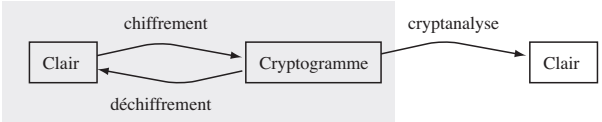
Cryptologie & sécurité

Bruno MARTIN,
Université Côte d'Azur

Cryptologie = cryptographie + cryptanalyse

Cryptologie = science de la communication en présence d'adversaires. Contrairement à la stéganographie, on ne dissimule pas l'existence du message.



But : **chiffrer** un **texte clair** avec une **clé** en un **chiffré** pour préserver sa **confidentialité** par diverses transformations.

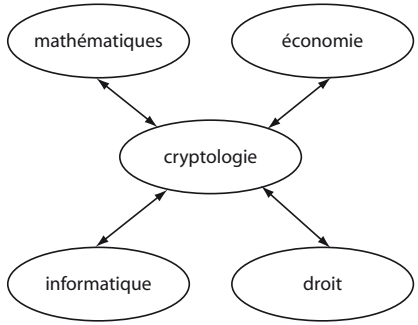
Le destinataire **déchiffre** le chiffré avec la **clé** pour avoir le clair. Un **cryptanalyste** ne doit pas pouvoir **cryptanalyser** le chiffré.

Sécurité

Ensemble de procédures pour assurer

- *confidentialité* : seules les personnes autorisées ont accès aux éléments considérés.
- *intégrité* : garantie que les éléments considérés sont exacts et complets
- *authentification* : garantie de l'origine d'un élément
- *disponibilité* : aptitude à remplir une fonction dans des conditions prédéfinies d'horaires de délais ou de performance
- *traçabilité* : garantie que les accès et tentatives d'accès aux éléments considérés sont tracés et que ces traces sont conservées et exploitables

Domaines (ou théories) connexes



Objectifs

- comprendre les principes
- comment utiliser les mécanismes de sécurité
- comment le faire bien
- comprendre leurs limites

Brève histoire de la cryptologie

J. Stern [1] scinde l'histoire de la cryptologie en 3 périodes :

- l'âge *artisanal* : modifier l'écriture
- l'âge *technique* : machines à chiffrer
- l'âge *paradoxal* : cryptographie à clé publique

1. cf. archives de l'université de tous les savoirs
http://www.canal-u.tv/video/universite_de_tous_les_savoirs/cryptologie_et_securite_informatique.1104

Âge artisanal – César

le clair « `toute la gaule` » devient `WRXWH OD JDXOH`.

(`A` devient `d`, `B` devient `e`...)

Âge technique – Enigma

le clair « `alles in ordnung` » devient `EDCGZVRRIOVRAY`

Âge paradoxal – DH

Pourquoi cette évolution ?

- parce que aucun (?) chiffre n'est sûr
- à cause des menaces :
  - passives (contre la confidentialité)
  - actives (contre l'intégrité et l'authenticité)

Pourquoi la cryptographie ?

Hier :

- pour des raisons stratégiques et assurer le SECRET (éviter qu'un ennemi lise un ordre de bataille)
- par l'église ou la diplomatie

Aujourd'hui :

↑ des communications (internet, phonie, rx mobiles. . .) requiert :

- confidentialité
- intégrité
- authentification

et un principe de **minimalité** ; rien n'est communiqué aux autres sauf ce qui est expressément spécifié.

Quelques attaques actives

- *impersonnification* (modif. émetteur ou récepteur)
- altération des données
- destruction du message
- retard de la transmission
- répudiation du message (émetteur nie l'envoi du message)

Cryptographie peut contrer ces attaques ; garantit confidentialité, intégrité, authentification ou identification, signature (et de plus en plus : vie privée ou *privacy*)

Buts de l'attaquant et techniques d'attaques

Attaques contre la confidentialité (l'intégrité et l'authenticité :)

- retrouver le(s) message(s) clair(s) envoyé(s)
- retrouver la clé
- lire/modifier les messages

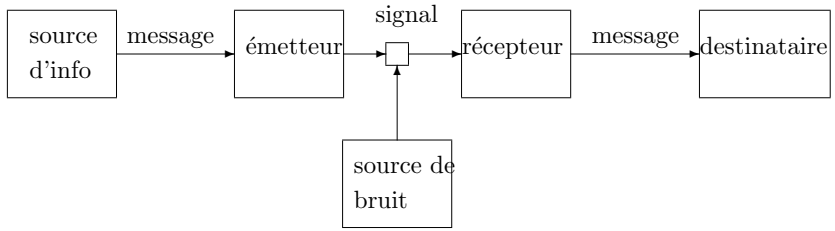
Techniques d'attaques :

- COA chiffré connu : accès à des chiffrés (mauvaise conception WEP, petit espace des clés de DES)
- KPA clair connu : accès à des couples clairs/chiffrés (vulnérabilité `zip` connu un couple$\mapsto$ clé)
- CPA clair choisi : choix clairs, obtention des chiffrés correspondants (attaque par dictionnaire mots de passe)
- CCA chiffré choisi : possibilité de déchiffrer des chiffrés choisis (accès au déchiffrement [oracle])

Modèles de sécurité

- **Calculatoire** : casser le chiffre requiert $> N$ opérations, N grand $\simeq 10^{120} \simeq 2^{400}$
- **Prouvée** : s'il existe une attaque, un problème difficile peut-être résolu (techniques de réduction, P$\neq$NP)
- **Sémantique** : impossible pour un adversaire à ressources bornées de déduire des informations sur le clair (ind.)
- **Parfaite** : ne peut être cassé, même par une attaque à ressources non bornées (théorie de l'information de Shannon)

Théorie de l'information (Weaver et Shannon [6] 1964)



Traite l'aspect quantitatif de l'information (notion d'**entropie**) ; on considère les systèmes **discrets** pour lesquels message et signal sont deux suites de symboles d'un alphabet fini.
Problématiques : **codage**, **compression** et **secret**.

Problématiques des systèmes discrets

- le codage
  - pour un canal non bruité (construction de codes optimaux)
  - pour un canal bruité (il existe de bons codes)
- la compression
  - avec pertes
  - sans perte
- le secret

Le codage

Canal non bruité (codage de source) : pour transmettre un message ou un signal, on le code. Le codage dépend du canal de transmission (écriture, parole, code Morse, code ASCII, unicode...). Souvent en binaire.

Canal bruité (codage de canal) : le signal peut être bruité entre l'émetteur et le récepteur (i.e. perturbé par un signal aléatoire). Pour recouvrer le message émis à partir du signal reçu, l'émetteur ajoute de la redondance. Le récepteur peut alors soit détecter (*codes détecteurs*) soit corriger 1 ou + erreurs (*codes correcteurs*).

But : transmettre rapidement et correctement des messages, notion de sûreté de la transmission.

Compression et secret

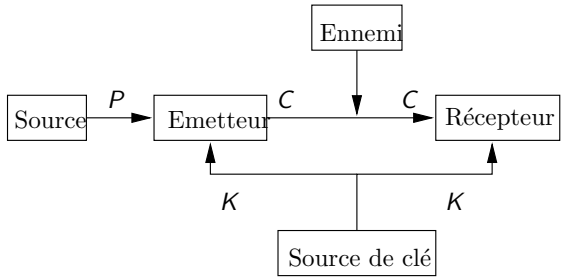
Compression : forte redondance des langues naturelles – et de la plupart des informations : sons, images, textes, programmes. Compression : retirer autant de **redondance** que possible

---

Secret : Un ≪ bruit ≫ perturbe le message (cacher son contenu). Rôle de l'émetteur : construire une perturbation du message pour :

- empêcher un *cryptanalyste* de décrypter le message
- permettre au destinataire légal de retrouver le clair

Modèle de Shannon pour un chiffre à clé secrète [5]

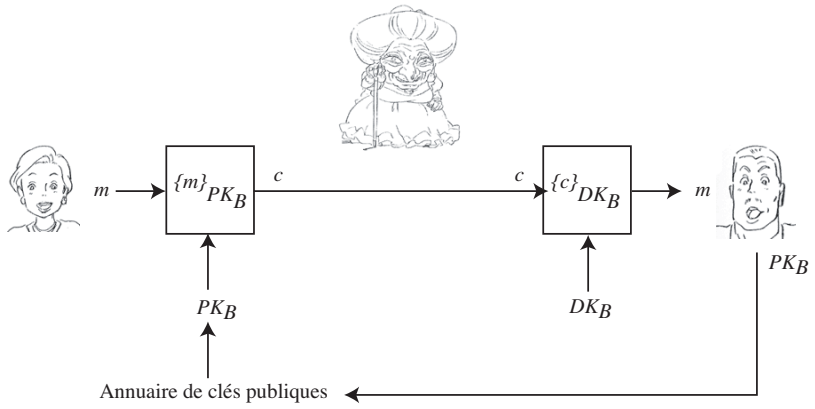


Repose sur 3 algorithmes :

- générateur de clé (*-aléatoire)
- algorithme de chiffrement $P, K \mapsto C = \{P\}_K$
- algorithme de déchiffrement $C, K \mapsto P = \{C\}_K$

Fournit un canal confidentiel pour K partagée sur un canal sûr

Chiffre à clé publique



Quel paradigme est le plus utilisé ?

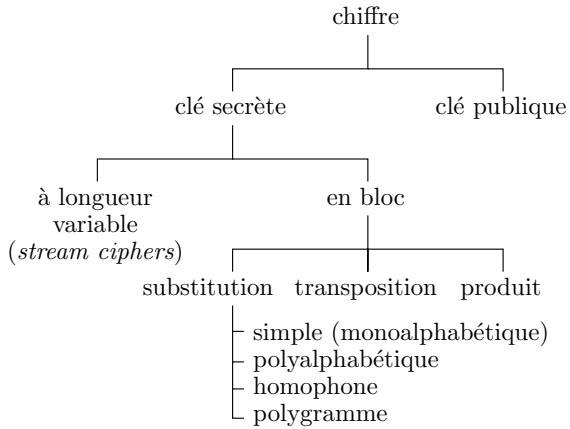
Les deux !

- clés publiques : pour l'échange de clés (secrètes, publiques)
- clé secrètes : pour chiffrer de grandes quantités de données (environ 1000× plus rapide, voire plus !)

Hypothèses de la cryptographie moderne

- Problème n^2 : mettre en communication n utilisateurs
- Principe de Kerckhoffs : *La sécurité d'un chiffre ne dépend pas du secret de l'algorithme mais seulement de celui de la clé*
- Loi de Moore (empirique) : la vitesse des CPU double tous les 2 ans ; facilite la recherche exhaustive de clé
- Loi de Murphy : s'il y a une faille de sécurité, quelqu'un la trouvera forcément !

Petite classification des chiffres



Substitutions : chiffres monoalphabétiques

Chiffre monoalphabétique : bijection entre l'alphabet des clairs et celui des chiffrés. Si alphabets identiques : permutation.

Exemple

Chiffre de César. *En identifiant les alphabets* $\{a,\ldots,z\}$ *et* $\{A,\ldots,Z\}$ *à* $\{0,\ldots,25\} = \mathbb{Z}_{26}$, **chiffre additif**.

- *chiffrement :* $x \mapsto x + 3 \mod 26\ \forall x \in \mathbb{Z}_{26}$.
- *déchiffrement :* $y \mapsto y - 3 \mod 26\ \forall y \in \mathbb{Z}_{26}$.

Combien de chiffres additifs ?
Avec l'alphabet $\mathbb{Z}_{26}$, 26 chiffres additifs en comptant $s = 0$.
L'ensemble des chiffres monoalphabétiques correspond aux permutations ; donc $26! \approx 4.10^{26}$ chiffres monoalphabétiques !

Substitutions : chiffres multiplicatifs

Multiplier au lieu d'additionner : $x \mapsto t \cdot x \mod 26$ pour $t \in \mathbb{N}$.
Valeurs de t acceptables : $\text{pgcd}(t, 26) = 1$ ou, $t \nmid 26$:
$\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$ (on en a 12)
Les autres n'assurent pas l'unicité du chiffrement. (p.e. 2)

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
0	2	4	6	8	10	12	14	16	18	20	22	24
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25
0	2	4	6	8	10	12	14	16	18	20	22	24

C'est la fonction `ROT13` d'Emacs (`M-x rot-13-other-window`) !

Substitutions : de bons chiffres ?

```
#!/usr/bin/ruby
clair='RAWFEJBANAREQSSQBDAWKRSKWK'
def codemod26(texte)
    tabnum=[]
    clair=texte.upcase
    for i in 0..clair.length-1 do
        tabnum[i]=clair[i].to_i - 65
    end
    tabnum
end
def euclidExtended(a, b)
#d=ax+by
  d, dd, x, xx, y, yy = 0
  if b == 0
    return a, 1, 0
  else
    dd, xx, yy = euclidExtended(b, a % b)
    d, x, y = dd, yy, xx - a / b * yy
    return d, x, y
  end
end
```

```
C=codemod26("RAWFEJBANAREQSSQBDAWKRSKWK")
D=[]

for i in 1..26 do
    d, x, y = euclidExtended(i,26)
    if d == 1
       j=0
       C.each do |c|
           D[j] = c*x % 26 + 65
           j += 1
       end
       D.each do |d|
             print d.chr
       end
       print " i=", i, "\t inv= ",x\%26,"\n"
    end
end
```


Conclusions sur la substitution simple

- Pas très robuste, ni à une recherche exhaustive, ni à une analyse statistique (distribution des fréquences des lettres).
- pour contrer l'analyse des fréquences, la distribution statistique des lettres doit tendre vers une loi uniforme.
- Amélioration : utiliser une transformation qui associe à chaque lettre du clair un ensemble de lettres de l'alphabet du chiffré.

On obtient ainsi les **chiffres polyalphabétiques**

Substitution : chiffres polyalphabétiques

Dans un **chiffre polyalphabétique**, les caractères du clair sont transformés au moyen d'une clé $K = k_0, \ldots, k_{j-1}$ qui définit j fonctions différentes $f_0, \ldots, f_{j-1}$ telles que

$$\forall i,\ 0 < j \leq n \quad f_{k_\ell} : \mathcal{A}_M \mapsto \mathcal{A}_C, \forall \ell,\ 0 \leq \ell < j$$
$$c_i = f_{k_i \mod j}(m_i)$$

Cas du chiffre de Blaise de Vigenère 1586 (diplomate français).
Idée : combiner des chiffres monoalphabétiques $\neq$.

Cryptanalyse de Vigenère

- test de Kasiski
- test de Friedmann

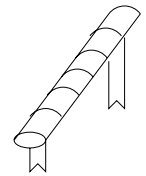
http://fr.wikipedia.org/wiki/Cryptanalyse_du_chiffre_de_Vigenere

Transposition

Elle implémente une permutation des caractères clairs $\mathcal{A}_C = A_M$.

$$\forall i, \quad 0 \leq i < 0 \quad \begin{array}{l} f : \mathcal{A}_M \to \mathcal{A}_M \\ \eta : \mathbb{Z}_n \to \mathbb{Z}_n \\ c_i = f(m_i) = m_{\eta(i)} \end{array}$$

Transposition : scytale



Exemple de transposition simple à tableau

A partir d'une phrase clé, on définit une clé numérique :

T	R	A	N	S	P	O	S	I	T	I	O	N	S	I	M	P	L	E
18	14	1	8	15	12	10	16	3	19	4	11	9	17	5	7	13	6	2

On chiffre, « *le chiffrement est l'opération qui consiste à transformer un texte clair, ou libellé, en un autre texte inintelligible appelé texte chiffré ou cryptogramme* » [4].

18	14	1	8	15	12	10	16	3	19	4	11	9	17	5	7	13	6	2
l	e	c	h	i	f	f	r	e	m	e	n	t	e	s	t	l	o	p
é	r	a	t	i	o	n	q	u	i	c	o	n	s	i	s	t	e	à
t	r	a	n	s	f	o	r	m	e	r	u	n	t	e	x	t	e	c
l	a	i	r	o	u	l	i	b	e	l	l	é	e	n	u	n	a	u
t	r	e	t	e	x	t	e	i	n	i	n	t	e	l	l	i	g	i
b	l	e	a	p	p	e	l	é	t	e	x	t	e	c	h	i	f	f
r	é	o	u	c	r	y	p	t	o	g	r	a	m	m	e			

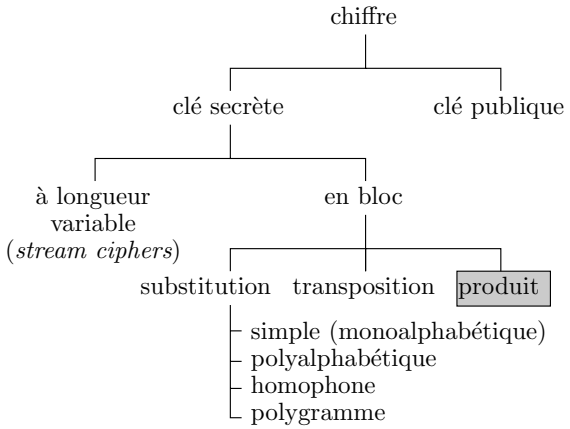
On prend ensuite par blocs de 5 lettres les colonnes prises dans l'ordre défini par la clé.

Exemple de transposition simple à tableau

CAAIE	EOPAC	UIFEU	MBIET	ECRLI	EGSIE
NLCMO	EEAGF	TSXUL	HEHTN	RTAUT	NNETT
AFNOL	TEYNO	ULNXR	FOFUX	PRLTT	NIIER
RARLE	IISOE	PCRQR	IELPE	STEEE	MLETL
TBRMI	EENTO				

Attaques
Par analyse de la distribution des lettres, bigrammes[1],...

Petite classification des chiffres



Chiffres produits et itérés [5]

Amélioration : combiner substitutions et transpositions (produit).
Chiffre produit **itéré** : chiffré obtenu par applications itérées d'une fonction de tour combinant texte d'entrée et clé de tour.

Définition

Dans un chiffre itéré à r tours, le chiffré est calculé par application itérée au clair d'une **fonction de tour** *g t.q.*

$$C_i = g(C_{i-1}, K_i) \quad i = 1, \ldots, r$$

où C_0 est le clair, K_i une clé de tour et C_r le chiffré.
Déchiffrer en inversant l'équation précédente. Pour une clé fixée K_i, g doit être inversible.

Cas particulier, **les chiffres de Feistel**.

Chiffre de Feistel

Un **chiffre de Feistel** de taille de bloc $2n$ à r tours est défini par :

$$g : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^n$$

$$X, Y, Z \mapsto (Y, F(Y, Z) \oplus X)$$

g de $2n \times m$ bits dans $2n$ bits et $\oplus$ XOR sur n bits.
Fonctionnement
Pour un clair $P = (P^L, P^R)$ et r clés de tour $K_1, \ldots, K_r$, le chiffré (C^L, C^R) est calculé en r tours.
On pose $C_0^L = P^L$ et $C_0^R = P^R$ et on calcule pour $i = 1, \ldots, r$

$$(C_i^L, C_i^R) = (C_{i-1}^R, F(C_{i-1}^R, K_i) \oplus C_{i-1}^L)$$

avec $C_i = (C_i^L, C_i^R)$ et $C_r^R = C^L$ et $C_r^L = C^R$ (permutation finale).
$K_1, \ldots, K_r$ calculées depuis une clé principale K par un algorithme de séquencement de clé.

Le DES

- appel d'offres du NBS en 1973
- DES (*Data Encryption Standard*) déposé par IBM en 1975
- adopté en 1977 avec évaluation tous les 4 ans
- attaque RSALabs 1999 : clé retrouvée en 22h (machine dédiée 250 000$)
- changement de la taille de la clé (3DES) : 128 bits
- appel d'offres du NIST en 1997 pour le remplacer
- son remplaçant élu en 2000 : AES ou Rijndael [2]
- exemple de chiffrement par le DES dans STINSON [7]

Utilité du DES
Le DES était le chiffre produit le plus utilisé.
Chiffre de Feistel avec des propriétés particulières.

Fonctionnement

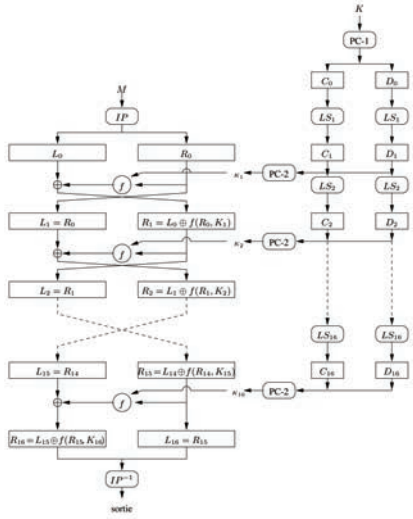
Entrée du DES :

- un message M de 64 bits ;
- une clé K de 56 bits.

Sortie : un chiffré C de 64 bits.
M subit une permutation initiale $IP \rightarrow$ message $\ll$ permuté $\gg M'$.
M' découpé ensuite en 2 mots de 32 bits :

- L_0 partie gauche
- R_0 partie droite.

DES itère 16 fois f qui combine substitutions/transpositions.



DES

Les parties gauches et droites deviennent pour $1 \leq i \leq 16$:

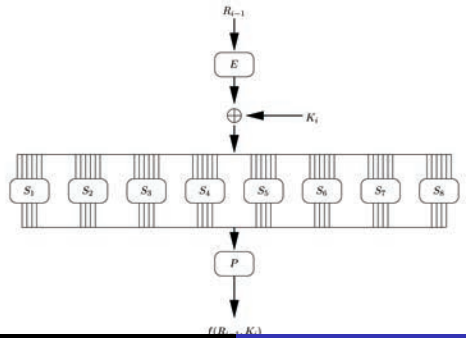
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

f entrée : 32 bits partie droite et 48 bits clé de tour ; 32 bits sortie.
f définie par 8 S-**boxes** –S pour substitution– qui associent à 6 bits d'entrée 4 bits de sortie.
Chaque K_i a un sous-ensemble différent des 56 bits de la clé.
Le *pré-chiffré* $C' = (R_{16}, L_{16})$ subit la permutation inverse de la permutation initiale IP^{-1} et donne le chiffré final.

Fonctionnement de $f(R_{i-1}, K_i)$

R_{i-1} est étendu en 48 bits par E, table de sélection des bits.

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



S-box S_1

	Colonne															
ligne	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	12	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

Chaque S-box associe à $B_j = b_1 b_2 b_3 b_4 b_5 b_6$ un bloc de 4 bits ; l'entier représenté par $b_1 b_6$ sélectionne une ligne de S_j et celui représenté par $b_2 b_3 b_4 b_5$ une colonne. La valeur de $S_j(B_j)$ est la représentation de l'entier inscrit dans la S-box à cette position.

Exemple

Si $B_1 = 010011$ *alors ligne* $= 01$ *ou 1 en décimal et colonne* $= 1001$ *ou 9 en décimal. La valeur de* $S_1[1, 9] = 6$ *ou* 0110 *en binaire et* $S_1(010011) = 0110$.

Permutation P

Les huits blocs de 4 bits correspondant à $S_1(B_1)S_2(B_2)\dots S_8(B_8)$ sont concaténés en 32 bits qui subissent une permutation 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

Le bloc de 32 bits retourné par $f(R_{i-1}, K_i)$ est donc :

$$f(R_{i-1}, K_i) = P(S_1(B_1)S_2(B_2)\dots S_8(B_8))$$

Diversification de la clé

À chacune des 16 itérations, utiliser une K_i différente ($1 \leq i \leq 16$) de 48 bits obtenue de la clé initiale K de 64 bits.
La clé initiale K est composée de 64 bits comprenant 8 bits de parité en position $8, 16, \dots, 64$. Ceux-ci sont supprimés et les 56 bits restants subissent une permutation à l'aide de la fonction PC-1 (pour permuted choice 1)

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

Diversification de la clé

PC-1(K)=$C \cdot D$, 2 blocs de 28 bits subissent une permutation circulaire vers la gauche pour construire chaque clé de tour K_i. En notant C_i et D_i les valeurs de C et D, on a :

$$C_i = LS_i(C_{i-1}), \qquad D_i = LS_i(D_{i-1})$$

pour C_0 et D_0 les valeurs initiales de C et de D et LS_i une permutation circulaire de j positions vers la gauche avec j une fonction de i, le numéro de l'itération :

itération	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
décalage	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Valeur de K_i donnée par $K_i = \text{PC-2}(C_iD_i)$ avec PC-2 une fonction de permutation comparable à PC-1 mais sur 56 bits.

Déchiffrement

Déchiffrement par le même algorithme en inversant l'ordre d'utilisation des clés de tour i.e. en utilisant K_{16} à la 1re itération, K_{15} à la 2e . . .et K_1 à la 16e. Dû au fait que la permutation finale est l'inverse de la permutation initiale et

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

Même si l'ordre des clés de tour est inversé, l'algo. reste identique.

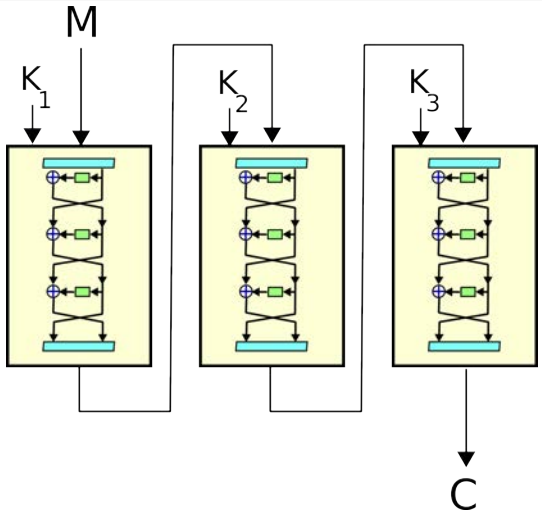
Triple DES

3 applications sur P (64 bits), avec $\underline{2}$ ($K_3 = K_1$) ou 3 clés.

$$C = E_{K_3}\left(D_{K_2}(E_{K_1}(P))\right)$$

- Usage standard : mode EDE (Encryption, Decryption, Encryption) ; si 3 fois la même clé on retrouve DES.
- Dans le cas d'une implémentation matérielle cela permet d'utiliser le même composant pour le DES et le 3DES.

Triple DES



UNIX passwords

- utilisation 25 fois d'un DES modifié (64 bits de sortie) $H = \{\ldots\{IV\}_P\ldots\}_P$ sur IV=0...0
- H est enregistré dans `/etc/passwd` sous forme de 11 caractères sur 6 bits dans l'ensemble $\{".","/",0-9,A-Z,a-z\}$.
- à chaque connexion, l'utilisateur fournit P' et on vérifie

$$H \stackrel{?}{=} \{\ldots\{IV\}_{P'}\ldots\}_{P'}$$

- P découpé en blocs de 8 char (dernier bloc à 0). Les 7 bits de poids faible de chaque bloc sont utilisés pour constituer la clé du DES :
  - le premier bloc de 56 bits est la clé initiale de DES, P_0
  - chaque bloc additionnel subit un XOR avec $\mathrm{DES}_{P_{i-1}}(P_{i-1})$

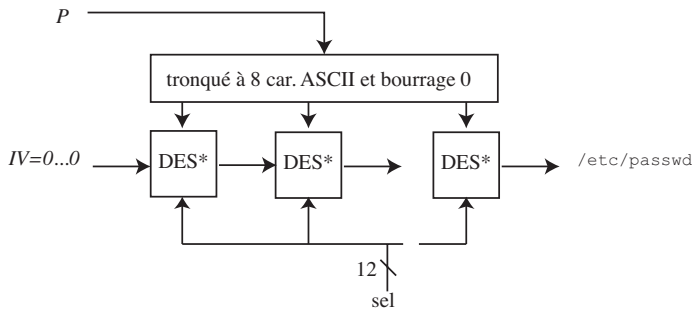
UNIX passwords

- le ≪ sel ≫ aléatoire : 12 bits (2 caractères) modifient la fonction E de DES en ajoutant une permutation –parmi 4096– enregistrée dans `/etc/password`. Si le bit i du sel est à 1, les bits i et $i+24$ de E sont échangés.

Contenu de `/etc/passwd`

password	salt	encrypted
nutmeg	Mi	MiqkFWCm1fNJI
ellen1	ri	ri79KNd7V6.Sk
Sharon	./	./2aN7ysff3qM
norahs	am	amfIADT2iqjAf
norahs	7a	7azfT5tIdyh0I

norahs chiffré avec 2 sels différents.
sel chiffre 1 même mdp de 4096 façons différentes.



et ses attaques par dictionnaire ...

- Consiste à tester une série de mots de passe potentiels. Repose sur le constat que de nombreuses personnes utilisent des mots de passe courants.
- Attaque utilisée en complément de l'attaque par force brute (Voir `John the ripper, L0phtCrack, RainbowCrack, THC Hydra`)

ou par rainbow tables

Une *rainbow table* est une table précalculée pour inverser les fonctions de hachage cryptographiques, généralement pour casser les empreintes de mots de passe. Les tables sont généralement utilisées pour récupérer un mot de passe en clair jusqu'à une certaine longueur, composé d'un ensemble limité de caractères. Il s'agit d'un exemple pratique de compromis espace/temps, utilisant moins de temps de traitement informatique et plus de stockage qu'une attaque par force brute qui calcule une empreinte à chaque tentative, mais plus de temps de traitement et moins de stockage qu'une simple table de consultation avec une entrée par hachage. L'utilisation d'une fonction de fonction de dérivation de clé qui utilise un sel rend cette attaque infaisable. Les rainbow tables sont une application d'un algorithme antérieur, plus simple, de Martin Hellman.

Password cracking tools

- John the Ripper : fast password cracker, available for many OSes. Its primary purpose is to detect weak Unix passwords. Besides several `crypt(3)` password hash types most commonly found on various Unix systems, supported out of the box are Windows LM hashes, plus lots of other hashes and ciphers in the community-enhanced version. Free and Open-Source.
- THC Hydra : proof of concept code, to give researchers and security consultants the possibility to show how easy it would be to gain unauthorized access from remote to a system.
- RainbowCrack : general purpose implementation of a faster time-memory trade-off technique. Cracks hashes with rainbow tables. Uses time-memory tradeoff algorithm to crack hashes. Differs from brute force hash crackers.

Autres algorithmes

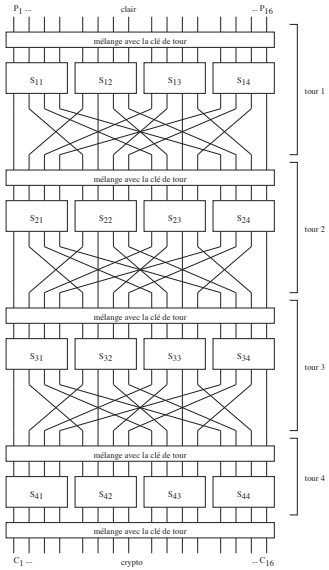
V.O. : `crypt()`
`crypt16()` : Certains UNIX (HP-UX, BSD 4.4) peuvent utiliser une librairie qui utilise plus de 16 caractères significatifs (au lieu des 11). Une valeur du sel plus grande est utilisée.

http://docstore.mik.ua/orelly/networking/puis/ch08_06.htm

/etc/shadow ou BSD /etc/master.passwd

- `/etc/passwd` en lecture par tout le monde !
- autorise attaque par dictionnaire
- `/etc/shadow` en lecture seule par root ; 2 fonctionnements :
  - comme `/etc/passwd`
  - ou une "status exception value" `$id$salt$encrypted` avec `$id` un algo de hachage cryptographique ($1 : MD5, $2 : Blowfish, $5 : SHA-256, $6 : SHA-512), `$salt`, la valeur du sel et `$encrypted` résultat de la fonction de hachage donnée par $id sur le sel+mdp.

et dans tous les cas, `/etc/passwd` contient un `x` à la place de l'information sur le mot de passe.



Présentation du chiffre utilisé

Réseau de substitution/permutation simplifié [3]. Entrée : 16 bits : fonctionne en 4 tours. Chaque tour consiste en :

- **un mélange** avec la clé : xor entre la clé de tour et le bloc d'entrée du tour. Opération répétée à l'issue du dernier tour.
- **une substitution**. 16 bits scindés en 4 sous-blocs qui entrent dans 4 boîtes-S identiques à 4×4. (MSB à gauche)

in	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*
out	*E*	4	*D*	1	2	*F*	*B*	8	3	*A*	6	*C*	5	9	0	7

- **une transposition** définie par *la i^e sortie de la j^e boîte-S est reliée à la j^e entrée de la i^e boîte-S*

Boîte S

0	0000	1110	*e*	8	1000	0011	3
1	0001	0100	4	9	1001	1010	*a*
2	0010	1101	*d*	*a*	1010	0110	6
3	0011	0001	1	*b*	1011	1100	*c*
4	0100	0010	2	*c*	1100	0101	5
5	0101	1111	*f*	*d*	1101	1001	9
6	0110	1011	*b*	*e*	1110	0000	0
7	0111	1000	8	*f*	1111	0111	7

Déchiffrement

En inversant le fonctionnement du chiffre :

- S-boxes sont des bijections
- application des clés de tour en ordre inverse du chiffrement

Références


F.L. Bauer.
Decrypted secrets.
Springer Verlag, 1997.


J. Daemen and V. Rijmen.
AES proposal : Rijndael.
Technical report, Katholieke Universiteit Leuven, 1999.


H. M. Heys.
A tutorial on linear and differential cryptanalysis, 2002.


D. Kahn.
La guerre des codes secrets.
InterEditions, 1980.


L.R. Knudsen.
Block ciphers – a survey.
In Springer Verlag, editor, *State of the art in applied cryptography*, number 1528 in LNCS, pages 18–48, 1998.


C.E. Shannon and W. Weaver.
The mathematical theory of communication.
University of Illinois press, 1964.


D. Stinson.
Cryptographie, théorie et pratique.
International Thomson Publishing, 1995.