP(R)+K\ ))\ \ ,\ \ \ S1(\ rhs(EP(R)+K\ ))\ \ )$

EP ("expansion/permutation")	4	1	2	3	2	3	4	1	4 bits to 8 bits
P4	2	4	3	1					4 bits to 4 bits
lhs	1	2	3	4					8 bits to 4 bits
rhs	5	6	7	8					8 bits to 4 bits

$S0(b_1b_2b_3b_4)$ = the [b_1b_4 , b_2b_3] cell from the "S-box" S0 below, and similarly for S1.

S0

	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	0	3

S1

	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

Decryption

Decryption is a similar process.

$$\text{plaintext} = IP^{-1}(f_{K_1}(SW(f_{K_2}(IP(\text{ciphertext})))))$$

Diagrams showing operation

These diagrams are reproduced from William Stallings' excellent book, *Cryptography and Network Security* [2]. These diagrams are copyright 2003 by Pearson Education Inc., and are not covered by the Gnu Free Documentation License which covers the other parts of this document.

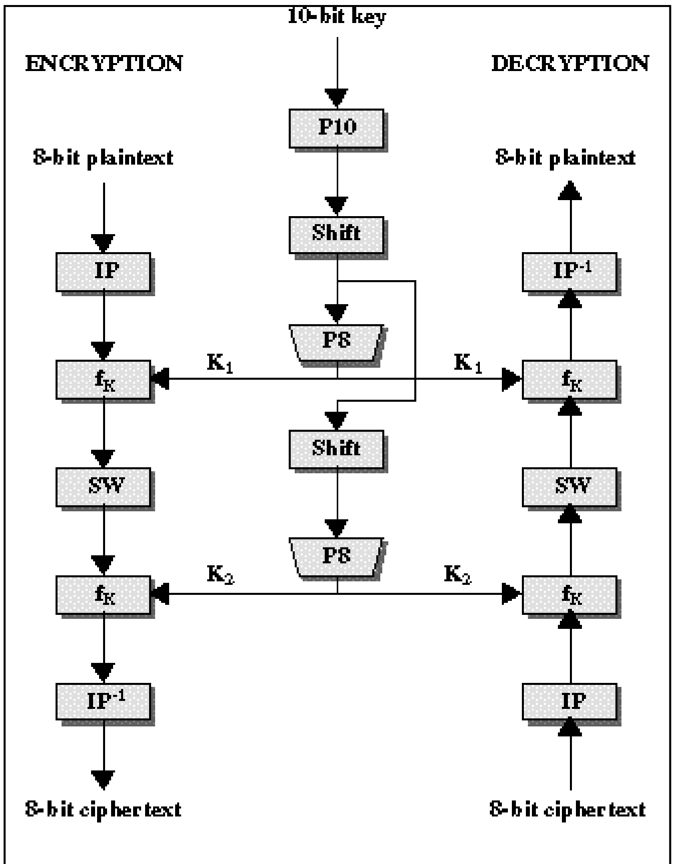


Figure 3.1 Simplified DES Scheme

Relation with DES

SDES is a simplification of a real algorithm. DES operates on 64 bit blocks, and uses a key of 56 bits, from which sixteen 48-bit subkeys are generated. There is an initial permutation (IP) of 56 bits followed by a sequence of shifts and permutations of 48 bits. F acts on 32 bits.

$$\text{ciphertext} = IP^{-1}(f_{K_{16}}(SW(f_{K_{15}}(. . . (SW(f_{K_1}(IP(\text{plaintext})))). . .))))$$

Analysis of SDES

The key question about the design of DES and SDES is: why? What motivated the design choices? The munging of the bits is so complicated that it appears to be impossible to analyse systematically. For example, a known-plaintext attack (in which we attempt to calculate a key, given ciphertext and plaintext) involves solving 8 nonlinear equations in 10 unknowns in the case of SDES, which is hard, and many more equations in more unknowns for full DES. But the small key length for SDES makes it vulnerable to a brute force attack. There are only 1024 keys.

Analysis of DES

For some years, a brute force attack was thought to be the most promising approach to attacking DES. This is much harder than in the case of SDES; there are 7×10^{16} keys. This would take over 2000 years if you checked each key in one microsecond. Michael Wiener designed a DES cracker with millions of specialised chips on specialised boards and racks [1, p.153]. He concluded in 1995 that for $1 million, a machine could be built that would crack a 56-bit DES key in 7 hours. Schneier estimates that this would be doable for $100,000 in 2000, so we might extrapolate to $10,000 in 2005. For these reasons, algorithms based on 56-bit keys, like DES, are no longer thought secure. Schneier: "insist on at least 112 bit keys."

Cracking DES "This book describes a machine which we actually built to crack DES. ... We have donated our design to the public domain, so it is not proprietary. ... We have published its details so that other scientists and engineers can review, reproduce, and build on our work. There can be no more doubt. DES is not secure." Basic statistics: 1998, < $250,000, cracked a key in 3 days.

More recently, there has been cryptanalysis against DES with some success. The best analytical attack is linear cryptanalysis which can reduce the brute force search to about 2^{43} operations.

Animation of the key schedule part of SDES

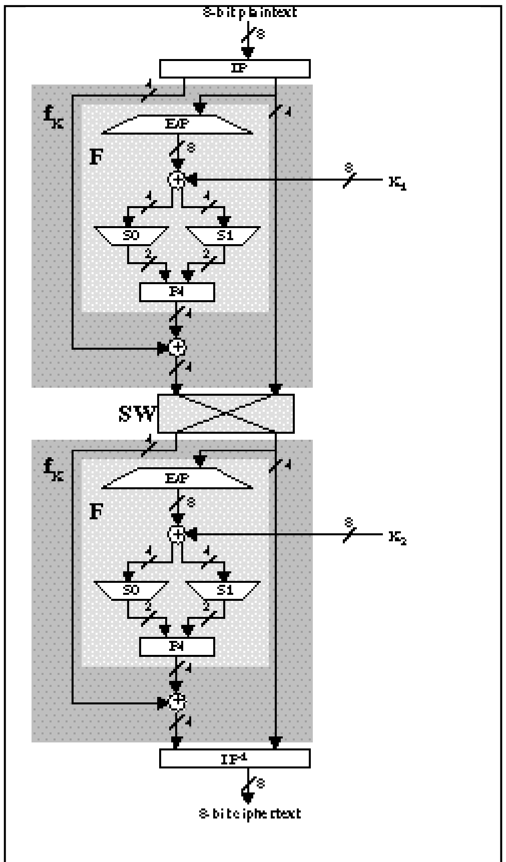


Figure 3.3 Simplified DES Encryption Detail

Advanced encryption standard (AES)

DES is now considered to be insecure for many applications; this is chiefly due to the 56-bit key size being too small. The **Advanced Encryption Standard** (**AES**) is the result of a 5-year US government standardisation process to select a replacement for DES. The result, called **Rijndael** after its inventors Vincent Rijmen and Joan Daemen, is expected to be used worldwide and analysed extensively, as was the with DES. AES is quite a lot more complicated than DES; for details, see the references. The main points about AES are:

- Longer key: key size of 128, 192 or 256 bits
- More elaborate key schedule
- As well as S-boxes, AES uses operations in finite fields (roughly, modulo arithmetic) which makes things harder to cryptanalyse.

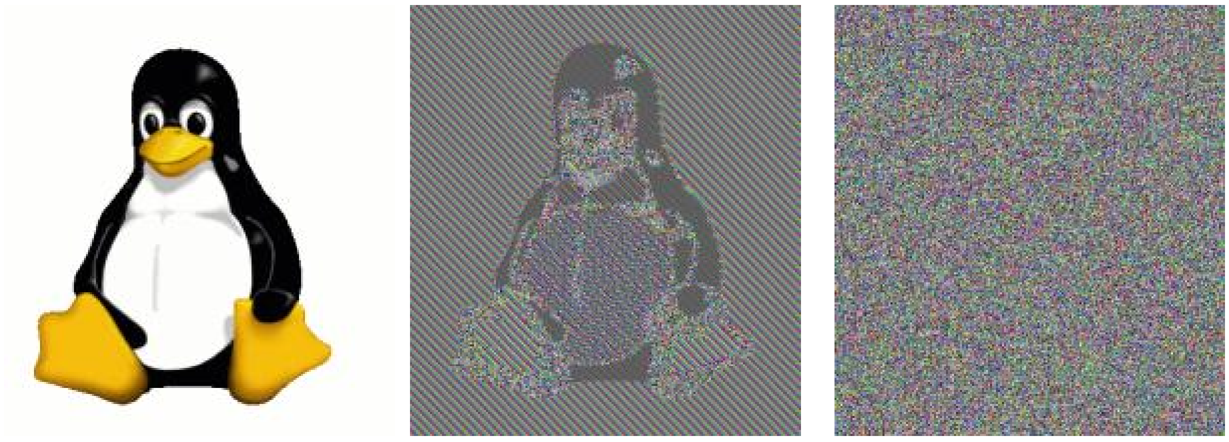
Other symmetric key algorithms

- 3DES (which is DES then inverse-DES then DES again, with different keys). This has the effect of tripling the keylength of DES to 168 bits, which makes it much more secure; it also has a backwards-compatibility advantage (a 3DES algorithm can be made to compute DES, by supplying a key which repeats twice after the first 56 bits).
- IDEA, rated by Schneier as the best one partly because of its "impressive theoretical foundations". It mixes XOR, addition modulo 2^{16}, and multiplication modulo 2^{16}-1, and is based on a 128-bit key.
- Blowfish, invented by Schneier to be fast, compact, easy to implement, and to have variable key length (up to 448 bits),

Block cipher modes

Block ciphers can be used in different *modes*:

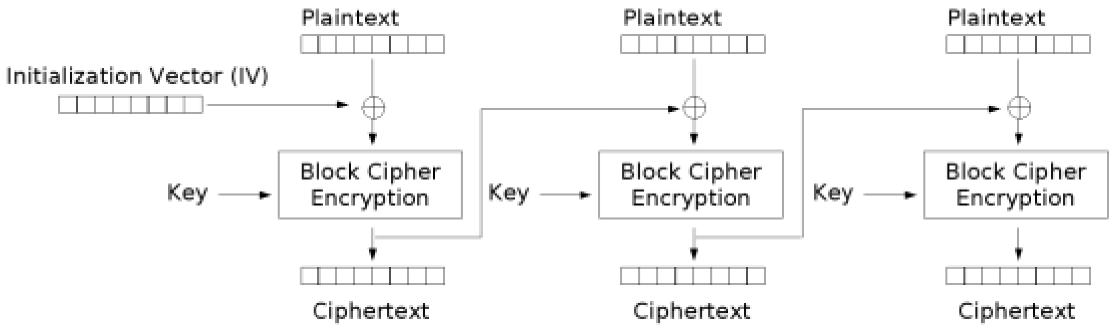
- **ECB (Electronic codebook mode)**. In this mode, each block is encrypted individually, and the encrypted blocks are assembled in the same order as the plain text blocks. This is the regular usage, but it leaks some information (e.g., if blocks are repeated in the plain text, this is revealed by the cipher text), and it is vulnerable to block replays. Here's a striking example [from Wikipedia article on *Block cipher modes*] of the degree to which ECB can reveal patterns in the plaintext. A pixel-map version of the image on the left was encrypted with ECB mode to create the center image:



Original *Encrypted using ECB mode* *Encrypted using secure mode*

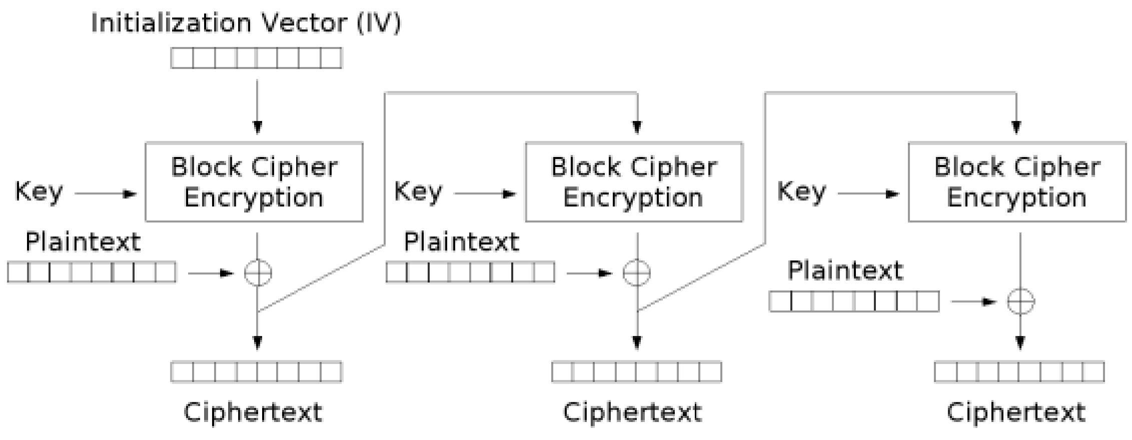
- **CBC (Cipher block chaining mode)** - each block XOR'd with previous block; helps overcome replay attack.
  Suppose the plain text is B_1, B_2, ..., B_n. We write + for XOR.
  C_1 = encrypt(B_1 + IV), where IV is a randomly chosen *initialisation vector*.
  C_2 = encrypt(B_2 + C_1).
  ...
  C_i = encrypt(B_i + C_{i-1}).

  The following diagram [taken from Wikipedia article on *Block cipher modes*] shows how it works.

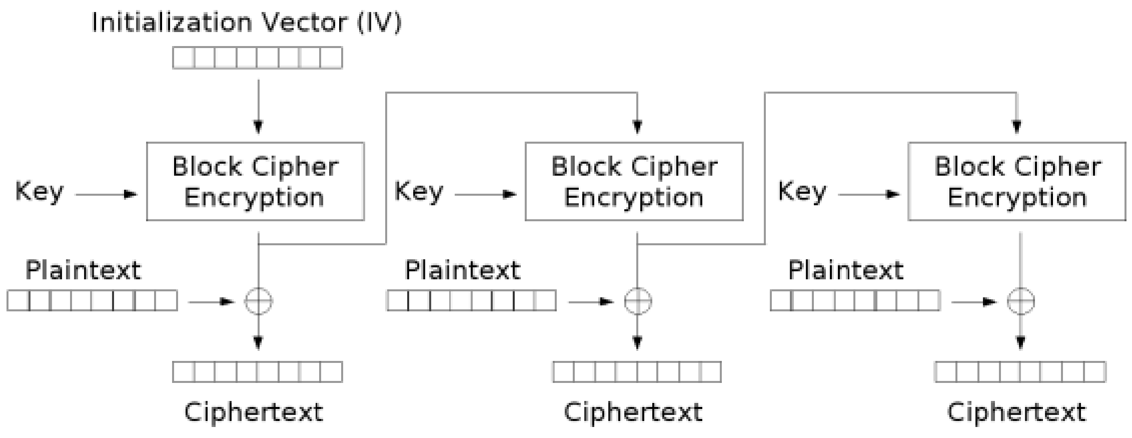


Cipher Block Chaining (CBC) mode encryption

- **CFB (Cipher feedback mode)** - makes a block cipher into a *stream cipher*, by maintaining a queue block (initialised to some initial value). **OFB (Output feedback mode)** is similar. The following diagrams [taken from Wikipedia article on *Block cipher modes*] show how these modes work.



Cipher Feedback (CFB) mode encryption



Output Feedback (OFB) mode encryption

Stream ciphers

Stream ciphers encrypt streams, e.g. a byte at a time, in cases that you can't wait for an entire block of text before starting the encyption. Typically, a stream is generated from the key, and the plaintext is XOR'd with the stream. Like a one-time pad generated on the fly (but without the security properties of one-time pads, of course!)

RC4 is a variable-key-size stream cipher developed in 1987 by Rivest. For seven years it was proprietary, but anonymously posted on the internet in 1994. It works in OFB: the keystream is independent of the plaintext. It maintains a vector S[0]...S[255], whose entries are a permutation of the numbers 0...255. The following algorithm encrypts a stream:

```
// initialise S
for i = 0 ... 255
    S[i] = i
for i = 0 ... 255 {
    j = (j + S[i] + key[i mod key_length]) mod 256
    swap (S[i],S[j])
    }
```

```
i = 0
j = 0
while (still some bytes left to encrypt/decrypt) {
    i = (i + 1) mod 256
    j = (j + S[i]) mod 256
    swap(S[i],S[j])
    k = S[(S[i] + S[j]) mod 256]
    output k XOR next_byte_of_input
    }
```

The first part of the algorithm uses the key to randomise the byte array S. The second part produces the bytes to be XORed with the plain text. Each line has a distinct notional purpose.

- `i = (i + 1) mod 256`

  makes sure every array element is used once after 256 iterations

- `j = (j + S[i]) mod 256`

  makes the output depend non-linearly on the array

- `swap(S[i],S[j])`

  makes sure the array is evolved and modified as the iteration continues

- `k = S[(S[i] + S[j]) mod 256]`

  makes sure the output sequence reveals little about the internal state of the array.

RC4 is optimused to work efficiently in software, and is the basis of several encryption schemes in current use, including WEP (the encryption standard used in wi-fi). Unfortunately, it has been shown that the first few bytes of the RC4 stream produced are significantly non-random. This vulnerability has been exploited to devise an attack on WEP. RC4 is not considered secure. Probably throwing away the first 1K bytes from RC4 is enough to solve this problem.

References

[1] Bruce Schneier, *Applied Cryptography*. Second Edition, J. Wiley and Sons, 1996.
[2] William Stallings, *Cryptography and Network Security, Principles and Practice,* Prentice Hall, 1999. Third Edition, 2003.
[3] The University of British Columbia Theoretical Physics department has a web page on cryptography, with lots of interesting remarks and links.
[4] Many useful Wikipedia articles.
[5] Nigel Smart, *Cryptography*. McGraw Hill, 2003.