Notes de cours
Semaine 8

Cours Turing

1 Cryptographie à clé secrète : suite

Cette semaine, nous poursuivons ce chapitre sur la cryptographie à clé secrète, en nous focalisant sur deux thèmes :

- la *clé à usage unique* : système démontré inviolable, mais pas sans défauts...

- le *système DES* (pour "Data Encryption Standard"), ancêtre du systèms AES (pour "Advanced Encryption Standard"), encore utilisé aujourd'hui !

1.1 Préliminaire : représentation binaire et addition modulo 2 (XOR)

Représentation binaire des nombres entiers positifs

Voici un bref rappel : pour représenter un nombre entier positif N en binaire, il importe d'écrire celui-ci comme une somme de puissances de 2, de la même façon que la représentaion décimale de ce même nombre utilise les puissances de 10. Ainsi, nous avons :

$$1984 = 1000 + 900 + 80 + 4 = \mathbf{1} \cdot 10^3 + \mathbf{9} \cdot 10^2 + \mathbf{8} \cdot 10^1 + \mathbf{4} \cdot 10^0$$

d'où la représentation décimale "1984". Voici maintenant la même chose en binaire :

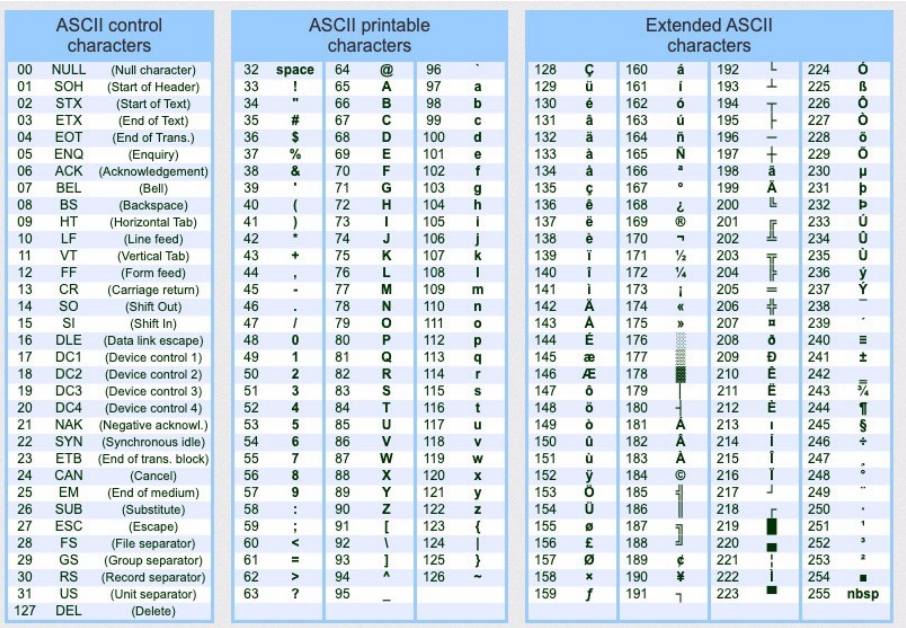
$$\begin{aligned} 1984 &= 1024 + 512 + 256 + 128 + 64 \\ &= \mathbf{1} \cdot 2^{10} + \mathbf{1} \cdot 2^9 + \mathbf{1} \cdot 2^8 + \mathbf{1} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{0} \cdot 2^0 \end{aligned}$$

d'où la représentation binaire "11111000000".

Avec n bits, il est possible de représenter tous les nombres entiers allant de 0 (000...0) à $2^n - 1$ (111...1). En particulier, avec 8 bits (équivalent à 1 octet), on peut représenter tous les nombres de 0 à 255.

Représentation binaire des chaînes de caractères

Dans le code ASCII étendu, un caractère est encodé sous la forme d'un nombre entier allant de 0 à 255 (et donc représentable sous forme binaire par 1 octet) :



ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Pour représenter une chaîne de n caractères sous forme binaire, n octets sont donc nécessaires.

Addition modulo 2 (opération XOR)

Une opération qui va se révéler très utile dans la suite est l'*addition modulo 2* de deux bits, dite aussi *opération XOR* (pour "eXclusive OR" en anglais), dénotée par le symbole $\oplus$ et définie ainsi :

$$0 \oplus 0 = 0, \quad 1 \oplus 0 = 1, \quad 0 \oplus 1 = 1, \quad 1 \oplus 1 = 0$$

Cette opération correspond à l'addition classique sur les nombres 0 et 1, *sauf* pour le dernier calcul, où on ne retient que le bit des unités de l'addition 1+1 (qui rappelez-vous vaut 10 en binaire). Comme nous allons le voir dans la section suivante, il peut être très utile de réaliser des opérations XOR sur des *séquences* de bits. Par exemple, on obtient :

$$\begin{aligned} & 01101001 \\ \oplus\ & 11010011 \\ = \ & 10111010 \end{aligned}$$

(à noter que cette opération n'a *rien à voir* avec l'addition "classique" en colonnes de deux nombres entiers représentés en binaire)

En Python, cette opération s'écrit **a^b** pour deux nombres entiers **a** et **b** (int).

1.2 Clé à usage unique (1917)

Ce système de chiffrement est aussi connu sur le nom de "masque jetable" ou "chiffre de Vernam" (du nom de son inventeur), ou encore "one-time pad" dans sa version anglaise. Pour le décrire, supposons que la clé secrète K en possession d'Alice et Bob soit une séquence de n bits (nous verrons encore plus en détail comment celle-ci doit être générée). Pour chiffrer un message M encodé sous la forme d'une autre séquence de n bits, Alice effectue l'opération suivante :

$$C = M \oplus K, \quad \text{ce qui est une notation abrégée pour :} \quad C_i = M_i \oplus K_i, \quad i = 0, \ldots, n-1$$

et envoie le message chiffré C à Bob. Celui-ci, également en possession de la clé K, effectue à son tour la *même opération* sur le message reçu : $D = C \oplus K$, et retrouve ainsi le message envoyé M, car

$$D = C \oplus K = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M$$

En effet :

- l'opération XOR est associative, comme l'addition standard ;

- la séquence $K \oplus K$ est une séquence de 0, car nous avons vu que $0 \oplus 0 = 1 \oplus 1 = 0$, donc en effectuant l'opération XOR de la séquence de bits K avec elle-même, on obtient simplement une séquence de 0.

Sécurité de ce système

Alice et Bob ont ainsi trouvé une façon de communiquer secrètement. Pour autant, ce système est-il 100% sûr ? C'est ici qu'il importe de décrire plus précisément comment la clé K doit être générée. Pour que tout fonctionne bien, il faut trois ingrédients :

- chaque bit K_i de la clé K doit être tiré *uniformément* au hasard, c'est-à-dire que les probabilités que $K_i = 1$ et $K_i = 0$ valent toutes deux $\frac{1}{2}$, pour toute valeur de i allant de 0 à $n-1$;

- les bits $K_0, K_1, \ldots, K_{n-1}$ doivent être tirés *indépendamment les uns des autres*, comme des dés qu'on lancerait sur une grande table ;

- le tirage aléatoire de la clé K doit être aussi effectué *indépendamment du message M à envoyer*.

Pourquoi toutes ces conditions ? Pour la bonne raison suivante : si maintenant Eve intercepte le message chiffré C, que peut-elle en déduire sur le message d'origine M ?

Pour chaque bit C_i, la probabilité que $C_i = 0$ est la même que la probabilité que $M_i \oplus K_i = 0$, ce qui revient à dire que $K_i = M_i$. Or $K_i = 1$ ou $K_i = 0$ avec la même probabilité $\frac{1}{2}$. Donc pour toute valeur de M_i, la probabilité que $K_i = M_i$ vaut aussi $\frac{1}{2}$. Et donc la probabilité que $C_i = 0$ vaut $\frac{1}{2}$ (et donc, il en va de même pour la probabilité que $C_i = 1$). Pour Eve, qui ne connaît ni M, ni K, la séquence de bits C apparaît donc comme une séquence de bits complètement aléatoire ! Elle ne peut donc strictement rien déduire de cette séquence sur le message M d'origine : le système est sûr à 100%.

Quiz

- Pourquoi une attaque par force brute ne permettrait-elle pas de casser le système de clé à usage unique (en supposant donc qu'Eve dispose de la puissance de calcul nécessaire pour essayer toutes les clés possibles) ?

- (Question reliée) Vu que la clé K est tirée au hasard, il y a une probabilité (petite, certes, mais non-nulle) que tous les bits de K valent exactement 0. Dans ce cas, le message chiffré C est égal au message d'origine M : est-ce grave ?

Une clé malheureusement non réutilisable (d'où son nom...)

Ceci dit, le plus gros défaut de ce système est le suivant : supposons qu'Alice désire réutiliser la clé K pour envoyer deux messages M_1 et M_2, chacun de même longueur n que la clé K. Ainsi, Alice envoie les deux messages chiffrés C_1 et C_2 suivants :

$$C_1 = M_1 \oplus K \qquad\qquad C_2 = M_2 \oplus K$$

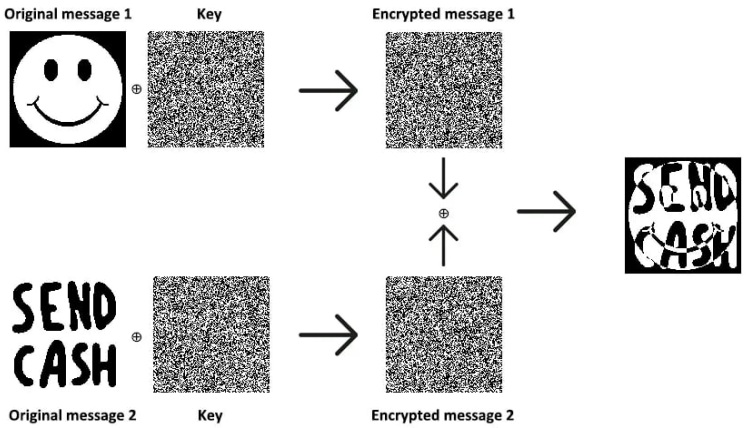
Individuellement, chaque message est bien protégé, comme nous venons de le voir. Mais si Eve intercepte les deux messages chiffrés C_1 et C_2, rien ne l'empêche d'effectuer une opération XOR sur ceux-ci, pour obtenir :

$$C_1 \oplus C_2 = (M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus K \oplus M_2 \oplus K = (M_1 \oplus M_2) \oplus (K \oplus K)$$

car l'opération XOR est non seulement associative, mais aussi *commutative* (comme l'addition standard). D'autre part, nous avons vu aussi que $K \oplus K = 0$, et donc, finalement :

$$C_1 \oplus C_2 = M_1 \oplus M_2$$

ce qui veut dire que la clé K a maintenant totalement disparu ! Certes, il peut être difficile pour Eve de déduire quelque chose à propos des messages M_1 et M_2 à partie du seul $M_1 \oplus M_2$, mais la sécurité du chiffrement n'est plus du tout assurée ! Pensez par exemple au cas où M_1 contient une longue suite de 0... Le problème est bien illustré sur le schéma ci-dessous : si deux images sont chiffrées avec la même clé, alors un XOR des deux images chiffrées (où noir et blanc jouent le rôle de 0 et 1) donne l'image à droite :



Cette impossibilité de réutiliser de la clé est clairement un très grave défaut du système ! Si Alice désirait chiffrer un gros fichier de données avec ce système, elle devrait produire une clé de même taille, ce qui en soi est déjà coûteux (car produire de longues séquences de bits aléatoires ne s'improvise pas), mais il y a bien pire : il faudrait d'abord partager la clé secrètement avec Bob avant de communiquer. Or si une telle occasion de communiquer secrètement se présentait, pourquoi ne pas utiliser celle-ci pour communiquer directement le fichier ?

Tentons donc autre chose pour réutiliser la clé K pour le chiffrement de plusieurs messages...

1.3 Data Encryption Standard (DES, 1977)

Revenons au chiffrement d'un seul message M avec une clé K de même longueur, disons n. Un premier essai serait le suivant : plutôt que d'effectuer une opération XOR de M et K, pourquoi ne pas effectuer le XOR du message M avec une *fonction* de la clé K et du message M ? Ceci donnerait le message chiffré suivant :

$$C = M \oplus f(K, M)$$

où f est donc une fonction qui à deux séquences de n bits K et M fait correspondre une autre séquence de n bits $f(K, M)$. Si la fonction f est suffisamment compliquée (pour être précis, on parle de fonction "non-linéaire"), cette opération combine le message M et la clé K d'une façon beaucoup plus intriquée que le système de clé à usage unique, et permet donc potentiellement de réutiliser la clé K pour chiffrer un autre message, sans qu'il soit cette fois-ci possible pour Eve d'effectuer une opération simple pour se débarasser de la clé K.

Oui, mais... Ce nouveau système a un gros défaut : Bob, même s'il connaît la clé K, n'est pas non plus capable de retrouver le message M ! Voilà donc une fausse bonne idée. Ceci dit, tout n'est pas à jeter. C'est Horst Feistel qui découvre le moyen de rendre cette idée applicable en pratique au moyen de l'algorithme suivant :

1. Commençons par découper le message M et la clé K chacun en deux parties égales :

$$M = (M_a, M_b) \quad \text{et} \quad K = (K_a, K_b)$$

Pour simplifier les notations, nous dirons que le message M et la clé K sont maintenant composés chacun de $2n$ bits, de sorte que M_a, M_b, K_a et K_b sont toutes des séquences de n bits.

2. Nous allons maintenant réutiliser la même fonction f que ci-dessus, mais sur chacune des deux parties du message M, pour obtenir successivement :

$$C_a = M_a \oplus f(K_a, M_b) \quad \text{puis} \quad C_b = M_b \oplus f(K_b, C_a)$$

Notez bien que ces deux opérations doivent s'effectuer dans cet ordre, car le résultat C_a de la première opération est utilisé pour calculer C_b.

3. Alice envoie le message chiffré $C = (C_a, C_b)$ à Bob. Si Eve intercepte le message C sans connaître la clé K, elle est toujours bien embêtée pour décrypter celui-ci. Mais Bob peut-il le déchiffrer, qui lui connaît la clé K ?

4. C'est là toute l'ingéniosité du système : Bob effectue à nouveau les *mêmes opérations qu'Alice, mais dans l'ordre inverse* :

$$D_b = C_b \oplus f(K_b, C_a) \quad \text{puis} \quad D_a = C_a \oplus f(K_a, D_b)$$

Qu'obtient-il ? Tout d'abord, observez que

$$D_b = C_b \oplus f(K_b, C_a) = (M_b \oplus f(K_b, C_a)) \oplus f(K_b, C_a) = M_b \oplus (f(K_b, C_a) \oplus f(K_b, C_a)) = M_b$$

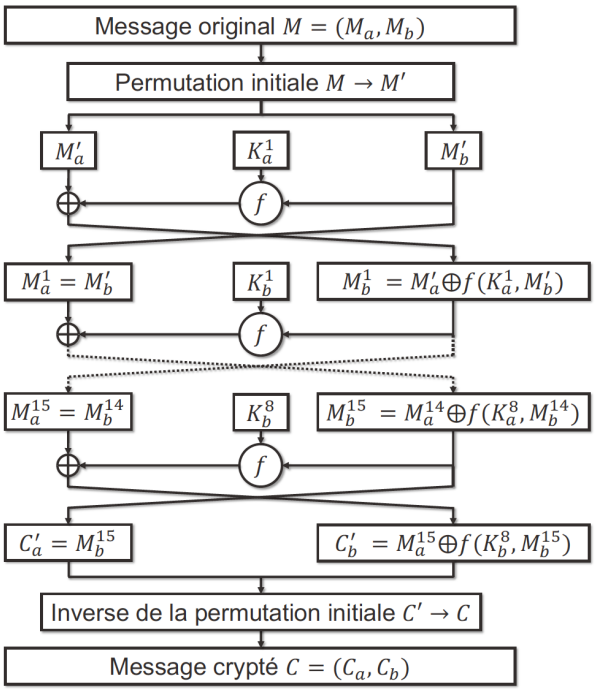
pour les mêmes raisons que lors de l'utilisation de la clé à usage unique. Aussi compliquée que soit la fonction f, elle disparaît avec l'opération XOR ! Donc la deuxième partie du message déchiffré D_b correspond bien à la deuxième partie du message d'origine M_b. Nous pouvons maintenant passer à la dernière étape :

$$\begin{aligned} D_a &= C_a \oplus f(K_a, D_b) = C_a \oplus f(K_a, M_b) \quad \text{(par ce qui vient d'être dit)} \\ &= (M_a \oplus f(K_a, M_b)) \oplus f(K_a, M_b) = M_a \oplus (f(K_a, M_b) \oplus f(K_a, M_b)) = M_a \end{aligned}$$

par le même mécanisme que précédemment. Ainsi Bob peut parfaitement déchiffrer le message envoyé par Alice, tandis qu'Eve ne peut rien faire.

L'avantage par rapport à la clé à usage unique est qu'Alice peut réutiliser ici ce système avec la même clé K pour chiffrer successivement deux messages M_1 et M_2, et qu'il sera très difficile pour Eve de se débarrasser de la clé K dans ce contexte, à cause de la fonction non-linéaire f.

Le système DES utilise quant à lui l'algorithme ci-dessus de manière répétée (8 fois, plus précisément), avec en plus une permutation de la séquence en entreée et son inverse en sortie. Il est illustré sur le schéma ci-dessous :



Notez que pour aussi complexe qu'il soit, le système DES a le même avantage que la machine Enigma et beaucoup d'autres systèmes cryptographiques : pour déchiffrer le message, il suffit de faire la même suite d'opérations (dans l'ordre inverse) que lors du chiffrement !

Finalement, notez qu'en pratique, $n = 32$, ce qui mène à une longueur totale de clé de 64 bits, ou 8 octets, pour la clé K ; cependant, dans chaque octet, un bit de est utilisé comme *bit de parité*, ce qui réduit la taille de la clé à $8 \cdot 7 = 56$ bits.

Attaque par force brute de la clé

Le système DES a finalement pu être décrypté par une attaque par force brute de sa clé K, d'une longueur de 56 bits. Combien d'opérations cela représente-t-il ?

En général, le nombre de possibilités pour construire une clé de longueur n, où chaque symbole peut être choisi dans un alphabet de k symboles, est donné par $\mathbf{k^n}$: veuillez noter que ce nombre augmente extrêmement rapidement avec n. Exemples :

- il y a 2^n possibilités pour une clé composée de n bits 0 ou 1 ;

- il y a 10^n possibilités pour une clé composée de n chiffres de 0 à 9 ;

- il y a 26^n possibilités pour une clé composée de n lettres majuscules de A à Z ;

- il y a 52^n possibilités pour une clé composée de n lettres minuscules et majuscules.

Voyons ce que ça donne concrètement pour certaines valeurs de n :

- clé composée de 6 chiffres : 10^6 possibilités

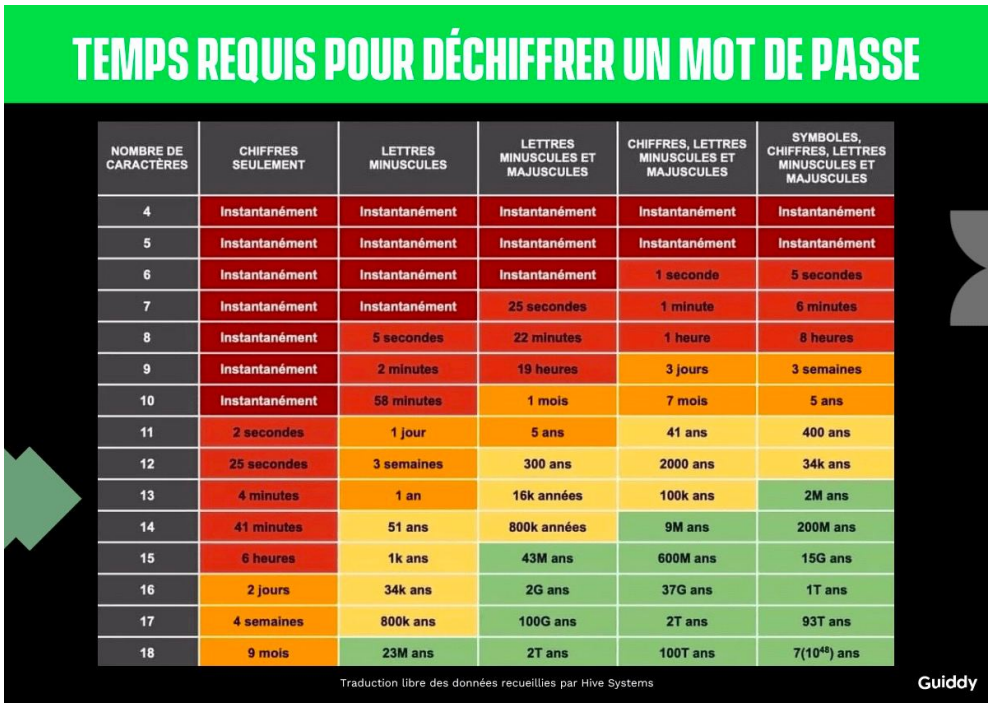
- clé composée de 6 lettres majuscules : $26^6 \simeq 3 \cdot 10^9$ possibilités

- clé composée de 6 lettres minuscules et majuscules : $52^6 \simeq 2 \cdot 10^{10}$ possibilités

- clé composée de 15 chiffres : 10^{15} possibilités

- clé composée de 56 bits : $2^{56} \simeq 7 \cdot 10^{16}$ possibilités

En faisant le parallèle entre ces clés et les mots de passe que vous utilisez tous les jours, ceci vous donne une idée du nombre d'opérations qu'il faut effectuer pour attaquer ce mot de passe par force brute. Le tableau de la page suivante résume bien la situation.



NOMBRE DE CARACTÈRES	CHIFFRES SEULEMENT	LETTRES MINUSCULES	LETTRES MINUSCULES ET MAJUSCULES	CHIFFRES, LETTRES MINUSCULES ET MAJUSCULES	SYMBOLES, CHIFFRES, LETTRES MINUSCULES ET MAJUSCULES
4	Instantanément	Instantanément	Instantanément	Instantanément	Instantanément
5	Instantanément	Instantanément	Instantanément	Instantanément	Instantanément
6	Instantanément	Instantanément	Instantanément	1 seconde	5 secondes
7	Instantanément	Instantanément	25 secondes	1 minute	6 minutes
8	Instantanément	5 secondes	22 minutes	1 heure	8 heures
9	Instantanément	2 minutes	19 heures	3 jours	3 semaines
10	Instantanément	58 minutes	1 mois	7 mois	5 ans
11	2 secondes	1 jour	5 ans	41 ans	400 ans
12	25 secondes	3 semaines	300 ans	2000 ans	34k ans
13	4 minutes	1 an	16k années	100k ans	2M ans
14	41 minutes	51 ans	800k années	9M ans	200M ans
15	6 heures	1k ans	43M ans	600M ans	15G ans
16	2 jours	34k ans	2G ans	37G ans	1T ans
17	4 semaines	800k ans	100G ans	2T ans	93T ans
18	9 mois	23M ans	2T ans	100T ans	$7(10^{48})$ ans

Sur ce tableau (et par ce qui précède, également), on voit que ce n'est pas tant en augmentant le nombre de symboles différents k dans l'alphabet qu'on rend un mot de passe plus sûr, mais bien en augmentant la taille n du mot de passe lui-même. A l'époque où le système DES a été conçu (dans les années 70), il était raisonnable de penser que 56 bits serait une longueur de clé suffisante pour résister à une telle attaque, mais tel n'était plus le cas vingt ans plus tard... Ce système a donc été remplacé par le système AES, qui utilise des clés de 128, 192 ou même 256 bits.