Lompo Issa
de Maillard Luc
Montiton Frédéric
Morreeuw Jean-François

Décryptage du chiffrement de Vigenère

Les faiblesses du chiffre carré

Maîtrise Informatique

Université Bordeaux I

Responsable de projet

Bruno Courcelle

Sommaire

1 Avant propos **2**

2 Introduction historique **3**

A - De l'apparition du chiffrement 3

B - De la mise en défaut des chiffrements monoalphabétiques 5

C - De la mise en équation du chiffrement de Vigenère 5

D - De la mise en défaut du chiffrement de Vigenère 6

E - De la mathématisation des méthodes de décryptage 9

F - De la modernisation des méthodes 11

3 Description des algorithmes utilisés **12**

A - Chiffrement et déchiffrement de Vigenère 12

B - Test de Friedrich Kasiski 12

B1 - Présentation de l'algorithme 12

B2 - Description de l'implantation 14

C - Test de Wolfe Friedman 15

C1 - Présentation de l'algorithme 15

C2 - Description de l'implantation 16

D - Analyse statistique des fréquences des lettres 16

D1 - Présentation de l'algorithme 16

D2 - Description de l'implantation 16

E - Méthode de Kerckhoff 17

E1 - Présentation de l'algorithme 17

E2 - Description de l'implantation 18

F - Indice de coïncidence mutuel 20

F1 - Présentation de l'algorithme 20

F2 - Description de l'implantation 20

4 Programmes sources **22**

A - Modules d'outils 22

A1 - Gestion des exceptions (`xcept`) 22

A2 - Gestion de la ligne de commande (`option`) 23

A3 - Structures de tri (`result/set`) 24

B - Modules des outils de décryptage 26

B1 - Méthode de Kasiski (`search`) 26

B2 - Test de Friedman (`ic`) 27

B3 - Méthode de Kerckhoff (`kh`) 29

B4 - Cryptanalyse de chiffres monoalphabétiques (`shift`) 32

C - Module de décryptage automatique 33

C1 - Principe de l'algorithme 33

C2 - Description de l'implantation 33

5 Exemples d'utilisation **35**

A - Utilisation depuis un shell Unix 35

A1 - Installation des logiciels 35

A2 - Transformation de textes 35

A3 - Chiffrement de Vigenère 36

A4 - Analyse statistique des fréquences 38

A5 - Méthode de Kasiski 39

A6 - Indice de coïncidence 40

A7 - Décalages relatifs entre les alphabets 41

A8 - Décryptage du chiffrement de Cæsar 42

A9 - Décryptage automatique 42

A10 - Logiciel de tests automatiques 43

B - Utilisation de l'interface HTML 44

B1 - Installation de l'interface 44

B2 - Présentation de l'interface 44

B3 - Exemple d'expérimentation 45

C - Utilisation de l'interface Xwindow 46

C1 - Installation de l'interface 46

C2 - Utilisation de l'interface 46

6 Analyse des résultats 47

A - Analyses fréquentielles des langues 47

B - Limites de validité des méthodes 51

B1 - Indice de coïncidence 51

B2 - Indice de coïncidence mutuel 51

B3 - Méthode de Kasiski 52

B4 - Décryptage automatique 52

C - Extensions possibles des sources 54

C1 - Structure de tri (`set`) 54

C2 - Minimisation de l'erreur d'un système 54

C3 - Tables fréquentielles des langues 54

D - Complexité des algorithmes 55

D1 - Chiffrement de Vigenère 55

D2 - Méthode de Kerckhoff 55

D3 - Méthode de Kasiski 56

D4 - Indices de coïncidence 56

D5 - Cryptanalyse du chiffrement de Cæsar 57

D6 - Structure de résultat 58

D7 - Structure de tri dynamique 58

Annexe A **59**

A.1 - Installation des logiciels mode texte 59

A.1.1 - Compilation complète 59

A.1.2 - Compilation réduite 60

A.2 - Utilisation des logiciels mode texte 60

A.2.1 - xtract 60

A.2.2 - vigenere 60

A.2.3 - frequency 61

A.2.4 - doubloon 61

A.2.5 - coincide 62

A.2.6 - minter 62

A.2.7 - offset 63

A.2.8 - decipher 63

A.2.9 - autotest 64

A.3 - Utilisation de l'interface HTML 64

A.3.1 - Installation de l'interface 64

A.3.2 - Utilisation de l'interface 65

A.4 - Utilisation de l'interface Xwindow 65

A.4.1 - Installation de l'interface 65

A.4.2 - Utilisation de l'interface 65

Annexe B **66**

Chapitre 1

Avant propos

Si l'informatique tend à devenir un outil à la fois ludique et pratique, il reste des domaines où seules les capacités d'un calculateur peut permettre de réaliser des opérations. Les méthodes de chiffrement, souvent présentées sous l'aspect de méthodes automatisées, ont conduit à l'invention de nombreuses machines mécaniques ou électriques de chiffrement telles que la machine allemande `Enigma`. Paradoxalement, les méthodes de décryptage, qui recherchent à l'aveugle la clé de chiffrement d'un texte, ont rarement été considérées comme des méthodes automatisables. Or, c'est justement l'informatique qui, en raison de la somme d'informations qu'il est possible de gérer simultanément, peut permettre de justifier si des méthodes particulières sont, ou non, adpatées à décrypter un texte chiffré.

Sur un chiffrement donné, le chiffrement de Vigenère, le but du projet sera par conséquent d'implanter les méthodes de décryptage pour pouvoir en tester les limites. Finalement, les possibilités offertes par les méthodes de décryptage seront suffisantes pour espérer réaliser un logiciel capable seul de déchiffrer un texte. C'est la genèse de ce logiciel que ce mémoire vous propose de découvrir ...

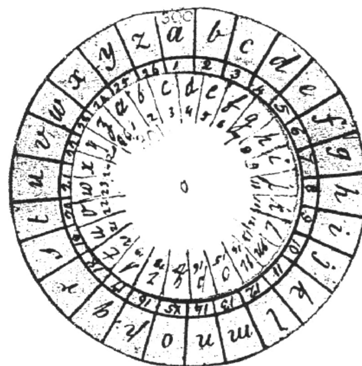
Chapitre 2

Introduction historique

Le but de cette introduction est de présenter l'historique des découvertes ayant permis le décryptage du chiffrement de Vigenère, tout en rappelant les principes de ces méthodes.

A - De l'apparition du chiffrement

Si le chiffrement par décalage des lettres d'un texte est connu depuis longue date sous le nom de *chiffrement de Cæsar*, l'idée d'un chiffrement polyalphabétique apparaît pour la première fois en **1466** dans un texte de l'architecte **Léone Battista Alberti** [?]. Dans ce texte, publié en 1468, Léone Battista Alberti présente le premier disque de chiffrement, directement inspiré des outils de calcul qu'il utilisait en architecture.



Disque de chiffrement

Indépendamment de cette invention, un abbé Bénédictin, **Johannes Trithemius**, envisage en **1508** l'utilisation d'un tableau pour un chiffrement similaire. Le texte, publié en 1518, esquisse vaguement la possibilité d'utiliser de façon répétée une clé. Ce n'est cependant qu'en **1553** que **Giovanni Battista Belaso** met en forme le principe du contrôle du chiffrement par l'utilisation d'une clé unique.

Finalement, **Giovanni Battista della Porta** permet le lien en **1563** entre le chiffrement par disque et le chiffrement par tableau. En **1586**, un diplomate français de la cour d'Henry III de France, **Blaise de Vigenère** (1523-1596), publie dans son *Traité des chiffres* le principe du *chiffre carré* connu plus tard sous le nom du chiffrement de Vigenère. Ce n'est cependant qu'en **1641** que la cryptographie prend ses lettres de noblesses avec la publication par **Bishop Wilkins**, beau frère de Olivier Cromwell et parmi les fondateurs de la Royal Society, de *The Secret and Swift Messenger*, avec principalement l'introduction dans la langue anglaise du terme *cryptographia*, et la vulgarisation de la méthode de chiffrage dite de Vigenère.

L'un des défauts majeurs des chiffrements monoalphabétiques est leur vulnérabilité à une analyse statistique des fréquences. Le principal travail des cryptographes a donc été d'immuniser le chiffrement contre une analyse fréquentielle, et la méthode employée pour le chiffrement de Vigenère consiste à utiliser plusieurs alphabets. Le tableau ci-dessous correspond aux 26 colonnes de chiffrement de Cæsar. Ainsi, si la clé est `F`, le chiffrement correspond à la cinquième colonne et la transformation de `A` est `F`, celle de `B` est `G` et ainsi de suite. Par exemple, le chiffrement de la lettre `N` est la lettre `S`. Le principe du chiffrement de Vigenère est à sa base d'utiliser un mot comme clé. Chacune des lettres de cette clé est utilisé successivement pour référencer la colonne de chiffrement.

	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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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

Si le chiffrement de Vigenère correspond, comme il sera expliqué plus loin, à une addition modulo 26 dans l'alphabet projeté sur les entiers de 0 à 25, le chiffrement correspondant à une soustraction et portant le nom du *chiffrement de Beaufort* apparaît en **1710** par **Giovanni Sestri**. D'une manière générale, les chiffrements seront successivement réinventés au fil du temps. Pour l'anecdote, **Charles L. Dogson**, plus connu sous le nom de **Lewis Carroll**, réinvente les deux chiffrements en **1868** et les perfectionne pour imaginer son *chiffrement du télégraphe* [**?**].

B - De la mise en défaut des chiffrements monoalphabétiques

Depuis au moins **1831**, des statisticiens tels le Belge **Lambert Adolphe Jacques Quetelet** publient des tables de fréquences de lettres et bigrammes des langues communes. Ces tables de fréquences permettent alors par simple comparaison de décrypter les chiffrements monoalphabétiques en comparant les *symétries de position* dans la répartition. Voici par exemple les tables publiées en 1840 par Charles Vesin de Romanini pour le Français et l'Allemand :

```
eusranilotdpmcbvghxqfjyzkw
enrisdutaghlobmfzkcwvjpqxy
```

Si l'ordre des lettres n'a pas une importance capitale, en fait ce sont les groupes de lettres qui sont importants. En considérant les 2 ou 3 premières lettres, `eus` pour le Français ou `enr` pour l'Allemand, il est possible de déterminer le décalage d'un texte chiffré. En effet, la répartition du texte chiffré présente 2 ou 3 lettres de fréquences importantes et dont les distances relatives correspondent généralement aux distances relatives de référence. Il suffit par conséquent de comparer ces distances pour en déduire le décalage utilisé pour chiffrer le texte.

C - De la mise en équation du chiffrement de Vigenère

Si le chiffrement de Vigenère semble d'un principe simple actuellement, il n'en était pas de même dans le début de ce siècle. En effet, ce n'est qu'en **1801** que **Gauss** introduit dans *Disquitiones Arithmeticae* la notion de modulo. Plus tard, en **1846**, **Joseph Serret** publie un livre d'algèbre basé sur les travaux en cours d'**Evariste Galois** à la Sorbonne, et ce n'est qu'en **1870** que **Camille Jordan** et **Otto Hölder** terminent la théorie des groupes dans leur *Traité des substitutions et des Equations Algébriques.*

Au court du mois de février 1846, l'Anglais **Charles Babbage** (1791-1871) tente par défi le décryptage d'un texte. Expert en cryptographie de la Royal Navy pour le compte de son ami **Rear-Admiral Sir Francis Beaufort** (1774-1857), Charles Babbage possède dans sa bibliothèque des ouvrages tels celui de Gauss ou de Bishop Wilkins, et c'est également Charles Babbage qui fournit à Lambert Adolphe Jacques Quetelet les tables de fréquences. Le **20 mars 1846** à 1h30 du matin, Babbage décrit pour la première fois le chiffrement de Vigenère sous forme mathématiques, expression dont l'origine se trouve initialement en 1 pour la lettre `A`.

```
Cypher=Key+Translation-1
Translation=Cypher-Key+1
```

D - De la mise en défaut du chiffrement de Vigenère

Symétrie des positions utilisée par Charles Babbage en février 1846

Lorsqu'il tente de décrypter le texte en 1846, Charles Babbage étudie la fréquence des lettres en supposant différentes longueurs de clé. Cette méthode généralisant le principe de la *symétrie des positions* lui permet de constater l'utilisation de trois clés dans le texte complet. Ce n'est cependant qu'en **1883** que le Français **Auguste Kerckhoff** divulgue le principe dans *La Cryptographie Militaire* [?], principe consistant à comparer l'empreinte dans la table des fréquences des lettres pour en déduire le décalage relatif entre chacune des lettres de la clé utilisée.

Analyse de la périodicité de la clé par Charles Babbage le 18 septembre 1854

La méthode, connue sous le nom de *méthode de Kerckhoff*, aide à la recherche de la clé, ainsi que de la longueur de cette clé sous certaines limites. En effet, le calcul justifie a posteriori la longueur de la clé intuitée, nécessitant de longues étapes inutiles. Le **18 septembre 1854**, Charles Babbage observe que deux segments identiques du texte clair distants de x positions sont chiffrés de la même manière dès que $x = 0\,(mod\,|clé|)$. En cherchant les répétitions dans le texte chiffré, il est possible de conjecturer que la longueur de la clé divise le plus grand diviseur commun des ces distances. Cette découverte, assurant la suprématie de l'Angleterre en cryptanalyse, est gardée secrète, et en automne l'Angleterre, la France, la Turquie et la Sardaigne déclarent la guerre à la Russie. Ce n'est finalement qu'en **1863** qu'un ancien major de l'armée Prusse, **Frederich Kasiski** [?], publie la méthode dans son livre de 95 pages *Die Geheimschriften und die Dechiffrir-Kunst.*

Dans l'exemple suivant, extrait d'*Alice aux pays des merveilles*, les lettres inscrites en majuscules montrent un certain nombre de suites de lettres répétées. La recherche des répétitions donne les distances de 160 pour `sth`, 120 pour `hdmjdmobwc`, 110 pour `jpe`, 15 pour `phd`, ou encore 10 pour `azp` ... La longueur de la clé a donc une forte probabilité d'être 5. L'analyse statistique confirme les résultats et permet de vérifier que les répartitions des fréquences correspondent, décalées respectivement par les lettres de `LEWIS`.

```
wakeupalicedearsaidh    hegmmaehquphaijdeelz
ersisterwhywhatalong    pvoqkeinezjadillpkvy
sleepyouvehadohiveha    dpamhjsqdwsezwztzaps
dsuchacuriousdREAmsA    owqkzlgqzazyolJPEiaS
IDaliceandshetolDHER    THwtaniwvvdlabgwHDMJ
SISTERaswellasshecou    DMOBWCeoewwpwaksiywm
ldrememberthemallthe    whnmepqxmjelauswpppw
sestrangeadventureso    diobjlrcmsozavlfvaag
fhersthatyouhavejust    qlazkelwbqzydinpnqal
beenJPEdingaboutandw    miavJPEzqfrexwmeejlo
henSHEhadfiniSHEDHER    sijAZPlwlxtreAZPHDMJ
SISTERkissEDHerandsA    DMOBWCoeakPHDmjlrzaS
IDitwasacuriousdream    THebolwwkmcmkckovaie
dearcertainlybutnowr    oiwzupvpiaypujmerkej
unintoyourteaitsgett    frevlzckcjeiwqldkabl
inglate                 trctsei
```

le tableau suivant contient les probabiliés d'apparition d'une lettre dans l'ensemble formé des lettres du texte, prises toutes les 5 positions. Il y a par conséquent 5 répartitions fréquentielles, constituant les 5 colonnes du tableau. Lorsque la longueur de la clé est bien intuitée, les tables fréquentielles sont proches à un décalage près. La lettre la plus fréquente dans la première colonne est le `P`. La lettre la plus fréquente dans la seconde colonne est le `I`. Il est donc permis de supposer que le `P` et le `I` sont les chiffrements de la même lettre, respectivement pour le premier et le second alphabet. La comparaison avec les fréquences dans la langue anglaise laisse à penser que cette lettre soit le `E`, d'où les deux premières lettres de la clé : `L` et `E`.

A	L	0.080	E	0.145	W	0.131	I	0.081	S	0.098
B	M	0.016			X	0.032	J	0.016		
C	N	0.016	G	0.016	Y	0.016	K	0.032	U	0.032
D	O	0.064	H	0.112	Z	0.049	L	0.065	V	0.016
E	P	0.161	I	0.145	A	0.180	M	0.131	W	0.081
F	Q	0.016							X	0.016
G	R	0.016	K	0.016	C	0.032			Y	0.016
H	S	0.048	L	0.080	D	0.081	P	0.032	Z	0.098
I	T	0.080	M	0.048	E	0.081	Q	0.065	A	0.049
J			N	0.016						
K			O	0.016	G	0.016				
L	W	0.064	P	0.080	H	0.016	T	0.032		
M			Q	0.016	I	0.016	U	0.016	E	0.032
N	Y	0.016	R	0.096	J	0.032	V	0.081	F	0.016
O	Z	0.048	S	0.016	K	0.065	W	0.049	G	0.032
P	A	0.016							H	0.016
Q										
R	C	0.048	V	0.064	N	0.032	Z	0.049	J	0.163
S	D	0.112	W	0.032	O	0.098	A	0.131	K	0.081
T	E	0.112			P	0.032	B	0.114	L	0.098
U	F	0.032	Y	0.032	Q	0.065	C	0.032	M	0.081
V			Z	0.032			D	0.016	N	0.016
W	H	0.016	A	0.016			E	0.049	O	0.032
X										
Y	J	0.032	C	0.016	U	0.016			Q	0.016
Z										

Comparaison des alphabets décalés

E - De la mathématisation des méthodes de décryptage

Les procédés construits sur les fréquences des lettres utilisent les distances entre les groupes de lettres les plus significatives. En **1920**, **Wolfe Friedman**[?] introduit la notion d'*indice de coïncidence mutuel*, permettant une mise en forme des méthodes. Noté $I_c(x)$, l'indice de coïncidence d'un texte x est la probabilité que deux caractères soient égaux. Cette probabilité peut s'écrire, en notant p_i la probabilité d'apparition de la $i^{ème}$ lettre de l'alphabet [1] :

$$I_c(x) \approx \sum_{i=0}^{25} p_i^2$$

1. En réalité la probabilité de la seconde apparition d'une lettre n'est pas exactement égale à celle de sa première apparition. L'approximation réalisée en omettant ce détail est cependant acceptable. D'un point de vue théorique, cela nécessite une taille de texte assez longue pour pouvoir négliger l'erreur. Néanmoins, dans le cas où cette condition n'est pas vérifiée, d'autres facteurs interviennent nécessitant un aménagement de la méthode ...

La fréquence d'apparition d'une lettre dépend de la langue utilisée. Les probabilités n'étant pas identiques, l'indice de coïncidence d'un texte dans une langue est supérieur à la valeur dans un texte où toutes les lettres auraient la même probabilité d'apparition 1/26. La valeur de l'indice de coïncidence diffère légèrement selon les langues considérées. Voici quelques exemples d'indices calculés sur des textes contemporains :

- $I_c\,(esperanto) = .06883$
- $I_c\,(su\acute{e}dois) = .07063$
- $I_c\,(allemand) = .07187$
- $I_c\,(norv\acute{e}gien) = .07350$
- $I_c\,(espagnol) = .07393$
- $I_c\,(francais) = .07405$
- $I_c\,(italien) = .07565$

Pour un texte aléatoire cette valeur est de .03846 (1/26). Dans la mesure où une permutation des lettres ne change pas la valeur de la somme, et que la valeur de cette somme permet de distinguer un texte clair d'un texte aléatoire, ce calcul permet de réaliser proprement la méthode consistant à rechercher si, pour une longueur de clé supposée, la fréquence des lettres est étalée ou non et donc à justifier le bon choix de la longueur.

L'indice de coïncidence mutuel est une généralisation du principe décrit ci-dessus, en calculant la probabilité pour que les alphabets correspondent dans deux textes. Il est donc nécessaire pour cela réaliser le calcul pour les 26 décalages possibles, et l'expression de l'indice de coïncidence mutuel peut s'écrire sous la forme suivante :

$$MI_c\,(x,y) \approx \sum_{i=0}^{25} p_i\,(x)\,.p_{(i+d\acute{e}calage)(mod\,26)}\,(y)$$

L'indice de coïncidence mutuel permet d'une part de rechercher les décalages relatifs des lettre de la clé, et donc de passer d'un chiffrement polyalphabétique à un chiffrement monoalphabétique. D'autre part, le calcul permet en utilisant des textes de référence de déterminer par la suite la lettre utilisée pour ce chiffrement monoalphabétique.

F - De la modernisation des méthodes

Le chiffrement à partir du disque, aussi bien que celui construit autour du tableau de chiffrement, utilisent un système automatisé rudimentaire prenant en entrée la lettre courante et donnant en sortie la lettre chiffrée correspondante. De ce point de vue, des machines chiffrant automatiquement telles que la machine `Enigma`[?] [2] utilisée par l'armée Allemande pendant la seconde guerre mondiale ne sont pas surprenantes.

Plusieurs programmes ont été réalisés pour implanter les méthode de décryptage. L'un de ces logiciels, programmée par Ralph Morelli [3], professeur au *Trinity College, Hartford, Connecticut* est construit sur la méthode de Kasiski, et sur la cryptanalyse des bandes de texte correspondant à chaque lettre de la clé. Une seconde implantation, `breakvig` a été réalisée à l'*University of California, San Diego*, n'utilisant que le test de Friedman pour trouver la longueur de la clé [4]. Un troisième logiciel fonctionnant sur le même principe, `solvevig`, a été réalisé par Mark Riordan [5] du *Michigan State University.*

La réalisation du projet de maîtrise décrit dans ce mémoire a pour motivation d'évaluer, à l'aide d'une modélisation informatique, les possibilités de décryptage offertes par l'utilisation conjointe de la totalité des méthodes explicitées dans ce chapitre. En conséquence aucun existant n'a pu être utilisé en dehors de la description des méthodes originales. Si l'intitulé initial du sujet ne porte que sur la réalisation des outils de décryptage pour pouvoir expérimenter à la main, celui-ci a été élargi à une cryptanalyse automatique. Les bons résultats obtenus par le logiciel sont en effet les meilleures preuves des possibilités offertes par l'utilisation conjointe de l'ensemble des méthodes.

2. L'ouvrage a été partiellement traduit en Français par N. Zimmermann[?]
3. `http://www2.trincoll.edu/~rmorelli/`, `ralph.morelli@mail.trincoll.edu`
4. `http://sdcc14.ucsd.edu/~ma187s`
5. `http://www.msu.edu/`, `mrr@ripem.msu.edu`

Chapitre 3

Description des algorithmes utilisés

Ce chapitre présente les adaptations des méthodes pour permettre le décryptage du chiffrement de Vigenère. Le chiffrement étant un chiffrement polyalphabétique, la première étape consiste à rechercher la longueur de la clé. Deux principes sont utilisés, d'abord le test de Kasiski qui recherche les répétitions de suites de lettres, ensuite le test de Friedman qui permet soit de confirmer le premier test, soit de donner une solution lorsque le premier test échoue. La seconde étape est, connaissant la longueur de la clé, de déplacer le problème vers la cryptanalyse d'un chiffrement monoalphabétique. Cette seconde étape est réalisée par la méthode de Kerckhoff. Pour terminer, l'indice de coïncidence mutuel permet le décryptage du chiffrement monoalphabétique, achevant la cryptanalyse.

A - Chiffrement et déchiffrement de Vigenère

Le chiffrement de Vigenère suit une méthode humainement utilisable à partir d'un simple tableau de décalages. Un tel procédé ne pose bien sûr aucune difficulté autant sur un plan théorique que pratique. Le chiffrement est effectué linéairement sur le texte en décalant par addition ou soustraction les lettres dans l'alphabet pour respectivement chiffrer et déchiffrer.

B - Test de Friedrich Kasiski

B1 - Présentation de l'algorithme

a - Algorithme initial

Le principe du test de Kasiski repose sur la remarque que des mots apparaissants dans un texte peuvent être substitués de la même manière. Plus précisément, si une clé de chiffrement a une longueur de n caractères, la probabilité pour que deux occurrences d'un mot soit chiffré avec les mêmes lettres est au minimum de $1/n$.

La méthode consiste à repérer dans le texte à décrypter des suites de lettres identiques, et à noter les distances entre les positions de ces suites. Pour chaque couple trouvé, l'hypothèse est que la répétition n'est pas fortuite, ce qui a pour conséquence que cette distance est un multiple de la longueur de la clé de chiffrement. L'ensemble de ces distances permet d'intuiter, si ce n'est la longueur de la clé, du moins un multiple raisonnable de cette longueur.

La difficulté de l'algorithme repose sur la possibilité qu'a une distance d'être fortuite. Dans le cadre d'un simple outil de calcul, des résultats bruts sont admissibles pour un humain qui se charge ultérieurement de pouvoir les remettre en doute. En revanche, l'utilisation de résultats par un autre logiciel nécessite une méthode plus raffinée assurant leur validité.

b - Algorithme amélioré

La modification de l'algorithme a pour but de permettre d'extraire de l'ensemble des résultats ceux qui ont une réelle importance. La première étape consiste à trier dans une structure les distances, associées à leur nombre d'apparition. La structure est dynamique, et permet d'accéder à un de ces éléments par un index représentant la position de l'élément dans l'ensemble trié par ordre décroissant. A un instant donné, d_0 est la plus grande distance rencontrée, d_1 la seconde plus grande distance ... Etant fixée une distance d_i dans la structure dont la taille est notée n, l'algorithme calcule les *pgcd* de d_i et d_{i+1}, d_i et d_{i+2}, jusqu'au *pgcd* de d_i et d_{n-1}. Chaque *pgcd* est à son tour inséré dans la structure associé au même coefficient que d_i, et par conséquent positionné dynamiquement. Comme $pgcd\left(d_i, d_{j\in[i+1,n-1]}\right) \leq d_j$, l'ordre n'est pas bouleversé pour ce qui concerne les calculs réalisés précédemment. L'algorithme est ensuite réitéré pour d_{i+1}, et jusqu'à épuisement du contenu de la structure. L'utilisation des *pgcd* précédents pour le calcul de nouveaux *pgcd* permet ainsi à la longueur de la clé de remonter en fin calcul.

Dans l'exemple décrit précédemment, la recherche des répétitions donne les distances de 160 pour `sth`, 120 pour `hdmjdmobwc`, 110 pour `jpe`, 15 pour `phd`, ou encore 10 pour `azp`. Un essai sur ces valeur permet d'illustrer le pricipe de fonctionnement de l'algorithme :

```
wakeupalicedearsaidh    hegmmaehquphaijdeelz
ersisterwhywhatalong    pvoqkeinezjadillpkvy
sleepyouvehadohiveha    dpamhjsqdwsezwztzaps
dsuchacuriousdREAmsA    owqkzlgqzazyolJPEiaS
IDaliceandshetolDHER    THwtaniwvvdlabgwHDMJ
SISTERaswellasshecou    DMOBWCeoewwpwaksiywm
ldrememberthemallthe    whnmepqxmjelauswpppw
sestrangeadventureso    diobjlrcmsozavlfvaag
fhersthatyouhavejust    qlazkelwbqzydinpnqal
beenJPEdingaboutandw    miavJPEzqfrexwmeejlo
henSHEhadfiniSHEDHER    sijAZPlwlxtreAZPHDMJ
SISTERkissEDHerandsA    DMOBWCoeakPHDmjlrzaS
IDitwasacuriousdream    THebolwwkmcmkckovaie
dearcertainlybutnowr    oiwzupvpiaypujmerkej
unintoyourteaitsgett    frevlzckcjeiwqldkabl
inglate                 trctsei
```

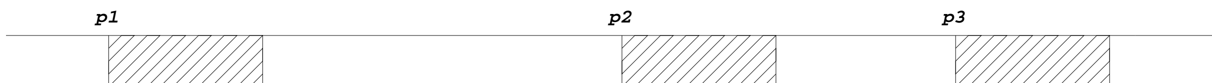
structure initiale	(160,1)	(120,1)	(110,1)	(15,1)	(10,1)		
$pgcd(160,120) = 40$	(160,1)	(120,1)	(110,1)	(40,1)	(15,1)	(10,1)	
$pgcd(160,110) = 10$	(160,1)	(120,1)	(110,1)	(40,1)	(15,1)	(10,2)	
$pgcd(160,40) = 40$	(160,1)	(120,1)	(110,1)	(40,2)	(15,1)	(10,2)	
$pgcd(160,15) = 5$	(160,1)	(120,1)	(110,1)	(40,2)	(15,1)	(10,2)	(5,1)
$pgcd(160,10) = 10$	(160,1)	(120,1)	(110,1)	(40,2)	(15,1)	(10,3)	(5,1)
$pgcd(160,5) = 5$	(160,1)	(120,1)	(110,1)	(40,2)	(15,1)	(10,3)	(5,2)
$pgcd(120,110) = 10$	(160,1)	(120,1)	(110,1)	(40,2)	(15,1)	(10,3)	(5,2)
$pgcd(120,40) = 40$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,3)	(5,2)
$pgcd(120,10) = 10$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,4)	(5,2)
$pgcd(120,5) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,4)	(5,3)
$pgcd(110,40) = 10$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,5)	(5,3)
$pgcd(110,15) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,5)	(5,4)
$pgcd(110,10) = 10$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,6)	(5,4)
$pgcd(110,5) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,6)	(5,5)
$pgcd(40,15) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,6)	(5,8)
$pgcd(40,10) = 10$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,8)
$pgcd(40,5) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,11)
$pgcd(15,10) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,12)
$pgcd(15,5) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,13)
$pgcd(10,5) = 5$	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,22)
structure finale	(160,1)	(120,1)	(110,1)	(40,3)	(15,1)	(10,9)	(5,22)

B2 - Description de l'implantation

a - Choix d'implantation

Une première solution naturelle serait de rechercher pour chaque suite de m lettres les éventuelles réapparitions dans la suite du texte. Or, si la longueur du texte est de n lettres, la complexité d'une telle méthode serait en n^2. Un texte pouvant raisonnablement avoir une taille de plusieurs milliers lettres, ceci n'est pas acceptable.

La solution retenue repose sur l'utilisation d'une structure dans laquelle sont enregistrées les suites de m lettres ainsi que leurs positions. Lors du parcours du texte, si la suite de lettres n'a pas déjà été rencontrée, celle-ci est insérée dans la structure. Si la suite est présente dans la table, la position enregistrée permet de calculer la distance avec la nouvelle position. La nouvelle position devient alors la position de référence dans la table pour la suite de lettres, et ce de manière à minimiser les distances calculées par la suite. En effet, dans le cas où la suite de lettres est de nouveau rencontrée, la distance est calculée par rapport à la plus proche apparition. Le graphe ci-dessous représente les positions de trois occurrences d'un même mot dans un texte, respectivement en position p_1, p_2 et p_3, pour illustrer la méthode.



A la lecture de la première occurrence, la suite de lettres est enregistrée dans la structure des mots lus, associée à sa position p_1. Lors de la lecture de la seconde occurrence, la suite ayant déjà été rencontrée, la distance $p_2 - p_1$ est enregistrée dans la structure mémorisant les distances et la position de la suite est modifiée dans la structure des mots pour prendre la nouvelle valeur p_2. Enfin, lors de la lecture de la troisième occurrence, la distance $p_2 - p_3$ est enregistrée et la position devient $p_3 \ldots$

Lors du calcul des *pgcd*, les valeurs calculées sont insérées dans la structure accompagnées du coefficient de la plus grande des deux positions ayant servi à l'obtenir. Cette méthode permet à la longueur de la clé de se distinguer de ses multiples plus rapidement, tout en réduisant l'impact des entrées fortuites. Une distance fortuite ayant une faible présence dans l'ensemble des distances, les valeurs erronées qui en découlent ont de même une faible présence.

b - Problèmes rencontrés

Bien que l'algorithme minimise autant que possible les distances insérées dans la structure, il arrive toutefois que certaines valeurs de taille importante, fortuites ou non, soient présentes. Ces distances sont par la suite utilisées pour calculer les *pgcd*, pouvant engendrer des temps de calcul relativement longs. La solution apportée a été de pouvoir définir une distance de recherche maximale. Une limite faible réduit le nombre de répétitions, mais permet de conserver un temps de calcul raisonnable.

La distance maximale a également un autre impact, impact sur la mémoire utilisée. En effet, les suites de m lettres étant conservées dans une structure, potentiellement il peut se trouver $n - m$ suites de m lettres ainsi allouées. En limitant la taille de recherche dans un bloc de p lettres, la taille réservée est alors limitée à $p - m$ suites de lettres. Pour permettre cette limitation de mémoire, la distance maximale n'est pas implantée lors de l'insertion des distances comme cela aurait pu être réalisé, mais dans la recherche des répétitions même en découpant le texte en blocs.

C - Test de Wolfe Friedman

C1 - Présentation de l'algorithme

Le test de Wolfe Friedman repose sur la fréquence des lettres d'une langue. Un texte clair se distingue d'une suite de lettre aléatoire par l'importance relative de la fréquence de certaines lettres par rapport à d'autres lettres. Si la longueur n de la clé est trouvée, alors l'ensemble des lettres prises toutes les n positions possède de telles disparités fréquentielles. Si la longueur n'est pas la bonne, la fréquence des lettres se rapproche de celle d'un texte aléatoire.

L'algorithme enregistre le nombre d'apparition des lettres pour une longueur de clé donnée, et en déduit l'indice de coïncidence de ces lettres. Ce calcul est réalisé sur chacun des ensembles de lettres distantes de n positions, donc sur n ensembles. L'indice final est la moyenne de ces indices, et il lui est soustrait l'indice correspondant à un texte aléatoire pour ramener l'origine à 0.

C2 - Description de l'implantation

L'indice de coïncidence est en réalité un cas particulier de l'indice de coïncidence mutuel, cependant, pour des raisons décrites en page 20, l'indice de coïncidence mutuel est implanté avec une légère variante.

D - Analyse statistique des fréquences des lettres

D1 - Présentation de l'algorithme

L'analyse fréquentielle des lettres n'est pas utile en elle même, mais en tant qu'outil pour d'autres algorithmes. Le travail consiste à enregistrer dans une structure les suites de m lettres du texte, et à afficher les fréquences correspondantes.

D2 - Description de l'implantation

Deux méthodes peuvent être envisagées. La première consiste à conserver dans une structure les suites de lettres rencontrées. L'avantage de cette méthode est d'économiser de la mémoire en supposant que beaucoup de combinaisons de lettres ne se rencontrent jamais. La seconde méthode est de positionner dans un tableau ayant une case par suite de lettres le nombre d'apparition. L'avantage de cette méthode est d'être bien plus rapide que la première puisqu'il n'y a aucun tri à réaliser.

En fait, les deux méthodes sont utilisées. La première est utilisée pour permettre à un utilisateur d'avoir le nombre d'apparitions d'une suite de lettres de longueur quelconque. La seconde est employée lors de l'utilisation de cette méthode pour des suites de lettres de longueur raisonnable.

E - Méthode de Kerckhoff

E1 - Présentation de l'algorithme

a - Méthode initiale

La méthode de Kerckhoff repose sur le calcul de l'indice de coïncidence mutuel. Ce n'est pas à proprement parler un algorithme, mais plutôt une méthode de recherche. La mise en application de cette méthode conduit, pour une clé de n lettres, à $\frac{n(n-1)}{2}$ équations. Chaque équation décrit le décalage relatif entre deux lettres de la clé, et ce décalage peut prendre 26 valeurs. L'indice de coïncidence mutuel permet, pour chaque équation, de pouvoir connaître les décalages les plus probables. Finalement, le résultat de la méthode est un système de $13n\,(n-1)$ équations, incompatibles entre elles, et ayant des probabilités de justesse variables.

Si la longueur de la clé est par exemple de 3 pour un alphabet de 4 lettres $\{a, b, c, d\}$, et les fréquences des lettres pour chacun des décalages respectivement f_{a1}, f_{a2}, f_{a3}, f_{b1}, f_{b2} ... la méthode de recherche peut être détaillée comme suit, avec en premier lieu la mise en forme des informations recherchées :

- Quel est le décalage relatif entre le premier et le second alphabet ?
- Quel est le décalage relatif entre le second et le troisième alphabet ?

Lorsque deux alphabets correspondent, l'indice des coïncidence est plus important. Les valeurs de ces indices peuvent donc servir à qualifier la *justesse* d'une hypothèse sur le décalage entre deux alphabets. L'ensemble des hypothèses conduit alors à un système d'équations entre les décalages relatifs d_{ij} d'un alphabet i et d'un alphabet j, où chaque équation est associée à un coefficient décrivant la probabilité pour que l'équation soit juste :

- $d_{12} = 0$ associé au coefficient MI^0_{c12}
- $d_{12} = 1$ associé au coefficient MI^1_{c12}
- $d_{12} = 2$ associé au coefficient MI^2_{c12}
- $d_{23} = 0$ associé au coefficient MI^0_{c23}
- $d_{23} = 1$ associé au coefficient MI^1_{c23}
- $d_{23} = 2$ associé au coefficient MI^2_{c23}
- $d_{13} = d_{12} + d_{23} = 0$ associé au coefficient MI^0_{c13}
- $d_{13} = d_{12} + d_{23} = 1$ associé au coefficient MI^1_{c13}
- $d_{13} = d_{12} + d_{23} = 2$ associé au coefficient MI^2_{c13}

Si les calculs permettent d'avoir $\frac{n(n-1)}{2}$ équations déclinées en autant de solutions que de décalages possibles, $3 * 2$ dans l'exemple ici, en fait seulement n sont libres. Ces équations seront nommées *équations principales*, les équations restantes *équations secondaires*. Si un choix est effectué sur les équations principales, il n'y a plus de liberté sur les équations secondaires. Un choix peut donc être considéré comme une extraction d'un sous-système du système initial. Le produit des coefficients de chaque équation du sous-système permet de juger de la probabilité de la sélection.

b - Mise en place d'un algorithme

L'algorithme de la méthode repose principalement sur la recherche des solutions permettant de minimiser l'erreur dans le système d'équations. L'erreur du système pour une solution données est ici le produit des justesses des équations, obtenues en projetant le système suivant la solution. D'un point de vue théorique, il suffirait d'essayer l'ensemble des décalages possibles entre deux lettres consécutives en les reportant dans les équations pour obtenir l'ensemble de tous les sous-systèmes possibles. Malheureusement, pour une clé de n lettres, la complexité d'un tel procédé serait de 26^{n-1} ce qui n'est bien sûr pas raisonnable.

La solution à ce problème repose sur le fait que les coefficients décrivant la justesse des équations sont connus. L'ensemble des possibilités pour une équation peut être ordonné, et les solutions testées sont choisies par rapport à cet ordonnancement. Le nombre de tests est toujours de 26^{n-1} en théorie, cependant uniquement un nombre limité de solutions est réellement utile. En parcourant les équations en partant des solutions les plus probables vers les solutions les moins probables, il est alors possible d'interrompre la recherche dès que la certitude de ne plus pouvoir trouver d'autres solutions intéressantes peut être acquise. Ce contrôle des recherches permet d'obtenir un algorithme dont la complexité est difficilement estimable, mais en pratique beaucoup plus rapide qu'un algorithme de test systématique.

Comme il a été dit précédemment, il est nécessaire de travailler sur des équations principales libres entre elles. Ce choix peut être quelconque, cependant il semble raisonnable de considérer les équations de décalages relatifs entre lettres consécutives. Les autres équations, les équations secondaires, sont conservées dans le calcul de la validité des solutions, permettant à ce choix arbitraire d'être sans conséquence.

E2 - Description de l'implantation

a - Choix d'implantation

La sélection des équations est réalisée suivant un algorithme procédant de manière récursive. Si le choix doit être réalisé sur n équations, l'algorithme en sélectionne d'abord $n-1$, incrémente les pointeurs correspondant sur les solutions ordonnées des équations, et recommence l'opération pour ces $n-1$ équations. Après avoir parcouru les possibilités de sélection de $n-1$ équations, l'algorithme passe à $n-2$, puis $n-3$ ... Lorsque l'ensemble des sélections est parcouru, les n pointeurs sont incrémentés et l'algorithme recommence les sélections depuis ce nouvel étage.

L'algorithme tel qu'il est décrit parcours de manière unique l'ensemble des solutions possibles. L'optimisation réalisée est de calculer au fur et à mesure la validité des équations. Un tel calcul étant trop coûteux, c'est une majoration du coefficient de justesse qui est utilisée en ne considérant que les équations principales, et les coefficients maxima des équations secondaires. Si une solution n'a pas son majorant suffisamment grand pour entrer dans la table des résultats, sa validité effective ne pourra a fortiori être assez grande pour permettre la prise en compte.

b - Problèmes rencontrés

La résolution du système d'équations est une partie critique en temps de calcul du décryptage du chiffrement de Vigenère. Par ailleurs, l'algorithme de parcours des equations par ordre de probabilité est hautement récursif, ce qui prohibe l'utilisation de fonctions d'allocation de la mémoire. Deux principes ont été utilisés pour répondre à ces problèmes d'implantation. En premier lieu, aucune allocation de mémoire n'est réalisée dans les parties critiques de l'algorithme. Les zones mémoires sont prises dans des variables locales passées en paramètres aux appels suivants. Le second principe est l'encapsulage de l'ensemble des données pour transmettre les intermédiaires des calculs.

Si le temps d'exécution est très réduit en travaillant sur les équations, il ne demeure pas moins que la complexité du calcul reste inacceptable pour de grands nombres d'équations. De plus, la qualité des solutions dépend du nombre d'équations, dans la mesure où cela permet de corriger davantage d'erreurs. Cependant, la pratique montre que le nombre d'équations peut devenir un facteur limitatif de la qualité des résultats. Le choix réalisé est donc de pouvoir limiter le nombre d'équations d'un système. Un système trop grand est découpé en deux systèmes, et après d'éventuels appels récursifs une jointure est effectuée sur les deux solutions pour déterminer la solution du système. En pratique, les temps de recherche restent raisonnables pour un systèmes à 6 équations principales, cependant au-delà de 5 équations il devient difficile pour l'algorithme d'écarter les équations erronées. En effet, ces équations peuvent sembler compatibles entre elles et introduire des résulats faux. A la suite de tests, le nombre optimal a été estimé à 4.

Si le nombre est fixé à 4, pour une clé de longueur 20 produisant 19 équations principales de décalages relatifs, l'algorithme découpe la recherche en la recherche de deux clés de 10 lettres, la clé en excés étant nécessaire pour permettre la jointure suivant une valeur commune. Le découpage continue récursivement de la manière suivante :

```
19
9+1=10            10
5+1=6    5        5+1=6    5
3+1=4  3  2+1=3  3  3+1=4  3  2+1=3  3
```

Pour terminer, la différence en temps de calcul est raisonnable entre 3 équations et 4 équations. Dans la mesure où les calculs sont plus fiables avec 4 équations, si le nombre d'équations est inférieur à 4, un débordement est réalisé sur la portion de la clé étudiée pour augmenter le nombre d'équations. Les équations ajoutées sont par la suite ignorées au moment de la jointure. Pour l'exemple décrit, il y a donc 8 minimisations pour 8 portions de la clé, et 7 jointures.

F - Indice de coïncidence mutuel

F1 - Présentation de l'algorithme

L'indice de coïncidence mutuel est, comme il a été dit précédemment, l'expression mathématiques du fait que l'aspect de la courbe des fréquences des lettres se conserve lors du décalage de l'alphabet. Il suffit par conséquent de calculer la distance de deux répartitions pour savoir si ces répartitions sont identiques. L'indice de coïncidence mutuel permet d'utiliser un principe similaire d'une manière plus fine, en calculant la probabilité pour que les alphabets de deux textes correspondent. En décalant circulairement l'une des répartitions de fréquences, il est alors possible de savoir quel est le décalage effectif entre les deux répartitions.

Si cet indice ne concerne initialement que des lettres seules, il a semblé intéressant de généraliser l'implantation de la méthode à des suites de lettres plus importantes. Il existe cependant une limite à la taille de ces suites, puisque les résultats sont enregistrés dans un tableau. Pour une suite de n lettres, le tableau a une taille de 26^n, la taille de la suite ne peut par conséquent pas dépasser certaines limites raisonnables. Par ailleurs, au-delà de 4 ou 5, il semble que cet indice ne puisse avoir un bien grand sens.

F2 - Description de l'implantation

a - Choix d'implantation

Le calcul de l'indice de coïncidence à partir de deux répartitions ne pose aucune difficulté. En revanche, le problème consiste à pouvoir obtenir la répartition d'un texte suivant un certain décalage sans pour autant devoir tout recalculer. Malgré les apparences, la solution de ce problème n'est pas si simple.

Dans le cas du calcul d'un indice de coïncidence sur des lettres seules, il suffit de déplacer le pointeur dans la table des fréquences en fonction du décalage souhaité. En revanche, la méthode ne peut pas fonctionner aussi brutalement pour des suites de lettres quelconques. Il est par conséquent nécessaire de jouer astucieusement avec les indices. Ce calcul est réalisé par une fonction décodant la position d'une suite de lettre dans la table pour chacune des lettres, et en calculant le décalage introduit pour obtenir la position décalée.

b - Problèmes rencontrés

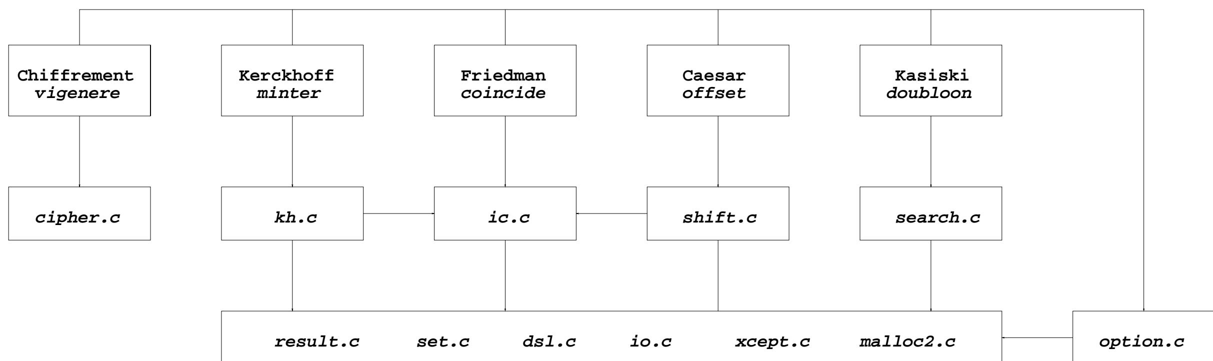
L'indice de coïncidence consiste à multiplier les probabilités d'apparition des lettres. Pour des textes courts, des lettres peuvent ne pas être présentes et l'indice est alors parfois nul, ne permettant plus la moindre conclusion. Ce problème est pourtant inhérent à l'utilisation d'une méthode dans des conditions non favorables.

La solution employée est de combiner le calcul de l'indice de coïncidence avec la distance quadratique des deux répartitions. Le second calcul est multiplié par un coefficient très faible, et n'intervient que dans les cas critiques pour distinguer des valeurs qui étaient initialement nulles. Finalement, pour s'assurer que les coefficients ne sont pas nuls, une constante extrêmement petite est ajoutée à l'indice.

Chapitre 4

Programmes sources

Les programmes sources ont été conçus pour permettre un maximum de portabilité. Les logiciels ont pu être compilés et testés sur les stations `SUN`, sur un powerPC sous système `BeOS`, et sur un pentium pro sous `Linux`. La portabilité a également été élargie au niveau des fichiers binaires créés et utilisés par certains des logiciels.



A - Modules d'outils

A1 - Gestion des exceptions (xcept)

Le module `xcept` a été construit à partir du module d'exceptions traité en travaux dirigés en maîtrise 1996-1997. Deux problèmes se posaient dans le module initial, problèmes résolus avec la nouvelle version. D'une part, lors d'erreurs systèmes telles que celles qui peuvent intervenir pour l'ouverture d'un fichier, il est possible de lever une exception. Cependant, si cette exception n'est pas interceptée, seules les informations concernant la position de la levée de l'exception sont fournies à l'utilisateur. Dans le cas d'une lecture sur un fichier, il n'y a par conséquent aucun moyen de connaître la cause de l'erreur système. D'autre part, si une fonction décide d'intercepter toutes les exceptions, par exemple pour effectuer des desallocations, en cas d'exception avant de transmettre elle-même l'exception, l'origine de l'exception est perdue. En cas de message d'erreur pour l'utilisateur, l'exception désignera la dernière fonction à avoir transmis l'exception et non la fonction l'ayant levée initialement.

Le premier problème a été résolu en associant à une levée d'exception un message permettant de justifier l'exception. Si aucune justification n'est apportée, le module consulte les erreurs systèmes et utilise comme justification les messages d'erreurs associés. Dans le cas où une erreur système remonte jusque l'utilisateur, les informations disponibles sont à la fois les raisons de l'erreur, par exemple `fichier:error` pour une erreur `I/O`, et la position où l'erreur a été décelée, `module.c:ligne`. De manière à apporter une solution au second problème, une nouvelle exception, `last_XCEPT`, a été créée pour permettre de faire suivre une exception sans en créer une nouvelle.

La conception du module d'exception a mis le point sur de gros problèmes de portabilité. Le premier problème a concerné la définition des fonctions comme des `lvalue`. Si le choix se justifie pour permettre une homogénéisation avec le mode d'utilisation des fonctions classiques, ce dernier implique des hurlements de compilateurs autres que `gcc`. En effet, si les deux arguments d'un appel `HANDLE` ne sont pas d'un même type, typiquement un appel de fonction retournant un `void` et une affectation retournant le résultat de l'affectation, l'utilisation de la forme `test?vrai:faux` n'est plus homogène. La solution a donc été de spécifier des *cast* en `void`. Le second problème est historique. Les anciennes libairies standards sous Unix ne contenaient pas la fonction `strerror` qui a été introduite plus tard. Pour pallier à l'absence de cette fonction sur encore de nombreuses machines, un patche a été ajouté au module d'exceptions. [1]

```
#if !(HAVE_STRERROR || __dest_os == __be_os)
extern int sys_nerr;
extern char *sys_errlist[];
#define strerror(E) ((E)<sys_nerr?sys_errlist[(E)]:"bogus error number")
#endif
```

A2 - Gestion de la ligne de commande (`option`)

Un module est dédié à l'interaction des logiciels avec la ligne de commande. Le texte d'usage est un tableau de caractères contenant deux suites de chaînes terminées par un pointeur `NULL`. Les chaînes avant le premier `NULL` sont tabulées par rapport au nom du logiciel, les chaînes comprises entre les deux pointeurs `NULL` étant copiée sans modification. La gestion des options est réalisée par un tableau de chaînes décrivant les options et terminé par un pointeur `NULL`. Un second tableau est associé, indiquant les codes respectifs des options.

Le module fournit des fonctions de lecture et d'écriture d'un fichier en utilisant le nom du fichier ainsi que l'adresse d'un pointeur de chaîne pointant, ou destiné à pointer, sur le contenu du fichier. Si les noms des fichiers sont des pointeurs `NULL`, l'entrée ou la sortie standard sont utilisées. La fonction de lecture effectue un filtrage des données lues, un caractère nul pouvant être très peu apprécié au milieu d'une chaîne ...

1. Nous tenons à remercier Philippe Narbel pour avoir réalisé les tests de compilation sur les anciennes librairies C, ainsi que Gérald Point pour nous avoir procuré les patches de `strerror` ...

A3 - Structures de tri (result/set)

a - Structure des résutats (result)

Le module `result` est construit pour permettre de trier des solutions sans effectuer la moindre allocation et libération de mémoire. A la construction d'une structure `result` par la fonction `result_create()`, un nombre de solution `depth` est spécifié, et une seule zone mémoire est allouée pour l'ensemble des données. Une solution est constitué d'un tableau de `short`, et d'un coefficient `double`. L'insertion d'une solution fonctionne de la manière suivante : une demande est effectuée pour obtenir l'adresse du tableau tampon de type `short`, et une autre pour celle du coefficient. Les deux champs sont remplis aux valeurs à insérer, puis la fonction `result_insert()` est exécutée. Si la solution a un coefficient suffisant pour entrer dans la sélection, alors elle est insérée et par conséquent il y a une solution qui est ôtée de la sélection. Les champs de cette solution écartée sont remis à zéro et servent de nouveaux tampons pour une insertion ultérieure.

Un certain nombre de possibilités ont été programmées. La fonction `result_normalize()` permet de positionner la meilleure solution à un coefficient de 1, `result_map()` applique une fonction à chacune des solutions, `result_clean()` élimine les champs non utilisés par des solutions, `result_sort()` permet de trier la structure si par exemple l'usage de `result_map()` avec une fonction particulière a modifié les coefficients ... Une fonction plus étrange est `result_join()`. Cette dernière fonction permet de réaliser la jointure de deux structures `result` suivant une valeur commune des données dans une colonne fixée. Les arguments utilisés déterminent d'une part les colonnes qui seront conservées pour la jointure, d'autre part les deux colonnes servant à réaliser la jointure. Si la jointure suivant les données communes ne suffit pas à remplir la nouvelle structure, les autres jointures sont insérées à la suite. Les insertions sont effectuées en utilisant comme coefficient le produit des deux coefficients, ainsi éventuellement qu'un coefficient correspondant au plus petit coefficient des jointures correctes[2].

```
result_p
result_join(result_p result1, short width1, short start1, short join1,
            result_p result2, short width2, short start2, short join2)
{
  long i, j;
  result_p result;
  double minimum = 1;   /* pour placer les pseudo-jointures apres les vraies */
  short stopit;         /* pour interrompre la recherche des que possible    */

  VERIFY(result1);
  VERIFY(result2);

  result = result_create((result1->depth<result2->depth)?
           result1->depth:result2->depth, width1+width2);

  for(i=0; i<result1->depth; i++)
    for(stopit=0, j=0; j<result2->depth && stopit!=-1; j++)
```

2. La fonction `_store()` insère dans la structure une solution dont le contenu est la concaténation de deux tableaux de type `short`

```
        if(result1->offset[i][join1]==result2->offset[j][join2])
          {
            double quality = result1->quality[i]*result2->quality[j];
            if(minimum>quality)
              minimum = quality;
            stopit = _store(result, result1->offset[i]+start1, width1,
                            result2->offset[j]+start2, width2, quality);
          }

    /* regarder si la derniere solution est utilisee */

    if(result->quality[result->depth-1]!=0)
      return result;

    /* il faut completer la structure avec ce qui reste */

    for(i=0; i<result1->depth; i++)
      for(stopit=0, j=0; j<result2->depth && stopit!=-1; j++)
        if(result1->offset[i][join1]!=result2->offset[j][join2])
          stopit = _store(result, result1->offset[i]+start1, width1,
                          result2->offset[j]+start2, width2,
                          (result1->quality[i]*result2->quality[j])*minimum);

    return result;
  }
```

b - Structure multi-ensemble (set)

Le module `set` permet des manipulations plus complexes qu'une simple gestion de multi-ensembles. Outre la possibilité d'utiliser la structure de tri sans se soucier ni des types, ni des allocations des objets insérés, `set` permet de choisir arbitrairement les coefficients. Par défaut, le coefficient d'un élément est son nombre d'insertions dans la structure. Il est cependant possible de forcer l'incrément utilisé pour les insertions lors d'une insertion, mais également de normaliser la structure pour que la somme des coefficients soit égale à 1. Les données dans la structure peuvent être accédées de deux manières: soit en recherchant un élément donné dans la structure, soit en parcourant linéairement la structure comme pour une liste chaînée en utilisant la fonction `set_forward()`. Le premier élément de la structure est obtenu par `set_forward(NULL)`, et un pointeur `NULL` est retourné lorsque le parcours est achevé.

La double méthode d'adressage peut sembler un peu étrange, mais elle permet d'améliorer la complexité des algorithmes construits dessus. En effet, une insertion est effectuée en $log_2(n)$ en tant qu'accés dichotomique, et un parcours s'effectue en n comme pour une liste chaînée. Pour permettre d'effectuer ces deux méthodes d'accés, le module a été construit sur une version légèrement modifiée du module `skiplist` réalisé en licence 1995-1996. Le module de `skiplist` modifié a été nommé `dsl`[3].

3. Pour `SkipList` à Double accés.

La structure gère une en-tête pour pouvoir connaître les valeurs des incréments, le poids total de la struture, le type des données ... L'extrait ci-dessous est la fonction permettant de ramener le poids de la structure à 1 en travaillant sur l'en-tête.

```
typedef struct set *set;

struct set
{
  int type;
  long entry;     /* entrees dans la structure */
  double size;    /* poids total de la structure */
  double step;    /* valeur de l'increment */
  dsl dsl;
};

void
set_normalize(set set)
{
  void *key = NULL;   /* le premier element est suivant de NULL */

  if(set->size==1)
    return;

  while(key=dsl_next(set->dsl, key))
    (*(double *) dsl_find(set->dsl, key)) /= set->size;

  set->step /= set->size;   /* modification de l'increment */
  set->size = 1;            /* modification du poids total */
  return;
}
```

B - Modules des outils de décryptage

B1 - Méthode de Kasiski (`search`)

La recherche de la longueur de la clé est réalisée en trois étapes. La première étape consiste à enregistrer les distances entre des occurrences multiples de suites de lettres. De manière à limiter le coût en mémoire et en temps de calcul, le texte est découpé en blocs sur lesquels sont recherchés les répétitions. La seconde étape est le calcul des *pgcd* pour permettre à la longueur de la clé d'apparaître. La dernière étape consiste à trier les longueurs possibles pour la clé par ordre de plus grande probabilité, et à n'en conserver qu'un nombre donné.

Le calcul des *pgcd* suivant l'algorithme détaillé en page 13 est construit sur la possibilité de parcourir séquentiellement par ordre décroissant la structure contenant les distances entre les occurrences multiples.

```
long _pgcd(long value1, long value2);   /* pgcd de deux entiers */

void
_allpgcd(set set)
{
  void *key1 = NULL;
  void *key2 = NULL;

  VERIFY(set);

  while(key2 = (key1 = set_forward(set, key1)))
    while(key2 = set_forward(set, key2))
      {
        double value = _pgcd(*(double *) key1, *(double *) key2);
        set_select(set, (void *) &value, set_member(set, key1));
      }
  return;
}
```

B2 - Test de Friedman (ic)

L'algorithme du test de Friedman décrit en page 15 est une application des indices de coïncidence et indices de coïncidence mutuels détaillés en page 20. Les calculs étant basés sur une analyse fréquentielle du texte, le module fournit des fonctions permettant de réaliser ces analyses : `ic_frequency()` qui retourne une table contenant les fréquences des lettres, et `ic_compute()` qui réalise la même opération mais en enregistrant les fréquences dans un tableau dont l'adresse lui est fournie.

Le calcul de l'indice de coïncidence intervient dans des algorithmes dont les complexités sont élevées, voire exponentielles. Il a donc été indispensable de pouvoir éviter des pertes de temps inutiles. L'utilité de permettre à la fonction appelante de gérer elle-même les tables de fréquences sans les calculer systématiquement, comme cela aurait été fait si uniquement `ic_coincide` avait été déclarée `extern`, est une illustration de ce problème. La seconde optimisation consiste à pouvoir déduire d'une table des fréquences, la table des fréquences d'un alphabet décalé. Or, le problème est épineux car si `AA`<`AB` et `BB`<`BC`, en revanche `AA`<`AZ` et `BB`>`BA`, ce qui revient à dire que l'ordre lexicographique n'est pas conservé par un décalage de l'alphabet. La convertion est par conséquent réalisée par la fonction `_index_shift()`.

```
int
_index_shift(int index, int shift)
{
  int newindex=0;
  int factor;       /* pour comptabiliser les 'dizaines' */

  for(factor=1; index || factor==1; index/=CHAR_RANGE, factor*=CHAR_RANGE)
    newindex += ((index+shift)%CHAR_RANGE)*factor;

  return newindex;
}
```

Comme il est détaillé en page 20, la version de l'indice de coïncidence est légèrement adaptée. En voici le source exacte[4] :

```
#define PSEUDO_VALUE .000001   /* coefficient tres faible */

double
ic_mutual(double *table1, double *table2, short width, int shift)
{
  int i;
  int depth=1;
  double result1=0;   /* indice de coincidence mutuel normal */
  double result2=0;   /* distance quadratique des tables     */

  VERIFY(table1);
  VERIFY(table2);

  for(i=0; i<width; i++)
    depth *= CHAR_RANGE;
  for(i=0; i<depth; i++)
    {
      int index = _index_shift(i, shift);
      result1 += table1[i]*table2[index];
      result2 += _sqr(table1[i]-table2[index]);
    }

  return result1*(1-2*PSEUDO_VALUE)+(1-result2)*PSEUDO_VALUE+PSEUDO_VALUE;
}
```

4. La fonction `_sqr()` calcule le carré de deux entiers.

B3 - Méthode de Kerckhoff (kh)

La fonction `kh_compute()` commence par demander le calcul des décalages relatifs entre les lettres de la clé. La clé demandée est d'une lettre trop longue, permettant de vérifier que la somme des décalages trouvés fasse bien 0 modulo 26. Dans le cas contraire, la solution est déclassée. Le filtrage est réalisé par la fonction `_filter_solutions()`.

La recherche est effectuée par `_analyse()`. En fonction de la longueur de la clé demandée, `_analyse()` a le choix soit de réaliser le calcul, soit de demander deux appels récursifs en divisant la clé en deux morceaux et de calculer la jointure des résultats. La partie de calcul direct consiste en réalité à construire le système d'equations, et a le faire analyser par `_find_solutions()`. Cette dernière fonction parcours les équations suivant un ordre bien particulier, tout en surveillant que le calcul est nécessaire. Dans le cas où le calcul ne peut aboutir a une solution convaincante, la fonction interrompt la recherche suivant cette direction et retourne à l'appel récursif précédant. Dans le cas contraire la fonction effectue des appels récursifs sur des parties du système.

a - Parcours des équations

Le principe de l'algorithme utilisé par `_find_solutions` est construit sur la remarque suivante : si dans le système d'équations la meilleure solution n'est pas celle correspondant aux meilleures solutions des équations principales, cela signifie qu'il existe une équations dont les coefficients sont erronés. Le parcours des équations commence donc par supposer qu'un ensemble de coefficients d'une équation est faux, et chaque équation est testée pour chaque coefficient en bloquant les autres équations. L'étape suivante suppose deux ensembles erronés, et ainsi de suite. Lorsque l'algorithme arrive à douter de l'ensemble des équations, les positions sont incrémentées et la méthode de recherche recommence sur cette nouvelle origine. L'exemple suivant illustre le parcours pour 4 équations ayant 3 coefficients, classées par ordre de probabilité de 0 à 2.

```
(0 0 0 0)
    (1 0 0 0)
        (2 0 0 0)
    (0 1 0 0)
        (0 2 0 0)
    (0 0 1 0)
        (0 0 2 0)
    (0 0 0 1)
        (0 0 0 2)
    (1 1 0 0)
        (2 1 0 0)
        (1 2 0 0)
        (2 2 0 0)
    (1 0 1 0)
        (2 0 1 0)
        (1 0 2 0)
        (2 0 2 0)
    (1 0 0 1)
        (2 0 0 1)
        (1 0 0 2)
        (2 0 0 2)
    (0 1 1 0)
```

```
        ...
    (0 1 0 1)
        ...
    (0 0 1 1)
        (0 0 2 1)
        (0 0 1 2)
        (0 0 2 2)
    (1 1 1 0)
        (2 1 1 0)
        (1 2 1 0)
        (1 1 2 0)
        (2 2 1 0)
        (2 1 2 0)
        (1 2 2 0)
        (2 2 2 0)
    (1 1 0 1)
        ...
    ...
(1 1 1 1)
    ...
(2 2 2 2)
```

Si le parcours peut sembler quelque peu étrange, sa programmation est réalisable récursivement. Le principe de base consiste à placer dans un tableau des pointeurs vers les versions des équations actuellement choisies. La permière position correspond au décalage relatif le plus probable entre les deux lettres de la clé, la dernière au décalage le moins probable. La fonction `_find_solutions()` reçoit n équations dont il faut choisir les versions. Elle commence par en bloquer $n-1$ et lance un appel récursif sur la dernière équation en incrémentant la position dans les probabilités. La fonction bloque ensuite $n-2$ équations, etc ... Le parcours des possibilités de sélections de p équations parmi n est réalisé par la fonction `_next_choice()`.

Lorsque l'ensemble des équations principales est déterminé, les décalages relatifs entre les lettres de l'extrait de la clé sont répercutés sur les équations secondaires. La solution testée est ainsi associée à un coefficient caractérisant sa probabilité. Lorsqu'avant même de calculer les répercutions des décalages, le produit des coefficients des équations principales est trop faible pour que la solution puisse être intéressante, il est alors possible d'interrompre une partie du parcours des équations, tout nouvel appel récursif engendrant lui-même un coefficient sur les équations principales encore plus faible.

b - Filtrage des solutions

La clé recherchée déborde d'une lettre, ce qui signifie que l'algorithme fournit les distances relatives d_{01}, d_{12} ... $d_{(n-1)(n-2)}$, mais également la distance $d_{(n-1)0}$. La dernière distance est en théorie redondante avec les autres solutions, mais permet de vérifier que les distances trouvées bouclent, c'est-à-dire que leur somme est nulle. `_filter_solutions()` effectue le filtrage en utilisant deux fois la fonction `result_map()`. Le premier passage permet de calculer le plus faible coefficient des solutions bouclant correctement. Le second passage multiplie les solutions ne bouclant pas avec ce minimum. Un tri final sur la structure permet de cette manière de déclasser les solutions douteuses.

```
double
_mapminimum(short *offset, double quality, short width, void *argument)
{
  short i;
  long value = 0;

  for(i=0; i<width; i++)
    value += offset[i];
  value %= CHAR_RANGE;

  if(value==0 && *((double *) argument)>quality)
    *((double *) argument) = quality;

  return quality;
}


double
_mapreduce(short *offset, double quality, short width, void *argument)
{
  short i;
  long value = 0;

  for(i=0; i<width; i++)
    value += offset[i];
  value %= CHAR_RANGE;

  if(value==0)
    return quality;
  else
    return quality*(*((double *) argument));
}


void
_filter_solutions(result_p result)
{
  double minimum = 1;

  result_map(result, _mapminimum, (void *) &minimum);
  if(minimum==0)
    minimum=1;
  result_map(result, _mapreduce, (void *) &minimum);
  result_sort(result);    /* remet un peu d'ordre dans la structure    */
  result_clean(result);   /* elimine les solutions a coefficients nuls */

  return;
}
```

B4 - Cryptanalyse de chiffres monoalphabétiques (`shift`)

La cryptanalyse d'un texte chiffré en utilisant un chiffre monoalphabétique est réalisé en utilisant les indices de coïncidence mutuels avec les tables de fréquences de langues connues. Le calcul en lui-même ne pose pas de problème, le coefficient retenu pour pouvoir juger du meilleur décalage ayant été choisi comme étant le produit du coefficient de coïncidence mutuel pour les monogramme et du carré de l'indice pour les bigrammes.

Les informations sur les langues sont des enregistrements binaires dont le chemin d'accés du repertoire les contenant est transmis à la fonction devant les charger en mémoire. La fonction `shift_install()` consulte le contenu du répertoire pour y reconnaître les fichiers de langues. Les enregistrements binaires permettent de limiter la taille des fichiers, mais pose un problème de portabilité des ressources. En effet, le codage d'un `double` pouvant varier entre des processeurs, il n'est pas question de réaliser une sauvegarde brutale au format interne des `double`. Les fonctions `shift_double2binary()` et `_binary2double()` respectivement `extern` et `static` permettent de convertir un tableau de format `double` en un enregistrement binaire portable, et l'opération inverse respectivement. Le format de codage utilisé est celui des flottants en simple précision suivant la norme IEEE 754[**?**].

```
void
_binary2double(unsigned char *binary_table, double *double_table)
{
  short i;

  for(i=0; i<CHAR_RANGE*CHAR_RANGE; i++)
    {
      double value = 0;
      unsigned char double_s = binary_table[LANGUAGE_DATA_BYTES*i]>>7;
      unsigned char double_e = (binary_table[LANGUAGE_DATA_BYTES*i]<<1)
                             | (binary_table[LANGUAGE_DATA_BYTES*i+1]>>7);
      long double_m = (((long) binary_table[LANGUAGE_DATA_BYTES*i+1])&0x7f)<<16
                    | ((long) binary_table[LANGUAGE_DATA_BYTES*i+2])<<8
                    | ((long) binary_table[LANGUAGE_DATA_BYTES*i+3]);
      value = pow(-1, (double) double_s)
       *(1+((double) double_m)*pow(2, -23))
       *pow(2, (double) (((long) double_e)-127));

      if(double_e==0x00)
        value = 0;
      double_table[i] = value;
    }
  return;
}
```

C - Module de décryptage automatique

C1 - Principe de l'algorithme

Les motivations nous ayant poussés à la réalisation du module de décryptage automatique, avec l'accord de monsieur B. Courcelle, sont de deux ordres. La première raison est de pouvoir effectuer des batteries importantes de tests pour vérifier le bon fonctionnement des logiciels réalisés. La seconde raison est d'apporter une réponse au problème initial, à savoir estimer précisément les possibilités offertes par l'utilisation conjointe de l'ensemble des méthodes de décryptage.

Pour pouvoir répondre au second point, la méthode de décryptage est construite sur une utilisation simple des outils disponibles, suivant une méthode utilisable par un humain. Le logiciel commence donc par rechercher la longueur de la clé dans le texte chiffré. Si les longueurs calculées ne remplissent pas un certain nombre de conditions, quelques variantes simples sont disponibles. Le logiciel dispose alors de longueurs plus ou moins probables, qu'il peut essayer pour trouver les décalages entre les lettres de la clé. En calculant un coefficient tenant compte, à la fois de la probabilité de la longueur de la clé, et du coefficient qualifiant la probabilité pour que les décalages relatifs soient bien trouvés, le logiciel établit une table des clés possibles à un décalage près. Les clés de cette table sont ensuite utilisées pour déchiffrer le texte, et le comparer avec les langues connues. La comparaison permet de connaître le décalage de la clé, et le logiciel classe finalement les clés pour donner les choix qui lui semblent les meilleurs.

C2 - Description de l'implantation

La première étape consiste à utiliser la méthode de Kasiski pour rechercher les répétitions. Cette partie est réalisée par `_doubloon()`. Si les longueurs de clé proposées sont trop peu nombreuses, il est probable que la longueur de la clé n'ait pas eu la possibilité d'apparaître. Dans ce cas le logiciel teste les diviseurs en utilisant l'indice de coïncidence pour éventuellement insérer de nouvelles possibilité de longueur de clé. Si malgré tout les longueurs de clé ne sont pas en nombre suffisant, le logiciel essaye systématiquement toutes les longueurs de clé.

L'ensemble des longueurs possibles est ensuite envoyé à `_coincide` qui modifie les coefficients en fonction de l'indice de coïncidence. Cette modification est réalisée en deux passes de la fonction `result_map()` :

```
struct maparg
{
  char * data;
  long   length;
};

double
mapfun2(short *offset, double quality, short width, void *argument)
```

```
{
  struct maparg *arg = (struct maparg *) argument;
  double value = ic_coincide(arg->data, arg->length, offset[0]);
  return quality*value*value;   /* nouveau coefficient */
}

void
_coincide(char *data, long length, result_p passlen)
{
  struct maparg argument;
  argument.data = data;
  argument.length = length;

  /* l'indice de coincidence a besoin du texte et de la longueur */

  result_map(passlen, mapfun2, (void *) &argument);

  /* il faut retrier la structure qui a ete bouversee */

  result_sort(passlen);
  result_clean(passlen);

  /* verification que tout s'est bien deroule */

  if(result_depth(passlen)==0)
    RAISE(fatal_XCEPT, NULL, NULL);

  return;
}
```

Le logiciel utilise alors `_minter()` pour calculer les décalages relatifs entre les lettres de la clé. Les décalages seront réutilisés par la suite, donc les tables de décalages possibles pour chaque longueur de clé sont conservées. La fonction enregistre les résultats qui lui semblent les meilleurs dans une structure `result` dont un des champs détermine la longueur de la clé retenue, et un second les décalages relatifs choisis pour cette longueur.

Pour finir, `_offset()` effectue une opération similaire mais pour déterminer le décalage de la clé, et le logiciel récupère les diverses informations pour déterminer la valeur de la clé et la langue reconnue. Ces informations sont envoyées à l'interface, qui peut être soit l'interface graphique, soit la ligne de commande.

Chapitre 5

Exemples d'utilisation

A - Utilisation depuis un shell Unix

A1 - Installation des logiciels

Les sources des logiciels peuvent être obtenus sur simple demande en adressant un courrier électronique à dbv@komicom.remcomp.fr. L'installation des logiciels est réalisée suivant la procédure :

```
gunzip vigenere.tgz
tar xvf vigenere.tar
cd vigenere/sources/
./configure
make
cd ..
```

A2 - Transformation de textes

Les logiciels fonctionnent pour la plupart uniquement avec les lettres de l'alphabet majuscule. Il convient par conséquent de convertir les fichiers au bon format. Cette transformation s'effectue aisément avec l'aide des logiciels `tr` et `sed`. Une version minimale ne tenant pas compte des accents est typiquement :

```
cat fichier.txt | tr a-z A-Z | tr -cd A-Z > nouveau.txt
```

Un utilitaire a été réalisé pour pouvoir extraire une partie d'un texte. Son utilisation nécessite deux paramètres que sont la position de départ dans le texte, et la longueur de la sélection. Les options `-f` et `-o` permettent de choisir respectivement le fichier à lire, et le fichier recevant l'extrait du texte.

```
Usage: xtract [-v] [-h] [-k] [-f FILENAME] [-o FILENAME] BEGIN LENGTH

Extract a part of a file

  -f --infile   choose the file (default is STDIN)
  -o --output   save the result as a file (default is STDOUT)
  -k --clean    remove the accents from the text
  -h --help     display this help
  -v --version  output version information
```

A3 - Chiffrement de Vigenère

```
Usage: vigenere [-v] [-h] [-f FILENAME] [-o FILENAME]
                [-a MIN MAX] [-|+]PASSWORD

[De]cipher datas with PASSWORD

  -f --infile   choose the file to cipher (default is STDIN)
  -o --output   save the result as a file (default is STDOUT)
  -a --alpha    select a range of characters (default is A-Z)
  -h --help     display this help
  -v --version  output version information
```

Fonctionnant sur le même principe que `xtract` pour ce qui est de la sélection des fichiers, le seul argument de `vigenere` est la clé de chiffrement. Cette clé doit être dans l'alphabet utilisé, par défaut celui des majuscules. Si la clé ne contient que des lettres, ou si elle est précédée du signe `+`, `vigenere` opère le chiffrement. Si au contraire la clé est précédée du signe `-`, `vigenere` déchiffre le texte. L'exemple suivant décrit le chiffrement des premières lignes du *Colonnel Chabert* de Balzac en utilisant la clé `CHIFFRE`.

Voici le contenu du texte original et le texte filtré correspondant :

```
Vers la fin de l'année 1612, par une froide matinée de décembre, un
jeune homme dont le vêtement était de très mince apparence, se
promenait devant la porte d'une maison située rue des Grands-
Augustins, à Paris. Après avoir assez longtemps marché dans cette rue
avec l'irrésolution d'un amant qui n'ose se présenter chez sa
première maîtresse, quelque facile qu'elle soit, il finit par
franchir le seuil de cette porte, et demanda si maître François
PORBUS était en son logis. Sur la réponse affirmative que lui fit une
vieille femme occupée à balayer une salle basse, le jeune homme monta
lentement les degrés, et s'arrêta de marche en marche, comme quelque
courtisan de fraîche date, inquiet de l'accueil que le roi va
lui faire.
```

```
bash$ cat fichier.txt
VERS LA FIN DE L'ANNEE 1612, PAR  UNE FROIDE MATINEE  DE DECEMBRE, UN
JEUNE HOMME  DONT LE VETEMENT   ETAIT   DE  TRES  MINCE APPARENCE, SE
PROMENAIT DEVANT  LA  PORTE D'UNE  MAISON  SITUEE  RUE  DES  GRANDS-
AUGUSTINS, A PARIS. APRES AVOIR ASSEZ LONGTEMPS MARCHE DANS CETTE RUE
AVEC L'IRRESOLUTION D'UN  AMANT   QUI N'OSE  SE   PRESENTER  CHEZ SA
PREMIERE MAITRESSE, QUELQUE FACILE   QU'ELLE    SOIT,  IL FINIT PAR
FRANCHIR LE  SEUIL  DE  CETTE  PORTE, ET DEMANDA SI MAITRE  FRANCOIS
PORBUS ETAIT EN SON LOGIS. SUR LA REPONSE AFFIRMATIVE QUE LUI FIT UNE
VIEILLE FEMME OCCUPE A  BALAYER UNE SALLE BASSE, LE JEUNE HOMME MONTA
LENTEMENT LES DEGRES, ET S'ARRETA  DE MARCHE EN MARCHE, COMME QUELQUE
COURTISAN DE FRAICHE  DATE, INQUIET   DE   L'ACCUEIL   QUE LE ROI VA
LUI FAIRE.
```

Le texte peut être chiffré soit directement, soit en utilisant l'entrée standard. De même le résultat peut être enregistré dans un nouveau fichier.

```
bash$ vigenere -f fichier.txt +CHIFFRE
XLZX QR JKU LJ Q'RRPLM 1612, UFI  YPL NWTZHG TIYNEIG  KM IJTIOIZJ, ZE
NGBVJ MFQOL  LTSK PG CMYJDIPA   MYFZX   FL  BWJJ  QKUKJ FGTCYMSHV, WG
WZTRVRCPB IJMEPA  TF  UFVVL L'ZSV  QCPATS  JMVBMJ  WLI  FLA  LWRRFZ-
IZLLWVPVX, F GETPA. FUIIU HDTNI EUZME QFRIAMRUJ QCYKMJ UEPZ KJYKI TBM
FAVG N'PZWJJSNBBNTE H'WU  IRFEX   SBQ S'TJI  UL   XWJJIPAMW  HYIB ZI
UWVQKLZJ RRMVYMXXV, UWLTVZV JCJQQJ   HY'GSTJ    XFMV,  PT KNEMV WIW
KIEPJPNW CI  ULCNQ  UI  ELBYJ  GSTAM, JY UIOHVIF JM OHQYWV  JTHVHTZW
RVZGZJ IVHQY JE WQU TTLZW. UBZ QF IIRVVXJ RJHPZRFKMXL YZJ CYK MQY ZEI
XPMNQCI HLURJ FGEBXJ F  SENHGJW LRG ZIQQV FCZAJ, QV NGBVJ MFQOL UTSKE
NLVYJDIPA TJX UIIYMX, JK W'CYZJYR  HG TIWHYI GU UFWTLG, JWRRV UWLTVZV
GQBZYNJEP KM KWRMEOM  IFKI, KUYZNVX   FL   T'FHTYGPT   VZV PG YWN AR
PWP NFNII.
```

```
bash$ cat fichier.txt | vigenere -o nouveau.txt +CHIFFRE
bash$ cat nouveau.txt
XLZX QR JKU LJ Q'RRPLM 1612, UFI  YPL NWTZHG TIYNEIG  KM IJTIOIZJ, ZE
NGBVJ MFQOL  LTSK PG CMYJDIPA   MYFZX   FL  BWJJ  QKUKJ FGTCYMSHV, WG
WZTRVRCPB IJMEPA  TF  UFVVL L'ZSV  QCPATS  JMVBMJ  WLI  FLA  LWRRFZ-
IZLLWVPVX, F GETPA. FUIIU HDTNI EUZME QFRIAMRUJ QCYKMJ UEPZ KJYKI TBM
FAVG N'PZWJJSNBBNTE H'WU  IRFEX   SBQ S'TJI  UL   XWJJIPAMW  HYIB ZI
UWVQKLZJ RRMVYMXXV, UWLTVZV JCJQQJ   HY'GSTJ    XFMV,  PT KNEMV WIW
KIEPJPNW CI  ULCNQ  UI  ELBYJ  GSTAM, JY UIOHVIF JM OHQYWV  JTHVHTZW
RVZGZJ IVHQY JE WQU TTLZW. UBZ QF IIRVVXJ RJHPZRFKMXL YZJ CYK MQY ZEI
XPMNQCI HLURJ FGEBXJ F  SENHGJW LRG ZIQQV FCZAJ, QV NGBVJ MFQOL UTSKE
NLVYJDIPA TJX UIIYMX, JK W'CYZJYR  HG TIWHYI GU UFWTLG, JWRRV UWLTVZV
GQBZYNJEP KM KWRMEOM  IFKI, KUYZNVX   FL   T'FHTYGPT   VZV PG YWN AR
PWP NFNII.
```

```
bash$ vigenere -f nouveau.txt -CHIFFRE
VERS LA FIN DE L'ANNEE 1612, PAR  UNE FROIDE MATINEE  DE DECEMBRE, UN
JEUNE HOMME  DONT LE VETEMENT   ETAIT   DE  TRES  MINCE APPARENCE, SE
PROMENAIT DEVANT  LA  PORTE D'UNE  MAISON  SITUEE   RUE  DES  GRANDS-
AUGUSTINS, A PARIS. APRES AVOIR ASSEZ LONGTEMPS MARCHE DANS CETTE RUE
AVEC L'IRRESOLUTION D'UN  AMANT   QUI N'OSE  SE    PRESENTER  CHEZ SA
PREMIERE MAITRESSE, QUELQUE FACILE    QU'ELLE     SOIT,  IL FINIT PAR
FRANCHIR LE  SEUIL   DE  CETTE  PORTE, ET DEMANDA SI MAITRE  FRANCOIS
PORBUS ETAIT EN SON LOGIS. SUR LA REPONSE AFFIRMATIVE QUE LUI FIT UNE
VIEILLE FEMME OCCUPE A  BALAYER UNE SALLE BASSE, LE JEUNE HOMME MONTA
LENTEMENT LES DEGRES, ET S'ARRETA  DE MARCHE EN MARCHE, COMME QUELQUE
COURTISAN DE FRAICHE  DATE, INQUIET   DE   L'ACCUEIL    QUE LE ROI VA
LUI FAIRE.
```

Les lettres n'appartenant pas à l'alphabet sont ignorées. Le résultat du chiffrement est donc indépendant de l'ordre avec lequel l'élimination des caractères qui n'appartiennent pas à l'alphabet est réalisée. Les logiciels présentés par la suite ne travaillent que sur l'alphabet, il est donc recommandé à partir de maintenant d'utiliser des textes filtrés ...

A4 - Analyse statistique des fréquences

```
Usage: frequency [-v] [-h] [[-f FILENAME] -l LANGUAGE]
                 [[-f FILENAME] [-d LENGTH START] [-s STEP]]

Analyse the content of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -s --step      choose the step of the analyse (default is 1)
  -d --distance  analyse every LENGTH character from START
  -l --language  create a language file LANGUAGE
  -h --help      display this help
  -v --version   output version information
```

Le logiciel `frequency` réalise une analyse fréquentielle de suites de lettres du texte. La longueur de la suite de lettres est définie par `-s STEP`. Il est possible de ne comptabiliser que 1 cas sur n. Ceci définit un découpage du texte par blocs de n lettres dont une seule est le début d'une suite de lettres comptabilisée. Il convient donc de pouvoir définir la position (`START`) de cette lettre dans le cycle (`LENGTH`).

```
bash$ frequency -f nouveau.txt -d 1 0
A 0.0206540447504302
B 0.0240963855421686
C 0.0206540447504302
D 0.00516351118760757
E 0.0327022375215146
F 0.0464716006884681
G 0.036144578313253
H 0.0292598967297762
```

```
I 0.0705679862306368
J 0.0791738382099827
K 0.0327022375215146
L 0.0481927710843373
M 0.0481927710843373
N 0.0327022375215146
O 0.0103270223752151
P 0.0395869191049913
Q 0.0395869191049913
R 0.0395869191049913
S 0.0189328743545611
T 0.0499139414802065
U 0.0430292598967297
V 0.0585197934595524
W 0.0499139414802065
X 0.027538726333907
Y 0.0447504302925989
Z 0.0516351118760757
bash$ frequency -f nouveau.txt -d 6 0
B 0.0120481927710843
C 0.0843373493975903
E 0.036144578313253
F 0.0481927710843373
G 0.156626506024096
H 0.0240963855421686
I 0.0240963855421686
K 0.0602409638554216
N 0.0481927710843373
O 0.0602409638554216
P 0.108433734939759
Q 0.0240963855421686
R 0.0240963855421686
S 0.0120481927710843
T 0.0481927710843373
U 0.0602409638554216
V 0.0843373493975903
W 0.0481927710843373
X 0.036144578313253
```

La seconde utilisation de `frequency` consiste à réaliser des tables de fréquences de langages, tables qui sont utilisées par la suite par les logiciesl `offset`, `decipher` et `autotest`. Ces tables ne sont qu'un enregistrement binaire d'une analyse pour des bigrammes telle que `frequency -s 2`. Une table contient $26 * 26$ flottants en simple précision, enregistrés sous le format `IEEE 754`.

A5 - Méthode de Kasiski

```
Usage: doubloon [-v] [-h] [-f FILENAME] [-s STEP] [-b BUFFER] [-d DEPTH]

Search some cycles in a file

  -f --infile    choose the file to analyse (default is STDIN)
  -s --step      choose the step of the search (default is 3)
  -b --buffer    choose the length of the search (default is 2048)
  -d --depth     choose the depth of the search (default is 10)
  -h --help      display this help
  -v --version   output version information
```

Les répétitions sont recherchées sur des morceaux de texte de BUFFER lettres. Plus cette taille est grande, plus il y a de chances de trouver des répétitions, mais au dépend du temps de calcul. L'option -s permet de choisir la taille des répétitions cherchées. Cette taille doit être la plus grande possible, tout en restant raisonnable pour pouvoir trouver des répétitions. les solutions proposées par le logiciel sont associées à un coefficient compris entre 0 pour mauvais, et 1 pour bon. Le logiciel ne propose au maximum que les DEPTH meilleures solutions, cependant la somme des coefficients pour l'ensemble des solutions est 1.

```
bash$ doubloon -f nouveau.txt -s 5
07  0.625
406  0.25
245  0.125
bash$ doubloon -f nouveau.txt -s 4
01  0.643599
07  0.200692
14  0.0570934
05  0.0432526
35  0.0190311
406  0.0173010
245  0.00865052
42  0.00346021
469  0.00173010
252  0.00173010
```

A6 - Indice de coïncidence

```
Usage: coincide [-v] [-h] [-f FILENAME] [-c LENGTH]

Coincidence value of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -c --cycle     choose the cycle of the analyse (default is 1)
  -h --help      display this help
  -v --version   output version information
```

Le logiciel coincide permet de calculer l'indice de coïncidence d'un texte. Le choix d'un cycle ordonne le calcul pour des bandes de textes formées de lettres distantes de LENGTH. Des tests successifs peuvent permettre d'intuiter la longueur de la clé.

```
bash$ coincide -f nouveau.txt
0.0072753801990888
bash$ coincide -f nouveau.txt -c 2
0.00942512596380064
bash$ coincide -f nouveau.txt -c 3
0.0104761542223917
bash$ coincide -f nouveau.txt -c 4
```

```
0.0129460190463564
bash$ coincide -f nouveau.txt -c 5
0.0141733092585252
bash$ coincide -f nouveau.txt -c 6
0.0156115770518785
bash$ coincide -f nouveau.txt -c 7
0.0520554347670039
bash$ coincide -f nouveau.txt -c 8
0.0215158679496891
bash$ coincide -f nouveau.txt -c 9
0.0193472504725509
```

A7 - Décalages relatifs entre les alphabets

```
Usage: minter [-v] [-h] [-f FILENAME] [-d DEPTH] CYCLE

Search some rules in a file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 10)
  -h --help      display this help
  -v --version   output version information
```

La longueur de la clé étant connue, `minter` permet de passer d'un chiffrement polyalphabétique à un chiffrement monoalphabétique. La suite des chiffres correspond aux décalages entre les lettres de la clé utilisée. La clé est donc ainsi connue, à un décalage de l'alphabet près. Ici, la clé indiquée est `AFGDDPC`, correspondant bien à un décalage de la clé `CHIFFRE`. Les coefficients compris entre 0 pour `mauvais` et 1 pour `bon` caractérisent les résultats.

```
bash$ minter -f nouveau.txt 7
05 01 23 00 12 13 24  1
05 01 23 00 12 17 20  0.260087
05 14 10 00 12 13 24  0.149333
18 01 23 00 12 13 24  0.0422718
09 01 23 00 12 13 24  0.0416481
05 01 23 00 12 13 02  0.0387774
22 01 23 00 12 13 24  0.0382997
05 01 23 00 12 13 11  0.0379191
05 14 23 00 12 13 24  0.0161746
05 01 23 22 12 13 24  0.014092
```

A8 - Décryptage du chiffrement de Cæsar

```
Usage: offset [-v] [-h] [-f FILENAME] [-d DEPTH] [-p PATH]

Search the offset of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 10)
  -p --path      choose the path for the language files (default is .)
  -h --help      display this help
  -v --version   output version information
```

Le logiciel `offset` recherche, à partir de fichiers construits en utilisant l'option `-l` de `frequency`, le décalage utilisé pour chiffrer le texte. Le chemin des langages doit être indiqué si ceux-ci ne sont pas dans le répertoire courant. En déchiffrant avec la clé `AFGDDPC` proposée par `minter`, le texte obtenu est devenu un texte chiffré par la clé `C`. L'exemple suivant suppose que les langues sont situées dans le répertoire `./langages`. La première colonne indique le décalage, la seconde la langue reconnue. Les coefficients, très faibles, permettent de comparer les solutions entre elles. La solution la meilleure est pour un décalage de deux lettres en se basant sur un texte écrit en anglais, ce qui correspond bien à la réalité.

```
bash$ cat nouveau.txt | vigenere -AFGDDPC | offset -p langages/
00 langages/anglais
01 langages/francais
02 langages/allemand

02 01  3.05161e-11
02 02  6.37216e-12
02 00  2.75622e-12
15 02  1.88775e-13
15 01  8.84552e-14
13 00  7.51460e-14
15 00  6.79718e-14
24 02  4.20908e-14
13 02  3.51863e-14
24 00  3.01762e-14
```

A9 - Décryptage automatique

```
Usage: decipher [-v] [-h] [-f FILENAME] [-d DEPTH] [-r RATIO] [-p PATH]

Search the password of a cipher file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 1)
  -r --ratio     choose the limit of the possible length of the key
```

```
    -p --path      choose the path for the language files (default is .)
    -h --help      display this help
    -v --version   output version information
```

Le logiciel `decipher` utilise directement les méthodes décrites dans les paragraphes précédents en procédant à ses propres choix. La seule paramétrisation possible est dans la liberté laissée au logiciel de pouvoir intuiter des longueurs de clé. Pour une valeur donnée de `RATIO`, la longueur de la clé sera au maximum la longueur du texte divisée par `RATIO`. En général, le logiciel utilise la méthode de Kasiski sur les doublons pour calculer la longueur de la clé et vérifie les résultats avec la méthode de Friedman utilisant les indices de coïncidence. Les diviseurs de la meilleure longueur sont par ailleurs controlés avec le test de Friedman pour être éventuellement ajoutés à la liste des longueurs possibles. Cependant, si `decipher` ne parvient pas à trouver suffisamment de doublons dans le texte pour en déduire la longueur de la clé, le test de Friedman est utilisé seul.

```
bash$ decipher -f nouveau.txt -p langages/
CHIFFRE (francais)
```

A10 - Logiciel de tests automatiques

```
Usage: autotest [-v] [-h] [-f FILENAME] [-r RATIO] [-p PATH]
                [-d DEPTH] [-n NUMBER] FIRST LAST

Repeat NUMBER tests of DEPTH parts for key's size from FIRST to LAST

  -f --infile    choose the reference file (default is STDIN)
  -d --depth     choose the depth of the search (default is 10)
  -n --number    choose the number of tests to compute (default is 1)
  -r --ratio     choose the limit of the possible length of the key
  -p --path      choose the path for the language files (default is .)
  -h --help      display this help
  -v --version   output version information
```

A partir d'un texte non chiffré, `autotest` effectue des tests pour des clés de longueurs comprises entre `FIRST` et `LAST`. Chaque test est effectué sur `DEPTH` extraits du texte, et le test est répété pour `NUMBER` clés de chiffrement. L'option `-r` agit de la même manière que pour le logiciel `decipher`. Le coefficient indiqué correspond à la probablité pour qu'une lettre de la clé trouvée par le logiciel soit correcte. Le logiciel réduit progressivement la taille des informations pour une clé fixée, et La boucle de test termine lorsque les lettres ont plus de chances d'être fausses que justes.

B - Utilisation de l'interface HTML

B1 - Installation de l'interface

L'expérimentation sous l'interface HTML nécessite l'utilisation d'un navigateur pour pages HTML permettant l'utilisation de formulaires. L'interface a pu être testée sous Netscape de la version 1.1 à la version 3.01 sans problème. Le logiciel peut être expérimenté soit en installant le fichier experiment.cgi en local, soit en consultant jusque septembre 1997 l'URL :

http://www.emi.u-bordeaux.fr/~morreeuw/universite/TER/experiment.html

Lors de la compilation des programmes sources, l'exécutable experiment.cgi est enregistré dans le répertoire html. L'installation de l'interface en local nécessite la copie du contenu du répertoire html de l'archive dans le répertoire devant contenir l'interface. Pour ajouter de nouveaux fichiers de références sur les langues, il suffit de les placer dans le même répertoire.

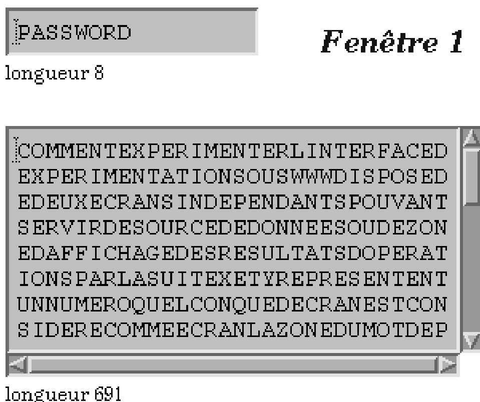
B2 - Présentation de l'interface

L'interface se présente sous la forme de deux fenêtres de chacune deux zones. Les zones correspondent à la clé de chiffrement et aux données à traiter. Les deux fenêtres ont un rôle strictement identique. Un menu de sélection permet de choisir l'opération à effectuer et les fenêtres concernées. Toute validation entraîne un formatage des informations dans les fenêtres avec la mise à jour des tailles affichées des données.

Interface d'expérimentation HTML

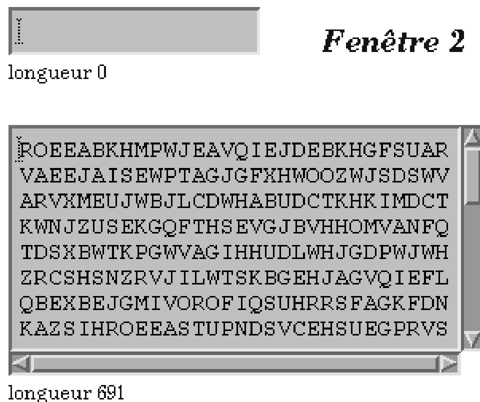
- `Formater` : filtre les lettres des fenêtres
- `Chiffrer` : chiffre le texte A avec la clé A dans la fenêtre B et efface la clé B
- `Déchiffer` : déchiffre le texte A avec la clé A dans la fenêtre B et efface la clé B
- `Décrypter` : recherche dans la fenêtre B la clé utilisée pour chiffrer le texte A

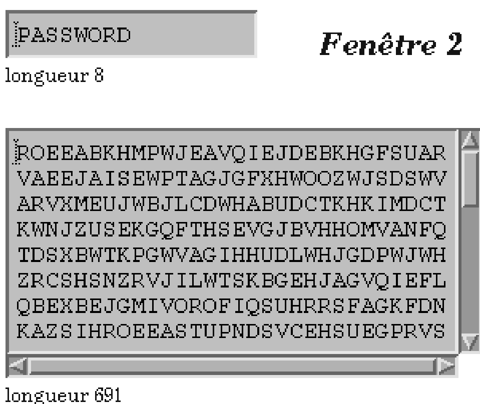
B3 - Exemple d'expérimentation



Pour commencer, le logiciel doit disposer de données sur lesquelles travailler. Il suffit pour cela de déposer avec un simple copier/coller le texte. Le texte déjà présent dans la fenêtre peut très bien convenir. L'étape suivante consiste à formater le contenu des fenêtres en cliquant sur `Exécuter`. Seuls les caractères alphabétiques sont conservés, et convertis en majuscules. Les accentuations des lettres minuscules sont également converties. Le résultat obtenu est affiché dans l'image ci-contre...

Il est maintenant possible de chiffrer le texte. Pour cela, il suffit de cliquer sur `Exécuter` après avoir sélectionné l'option `Chiffrer`. La clé `PASSWORD` sert alors pour chiffrer le contenu du texte en utilisant le chiffrement de Vigenère. La zone de la clé est vidée lors de cette opération. Le texte obtenu est affiché ci-contre...





Le décryptage peut à présent être expérimenté. Pour cela, il faut sélectionner fonction `Décrypter` de 2 vers 2 et `Exécuter`, la clé de chiffrement `PASSWORD` apparaît dans la fenêtre 2. L'opération peut être répétée en réduisant progressivement la taille du texte chiffré pour rechercher la taille limite. La limite pour cet exemple est de 205 caractères, ce qui représente environ 25 lettres par lettre de la clé.

C - Utilisation de l'interface Xwindow

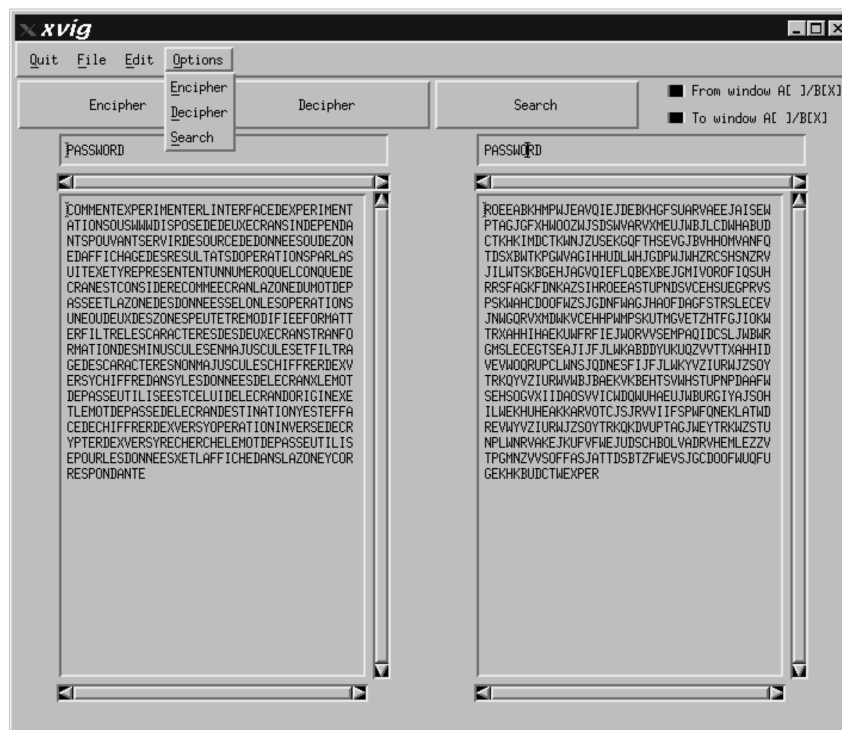
Le sujet de mémoire ne contient aucune demande d'interface graphique. Cependant, un portage rapide en Xwindow du module de décryptage automatique a été réalisée, pour démontrer la réutilisabilité des programmes sources. Du fait de l'existence de l'interface `HTML` réalisée préalablement, cette nouvelle interface n'a pas été poussée au-delà de la simple ébauche. Comme toutefois cette ébauche a le mérite de fonctionner, cette section présente son utilisation ...

C1 - Installation de l'interface

L'interface Xwindow est créée en même temps que les autres exécutables lors de la compilation des programmes sources.

C2 - Utilisation de l'interface

le principe d'utilisation a été calqué sur celui de l'interface `HTML`. Seules quelques options ont été ajoutées telles que la possibilité d'ouvrir un fichier dans une fenêtre, ou celle d'enregistrer le résultat du chiffrage des données.



Interface d'expérimentation Xwindow

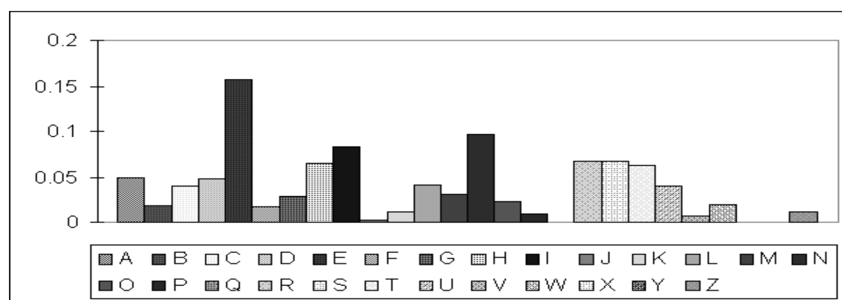
Chapitre 6

Analyse des résultats

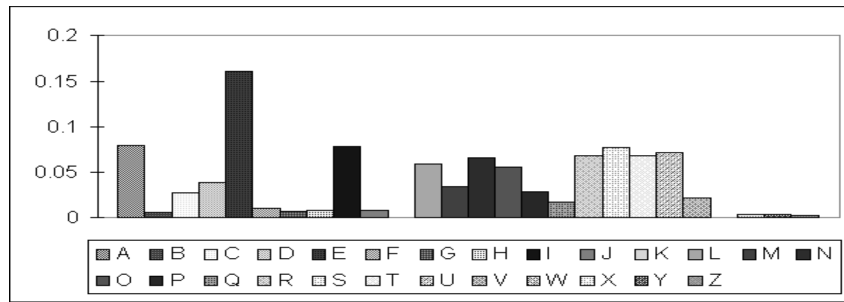
A - Analyses fréquentielles des langues

Le logiciel `frequency` permet de réaliser l'analyse fréquentielle de langues. Les textes utilisés pour les calculs ont été triés selon leur date de publication dans le cas du Français pour permettre de réaliser une comparaison. La taille minimale des fichiers est de 200.000 lettres, fichiers composés de textes tels que ceux de Rousseau, Voltaire, Balzac, Zola ou encore Rostand. Si les fréquences correspondent dans l'ensemble, il est toutefois possible de remarquer certaines évolutions comme l'augmentation de la quantité de `B` ou de `C`, ou bien le recul du `M`. La tendance à la simplification des langues d'une part, et l'assimilation de lexèmes en provenance d'autres langues d'autre part, expliquent certainement ces variations. De la même manière, une étude des bigrammes et trigrammes peut être réalisée, avec par exemple pour l'anglais, l'allemand et le français les trigrammes les plus fréquents :

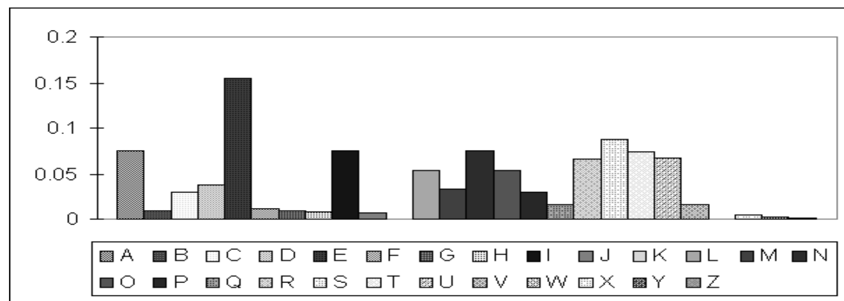
```
the (1.3%) and/you (0.7%) her/hat (0.5%)
ich (1.6%) ein/sch (0.9%) cht/und (0.7%)
ent (0.8%) ait/les (0.7%) que (0.6%)
```



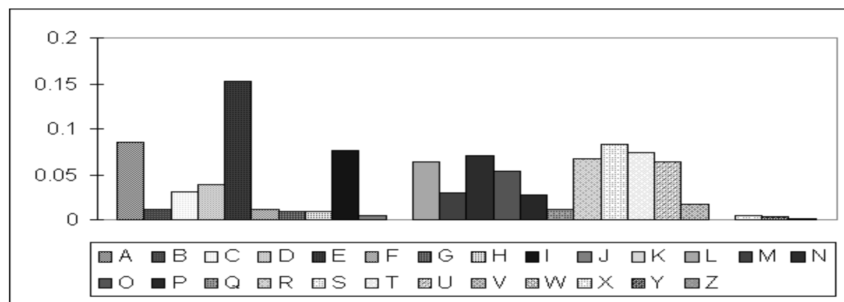
Fréquences de l'allemand



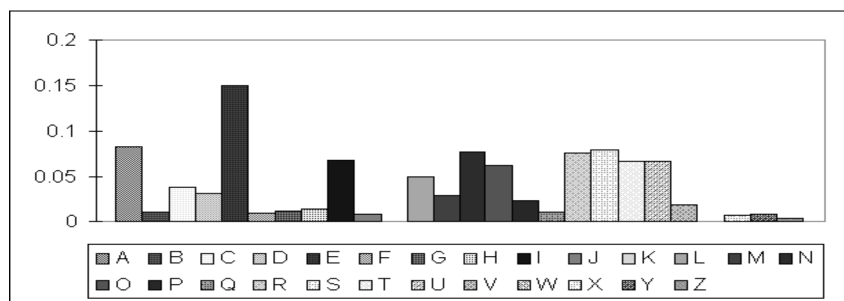
Fréquences du français XVIIème



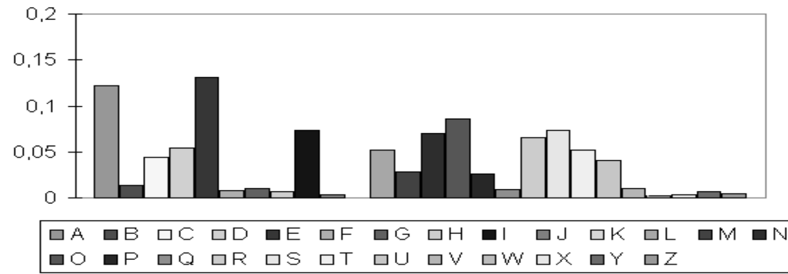
Fréquences du français XVIIIème



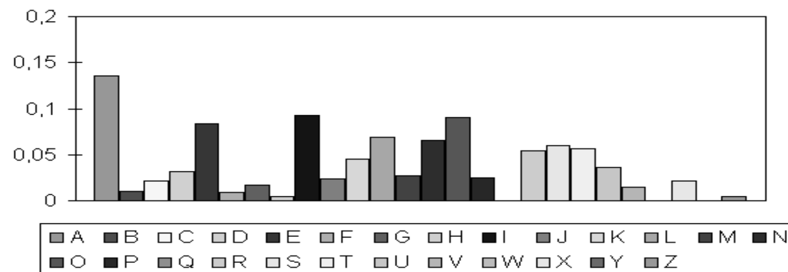
Fréquences du français XIXème



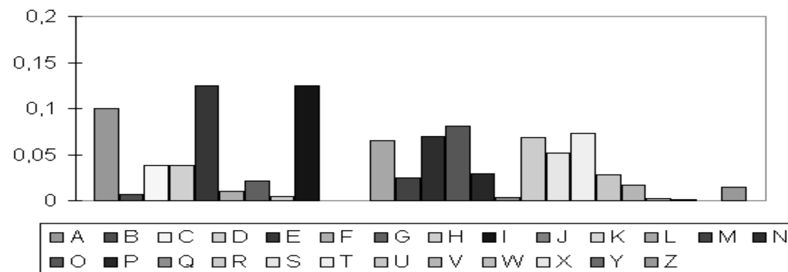
Fréquences du français XXème



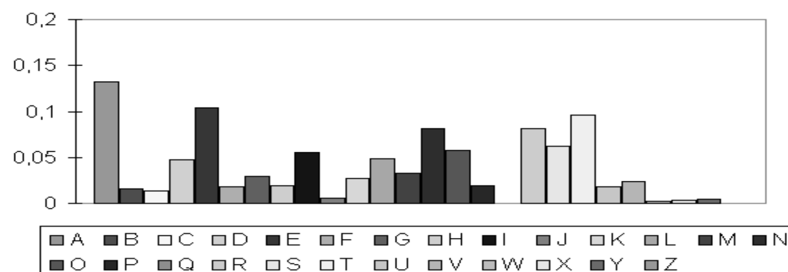
Fréquences de l'espagnol



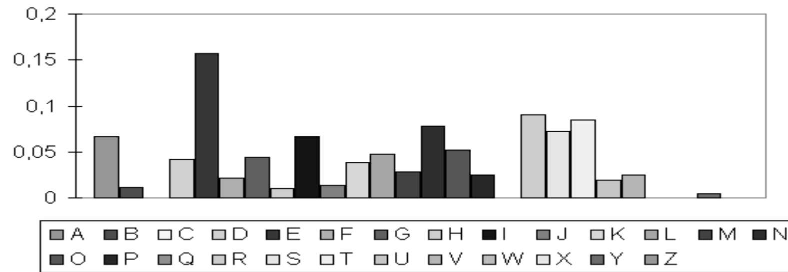
Fréquences de l'italien



Fréquences de l'esperanto



Fréquences du suédois



Fréquences du norvégien

	$XVII^{ème}$	$XVIII^{ème}$	$XIX^{ème}$	$XX^{ème}$
A	.0792135	.0745973	.0849029	.0819487
B	.0057010	.0090169	.0109760	.0108432
C	.0276027	.0301231	.0312133	.0381965
D	.0385741	.0381959	.0385167	.0313010
E	.1612776	.1555366	.1527927	.1504268
F	.0100338	.0113605	.0112783	.0094641
G	.0071453	.0095060	.0096320	.0121178
H	.0081150	.0077054	.0095507	.0133269
I	.0781472	.0741741	.0761314	.0677655
J	.0074529	.0074539	.0045653	.0076733
K	.0000058	.0000658	.0000370	.0000026
L	.0592336	.0540380	.0642798	.0498960
M	.0342265	.0329179	.0301253	.0292226
N	.0667949	.0747749	.0697493	.0767461
O	.0554398	.0543854	.0537380	.0624648
P	.0289415	.0296600	.0270722	.0235623
Q	.0174955	.0156245	.0113411	.0108955
R	.0680781	.0670715	.0675803	.0749029
S	.0762986	.0877245	.0825228	.0782493
T	.0683945	.0740323	.0739118	.0665891
U	.0705067	.0680915	.0642905	.0670793
V	.0221858	.0158381	.0169838	.0185427
W	.0000117	.0000279	.0000620	.0000457
X	.0034305	.0044575	.0042181	.0075033
Y	.0030379	.0025811	.0029604	.0075491
Z	.0026542	.0010380	.0015671	.0036601

Comparaison entre les tables du français

D'un point de vue pratique, ce n'est pas tant le classement exact des lettres qui importe, mais l'existence de groupes dans l'alphabet. Dans un même groupe de fréquences, l'ordre des lettres peut évoluer pour une langue, tandis que la structure générale en groupe n'est pas modifiée. Les listes de la page suivante décrivent les groupes de fréquences pour l'anglais[1], l'allemand[2] et le français.

1. Laurence Dwight Smith, 1943.
2. André Lange et E.-A. Soudart, 1935.

```
{e}{t}{aoin}{srh}{ld}{cumf}{pgwyb}{vk}{xjqz}
{e}{n}{irsat}{dhu}{lg}{ocm}{bfwkz}{pv}{jyxq}
{e}{a}{snr}{iuto}{lcdm}{pv}{hgqbf}{jyxz}{wk}
```

B - Limites de validité des méthodes

B1 - Indice de coïncidence

Comme la plupart des outils construits sur une analyse statistique des lettres, l'indice de coïncidence est limité par la taille des informations. Typiquement, la limite peut être considérée entre 20 et 30 caractères, mais cette valeur est donnée à titre indicatif, puisque dépendant pleinement du texte considéré dans le cas de suites de lettres aussi courtes. En pratique, l'indice de coïncidence permet d'intuiter la longueur de la clé, comme il a été montré précédemment, même dans des cas extrêmes. En revanche, la réduction de la quantité d'information a tendance à augmenter la valeur de l'indice, permettant difficilement de conclure avec certitude. L'exemple suivant correspond aux indices sur un texte anglais de 100 lettres, chiffré avec une clé de longueur 4. Tandis que l'indice augmente pour la longueur 4, il devient difficile d'être assuré du résultat en raison que l'augmentation au-delà de 5. La limite pour cet exemple peut être estimée entre $100/4 = 25$ et $100/5 = 20$.

```
1 : 0.0217384615384615
2 : 0.0339384615384615
3 : 0.0424799938798051
4 : 0.0695384615384615
5 : 0.0615384615384616
6 : 0.0741213623458433
```

B2 - Indice de coïncidence mutuel

L'indice de coïncidence mutuel est probablement la méthode statistique la plus sensible à la taille des informations. Construit sur l'analyse statistique des fréquences, son utilisation devient hazardeuse lorsque les lettres ne sont plus assez nombreuses pour permettre une analyse statistique précise. Les fréquences d'apparition étant multipliées entre elles, le problème se pose d'autant plus. L'utilisation du logiciel `offset` permet d'apporter une estimation de la limite. En effet, ce dernier logiciel est construit sur l'utilisation des indices de coïncidence mutuels pour rechercher le décalage d'un texte chiffré à l'aide d'un chiffrement monoalphabétique. La limite peut être évaluée aux environs de 30 lettres en analysant les monogrammes, mais est réduite par l'utilisation conjointe des fréquences des bigrammes.

B3 - Méthode de Kasiski

La méthode de Kasiski repose sur la présence de répétitions dans le texte chiffré. Il est par conséquent impossible de pouvoir conclure sur un texte chiffré si la longueur du texte n'est pas suffisante. Contrairement aux autres méthodes, la méthode de Kasiski est dépendante de la langue utilisée. En effet, l'analyse fréquentielle des trigrammes montre une importante présence de `the` dans un texte anglais, avec environ 3.5 pourcent. Cela signifie qu'il suffit en moyenne de $100/1.3 = 76$ lettres pour espérer trouver une répétition. Comparativement, pour du français, la longueur nécesaire est en moyenne de $100/0.8 = 125$ lettres, et de $100/1.6 = 62$ lettres pour l'allemand. En pratique, il suffit de quelques répétitions pour pouvoir conclure, et même pour un texte français chiffré avec une clé de longueur n et de longueur de l'ordre de $40n$ il est permis d'espérer trouver des répétitions. L'exemple suivant est réalisé avec un texte français de longueur 140, et chiffré avec une clé de longueur 4.

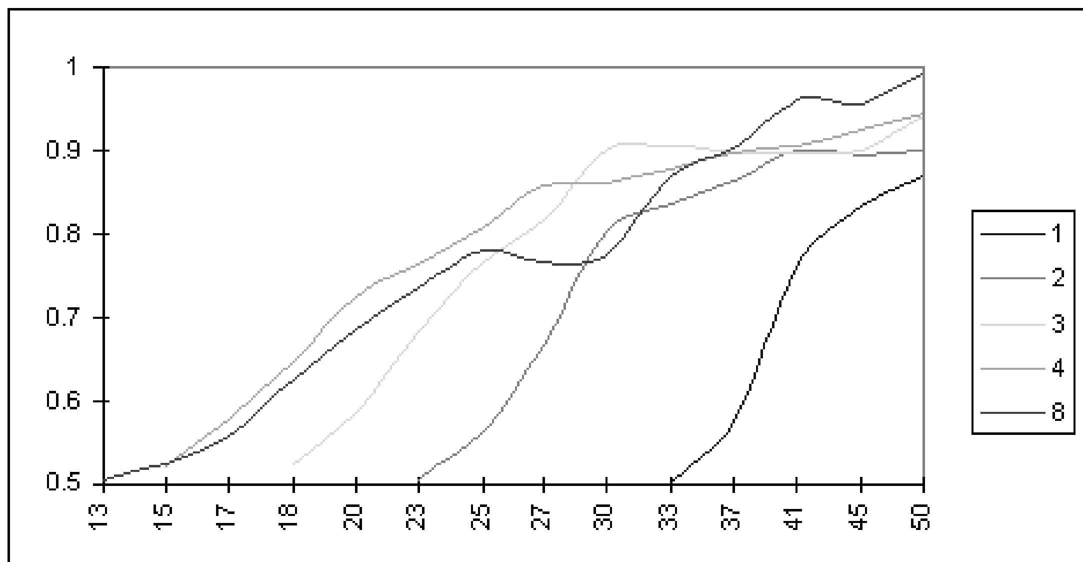
```
xtract -f texte.francais 1 140 | vigenere +BEOS | doubloon -s 3

04  0.735849
108  0.226415
52  0.0188679
32  0.0188679
```

B4 - Décryptage automatique

Les tests de décryptage ont été réalisés par `autotest` à partir d'un texte français. Les résultats obtenus sont pour 50 clés choisies aléatoirement, et servant à chiffrer 50 extraits du texte français à chaque étape. `autotest` réduit progressivement la taille des extraits pour en déduire la limite des possibilités de décryptage, étant données les tables des fréquences de 11 langues. Les résultats ne sont pas à l'abri d'aléas pouvant déformer certaines valeurs calculées, mais le manque de temps n'a pu permettre d'effectuer des tests sur un nombre plus grand d'extraits[3]. Néanmoins, la courbe ci-après indiquant la probabilité pour une lettre de la clé d'être juste, en fonction du rapport $|texte|/|clé|$, pour plusieurs longueurs de clé est instructive.

3. En moyenne 10 étapes sont effectuées pour encadrer la limite, entraînant, pour un temps de calcul de 1 seconde et demie sur un powerPC 603e de chaque décryptage, une durée totale pour une taille de clé donnée de plus de 10 heures ...



Probabilités des succès en fonction du rapport $|texte|/|clé|$

Dans la mesure où il est possible de reconstituer le sens d'un texte malgré quelques erreurs, il a été considéré qu'une probabilité de 0.80 était suffisante pour permettre de décrypter avec certitude un texte. En effet, cela signifie que pour cinq lettres de la clé en moyenne une seule est fausse. Ce pourcentage est très acceptable, puisqu'il suffit très aisément de reconstituer quelques mots pour pouvoir corriger la clé trouvée.

Globalement, les courbes permettent d'apporter plusieurs indications sur la validité de l'utilisation conjointe des méthodes de décryptage. Tout d'abord, le rapport entre la longueur du texte et la taille de la clé doit être supérieur à une limite comprise entre 23 et 30. Cette limite n'est nullement aberrante dans la mesure ou il est difficile de pouvoir réaliser une analyse statistique des lettres sur moins d'une vingtaine de lettres. Ensuite, la limite diminue lorsque la taille de la clé augmente. Ce résultat tient au fait que le texte est transformé en chiffrement monoalphabétique avant d'être comparé avec les tables fréquentielles des langues, le décryptage par cryptanalyse des fréquences des lettres est par conséquent réalisé avec plus d'informations.

En conclusion, l'analyse des courbes tend à montrer l'importance de la méthode de Kerckhoff pour réduire le chiffrement à un chiffrement monoalphabétique. L'implantation de l'algorithme est donc un point crucial dans la cryptanalyse du chiffrement de Vigenère, ce qui peut être rassurant. En effet, comme le montre l'analyse de la complexité en page 55, l'algorithme de recherche des distances relatives aurait été exponentiel si une méthode de limitation n'avait pas été implantée. Cette partie de l'algorithme qui représente une grande partie du temps de calcul total est par conséquent le cœur de la méthode de décryptage du chiffrement de Vigenère, justifiant ainsi l'importance de son coût.

C - Extensions possibles des sources

C1 - Structure de tri (`set`)

La structure de tri est implantée sur les `skiplist` étudiées en licence 1996-1997. Cette structure procure des avantages, tels que le parcours séquentiel des éléments. Le module utilisé possède cependant un certain nombre de faiblesse, dont la plus flagrante est la gestion de la mémoire. Il serait intéressant de pouvoir limiter les allocations répétées. En effet, les longueurs des suites de lettres insérées dans la structure de tri sont constantes. Il pourrait être envisagé d'enregistrer par conséquent un pointeur vers la position dans le texte plutôt que de dupliquer. La fonction de comparaison devant connaître la longueur des suites à comparer, il pourrait être utilisé un système de pseudo-adresses codant la position relative par rapport à l'adresse du texte et la longueur de la suite. Ce codage serait effectué par un module recevant les adresses et gérant plusieurs adresses réelles de base.

C2 - Minimisation de l'erreur d'un système

Le parcours des équations dans l'algorithme implantant la méthode de Kerckhoff est réalisé par une fonction hautement récursive. Il serait bon de pouvoir étudier la possibilité de dérécursifier ce parcours pour réduire les empilements de contextes. Une des possibilités serait par exemple de réserver une zone de mémoire et de se déplacer dans cette zone pour simuler l'empilement de contexte. Une solution plus propre serait bien sûr de pouvoir effectuer un numérotage explicite du parcours, mais un tel numérotage n'a pu être trouvé malheureusement ...

C3 - Tables fréquentielles des langues

Les fichiers des analyses fréquentielles des langues sont des enregistrements binaires dont la qualité dépend de la taille du texte utilisé pour les calculer. Il serait certainement possible d'enregistrer un champ indiquant la taille de ces fichiers pour permettre au logiciel de décryptage de pouvoir tenir compte de la fiabilité de ces tables de fréquences.

D - Complexité des algorithmes

Les complexités de cette section sont écrites formellement. Elles permettent d'estimer les facteurs jouant sur les temps d'exécution de chacun des algorithmes et d'avoir une idée sur ceux qui sont critiques. La complexité d'une fonction est nommée par le nom de la fonction, et les diverses dépendances principales sont indiquées pour justifier les expressions. Les expressions encadrées sont les complexités des fonctions principales. Par exemple pour la fonction de chiffrement, l'expression indique que la complexité du chiffrement est une fonction linéaire de la longueur du texte.

D1 - Chiffrement de Vigenère

Notation $|texte|$ est la longueur du texte chiffré. $cipher_compute = |texte|$

D2 - Méthode de Kerckhoff

Notations N est la longueur de clé maximale, au-delà de laquelle la recherche est découpée en plusieurs recherches, dont les résultats sont ensuite réunis par jointures. $|clé|$ est la longueur de la clé, $|texte|$ celle du texte, K le nombre de lettres de l'alphabet, $depth$ le nombre de propositions demandé, et $largeur$ la longueur d'un morceau de la clé. $kh_compute \approx _analyse$

$$\boxed{kh_compute \leq |clé|\, log_2|clé| * depth\,(depth + N) + \frac{|clé|}{N}\,|texte| + depth\,|texte|\,N.K^N}$$

_analyse() $_analyse = |clé| log_2\,(|clé|) * result_join \cdots$
$\cdots + \frac{|clé|}{N} * \big(K + _frequency_fill + _equation_order + result_create + _find_solutions_{(largeur \leq N)}\big)$
donc $_analyse \leq |clé| log_2\,(|clé|) * depth\,(depth + N) + \frac{|clé|}{N} * \big(|texte| + depth * N^2 K^N\big)$

_frequency_fill() $_frequency_fill = |clé| * ic_compute_{(largeur=1)} + \frac{|clé|^2}{2} * K * ic_mutual_{(largeur=1)}$
d'où $_frequency_fill \approx |texte| + \frac{|clé|^2 K^2}{2}$

_equation_order() $_equation_order = \frac{(|clé|-1)^2}{2} * (K + K * log_2\,(K))$

_find_solutions() $_find_solution \leq K^{largeur} * _test_solution$
d'où $_find_solution \leq depth * largeur^2 K^{largeur}$

_test_solution() $_test_solution = _choice_project + _choice_spread + _choice_quality \cdots$
$\cdots + _choice_maximum$ donc $_test_solution = largeur^2 depth$

_choice_project() $_choice_project = largeur$

_choice_spread() $_choice_spread = \frac{largeur^2}{2}$

_choice_quality() $_choice_quality = \frac{largeur^2}{2}$

_choice_maximum() $_choice_maximum = \frac{largeur^2}{2}$

_filter_solutions() $_filter_solution = result_map + result_sort + result_clean$
donc $_filter_solution \approx depth^2$

D3 - Méthode de Kasiski

Notations $buffer$ est la taille du découpage du texte pour la recherche des répétitions, $|texte|$ la longueur du texte complet, $entry$ le nombre moyen d'insertion dans la structure de tri.
$search_key_length \leq \frac{|texte|}{buffer} \left(buffer * log_2\left(buffer\right) + _compute_{(|texte|=buffer)} + _allpgcd\right) \cdots$
$\cdots + depth * buffer + buffer * log_2\left(buffer\right)$

$$\boxed{search_key_length \ \leq \ |texte|\, buffer.log_2\left(buffer\right) + buffer * depth}$$

_compute() $_compute = |texte| log_2\left(buffer\right)$

_allpgcd() $_allpgcd = \frac{buffer^2}{2} log_2\left(buffer\right)$

_order() $_order = entry\left(log_2\left(entry\right) + depth\right)$

D4 - Indices de coïncidence

Notations K est le nombre de lettres de l'alphabet, $largeur$ le nombre de lettres pour une suite de lettre, $cycle$ la longueur supposée de la clé, soit encore le nombre d'alphabets à considérer. $ic_coincide = ic_frequency_{(largeur=1)} \cdots$
$\ldots + (cycle - 1) * \left(ic_compute_{(largeur=1)} + ic_value_{(largeur=1)}\right)$

$$\boxed{ic_coincide \approx cycle\left(K + \frac{|texte|}{cycle}\right)}$$

$$\boxed{ic_frequency = K^{largeur} + \frac{|texte|}{cycle}\, largeur}$$

ic_compute() $ic_compute = K^{largeur} + \frac{|texte|}{cycle} * largeur$

ic_value() $ic_value = K^{largeur}$

ic_mutual() $ic_mutual = K^{largeur} * _index_shift$

$$\boxed{ic_mutual = K^{largeur}\ (log_K\,(largeur) + 1)}$$

_index_shift() $_index_shift = log_K\,(index) + 1$

D5 - Cryptanalyse du chiffrement de Cæsar

Notations K est le nombre de lettres de l'alphabet, *langage* le nombre de fichiers de référence sur les langues et $|texte|$ la longueur du texte.
$shift_search = ic_compute_{(largeur=1)} + ic_compute_{(largeur=2)} \cdots$
$\cdots + langage * K\, \left(ic_mutual_{(largeur=1)} + ic_mutual_{(largeur=2)} + depth\right)$

$$\boxed{shift_search\ \approx\ langage.K^3 + |texte|}$$

shift_install() $shift_install = K * langage$

shift_remove() $shift_remove = langage$

D6 - Structure de résultat

Notations *depth* est le nombre de résultat exigé, *width* le nombre de champs de la structure `result`.

result_create() $result_create = depth$

result_insert() $result_insert = depth * width$

result_normalize() $result_normalize = depth$

result_sort() $result_sort = \frac{depth^2}{2}$

result_join() $result_join = depth^2 + depth * width$

D7 - Structure de tri dynamique

Notation *entry* est le nombre moyen d'éléments insérés dans la structure.

set_insert()/set_select() $set_insert = set_select \approx log_2 (entry)$

set_normalize() $set_normalize = entry * log_2 (entry)$

Annexe A

A.1 - Installation des logiciels mode texte

A.1.1 - Compilation complète

L'ensemble des programmes sources peut être obtenu en s'adressant par courrier électronique auprès de `dbv@komicom.remcomp.fr`, ou jusqu'en septembre 1997 à l'URL :

```
http://www.emi.u-bordeaux.fr/~morreeuw/universite/TER/vigenere.tgz
```

L'installation complète du logiciel est réalisée par l'exécution du script `configure`, recherchant les librairies nécessaires à la compilation et les chemins d'accès des exécutables. La suite d'actions sur un interpréteur de commandes est donc :

```
gunzip vigenere.tgz
tar xvf vigenere.tar
cd vigenere/sources/
./configure
make
make clean
cd ..
```

Les exécutables sont créés dans le répertoire `vigenere/`, à l'exception de l'interface HTML qui est compilée dans le répertoire `vigenere/html/`.

A.1.2 - Compilation réduite

Dans le cas où les fonctionnalités Xwindow ne sont pas implantées, il est possible de ne compiler que les versions des logiciels pour la ligne de commande. Cette installation est réalisée de la manière suivante :

```
gunzip vigenere.tgz
tar xvf vigenere.tar
cd vigenere/sources/
make -f Makefile.shell
cd ..
```

A.2 - Utilisation des logiciels mode texte

A.2.1 - xtract

Le logiciel `xtract` permet de découper un extrait de texte en utilisant deux arguments entiers représentant le début de la sélection et la longueur. L'option `-k` transforme les caractères accentués en caractères non accentués, en respectant les différences entre majuscules et minuscules.

```
Usage: xtract [-v] [-h] [-k] [-f FILENAME] [-o FILENAME] BEGIN LENGTH

Extract a part of a file

  -f --infile   choose the file (default is STDIN)
  -o --output   save the result as a file (default is STDOUT)
  -k --clean    remove the accents from the text
  -h --help     display this help
  -v --version  output version information
```

A.2.2 - vigenere

Le signe devant la clé de chiffrement permet de choisir entre le chiffrement et le déchiffrement. L'absence de signe est équivalente à la présence du signe `+`. Par défaut le chiffrement est réalisé sur l'alphabet des lettres majuscules, cependant l'option `-a` permet de changer ce comportement en indiquant les deux codes `ASCII` délimitant l'alphabet à utiliser. Tout caractère ne faisant pas parti de l'alphabet est inchangé dans le texte chiffré.

```
Usage: vigenere [-v] [-h] [-f FILENAME] [-o FILENAME]
                [-a MIN MAX] [-|+]PASSWORD

[De]cipher datas with PASSWORD

  -f --infile   choose the file to cipher (default is STDIN)
  -o --output   save the result as a file (default is STDOUT)
  -a --alpha    select a range of characters (default is A-Z)
  -h --help     display this help
  -v --version  output version information
```

A.2.3 - frequency

Le logiciel `frequency` permet de réaliser d'une part l'analyse fréquentielle d'un texte, et d'autre part de générer des fichiers binaires de référence pour les logiciels `minter`, `decipher` et `autotest`. L'utilisation standard du logiciel permet de ne considérer que certains caractères de manière cyclique. L'option `-d` indique la longueur du cycle, et la position à considérer dans le cycle. L'option `-s` permet de comptabiliser des longueurs quelconques de suites de lettres, la valeur par défaut étant 1 pour des monogrammes.

```
Usage: frequency [-v] [-h] [[-f FILENAME] -l LANGUAGE]
                 [[-f FILENAME] [-d LENGTH START] [-s STEP]]

Analyse the content of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -s --step      choose the step of the analyse (default is 1)
  -d --distance  analyse every LENGTH character from START
  -l --language  create a language file LANGUAGE
  -h --help      display this help
  -v --version   output version information
```

A.2.4 - doubloon

`doubloon` recherche les longueurs les plus probables pour la clé ayant servi à chiffrer un texte. Le paramètre `-b` détermine la taille du découpage du texte, sachant que cette taille intervient dans le temps de calcul. L'option `-s` indique la longueur des occurrences multiples à rechercher pour le calcul. Pour terminer, il est possible de préciser le nombre de propositions souhaité. Les résultats sont associés à des coefficients compris entre 0 et 1, qualifiant la probabilité pour qu'une proposition soit plausible.

```
Usage: doubloon [-v] [-h] [-f FILENAME] [-s STEP] [-b BUFFER] [-d DEPTH]

Search some cycles in a file

  -f --infile    choose the file to analyse (default is STDIN)
  -s --step      choose the step of the search (default is 3)
  -b --buffer    choose the length of the search (default is 2048)
  -d --depth     choose the depth of the search (default is 10)
  -h --help      display this help
  -v --version   output version information
```

A.2.5 - coincide

L'indice de coïncidence est calculé sur le texte en fonction de la longueur supposée de la clé. Deux lettres distinctes de la clé de chiffrement conduisant à deux décalages différents, les calculs doivent être réalisés indépendemment sur chacun des chiffrements. L'option `-c` permet par conséquent au logiciel `coincide` de tenir compte de ce facteur.

```
Usage: coincide [-v] [-h] [-f FILENAME] [-c LENGTH]

Coincidence value of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -c --cycle     choose the cycle of the analyse (default is 1)
  -h --help      display this help
  -v --version   output version information
```

A.2.6 - minter

Etant donnée une longueur supposée pour la clé de chiffrement, le logiciel `minter` calcule les valeurs relatives de chacune des lettres de la clé. L'option `-d` détermine le nombre de propositions souhaité.

```
Usage: minter [-v] [-h] [-f FILENAME] [-d DEPTH] CYCLE

Search some rules in a file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 10)
  -h --help      display this help
  -v --version   output version information
```

A.2.7 - offset

`offset` calcule à partir de langues connues les décalages circulaires les plus probables entre ces alphabets, d'une part pour déterminer la clé proposée par le logiciel `doubloon` de manière absolue et non plus relative, d'autre part pour reconnaître la langue du texte. Les fichiers de langues sont créés par le logiciel `frequency`, et le chemin d'accès à ces fichiers doit être indiqué par l'option `-p`. Par défaut, les fichiers sont recherchés dans le répertoire courant.

```
Usage: offset [-v] [-h] [-f FILENAME] [-d DEPTH] [-p PATH]

Search the offset of a file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 10)
  -p --path      choose the path for the language files (default is .)
  -h --help      display this help
  -v --version   output version information
```

A.2.8 - decipher

Le logiciel `decipher` utilise l'ensemble des méthodes de décryptage décrites dans ce mémoire pour rechercher la valeur de la clé ayant servi à chiffrer le texte. L'option `-r` détermine la quantité minimale de texte exigée par lettre de la clé pour réaliser la recherche. Par défaut, le logiciel utilise un facteur de 20 entre la taille du texte et les longueurs de clés étudiées.

```
Usage: decipher [-v] [-h] [-f FILENAME] [-d DEPTH] [-r RATIO] [-p PATH]

Search the password of a cipher file

  -f --infile    choose the file to analyse (default is STDIN)
  -d --depth     choose the depth of the search (default is 1)
  -r --ratio     choose the limit of the possible length of the key
  -p --path      choose the path for the language files (default is .)
  -h --help      display this help
  -v --version   output version information
```

A.2.9 - autotest

`autotest` réalise la même opération, mais de manière automatique en choisissant des clés au hasard et plusieurs extraits de textes. `autotest` réduit progressivement la longueur des extraits pour calculer le rapport limite entre la longueur du texte et celle de la clé. Les coefficients donnés représentent la probabilité pour une lettre de la clé d'être trouvée. Le calcul est effectué pour `NUMBER` clés, en utilisant `DEPTH` extraits de texte. Les deux paramètres obligatoires indiquent la taille de départ de la clé et la taille de fin, pour que le logiciel `autotest` puisse successivement rechercher les limites des tailles de clé compris entre ces deux bornes.

```
Usage: autotest [-v] [-h] [-f FILENAME] [-r RATIO] [-p PATH]
                [-d DEPTH] [-n NUMBER] FIRST LAST

Repeat NUMBER tests of DEPTH parts for key's size from FIRST to LAST

  -f --infile     choose the reference file (default is STDIN)
  -d --depth      choose the depth of the search (default is 10)
  -n --number     choose the number of tests to compute (default is 1)
  -r --ratio      choose the limit of the possible length of the key
  -p --path       choose the path for the language files (default is .)
  -h --help       display this help
  -v --version    output version information
```

A.3 - Utilisation de l'interface HTML

A.3.1 - Installation de l'interface

L'exécutable de l'interface est créé dans le répertoire `vigenere/html/` lors de la compilation complète. L'installation de cette l'interface s'effectue en copiant le contenu du répertoire vers le répertoire destiné à l'héberger. Il n'est pas nécessaire de conserver l'ensemble des fichiers de langues qui ralentissent les recherches et peuvent fausser les résultats, cependant il doit exister au moins un tel fichier dans le répertoire contenant `experiment.cgi`.

Le logiciel doit recevoir des données pour pouvoir fonctionner, aussi la première exécution est lancée depuis le fichier `experiment.html` contenu dans ce même répertoire.

A.3.2 - Utilisation de l'interface

L'interface offre deux fenêtres ayant un rôle identique. Toute opération s'effectue en choisissant l'opération, la fenêtre dans laquelle sont pris les arguments, et la fenêtre recevant le résultat. Le choix est ensuite validé en cliquant sur le bouton `Exécuter`. Tout appel au logiciel entraîne un formatage du contenu des fenêtre réalisant en particulier le filtrage des lettres de l'ensemble des fenêtres.

Les champs utilisés et modifiés dépendent de l'opération demandée. Le chiffrement de la fenêtre `A` vers la fenêtre `B` utilise la clé et le texte de la fenêtre `A` pour afficher le résultat dans la fenêtre `B` en effaçant par ailleurs la clé de la fenêtre `B`. La recherche de la clé de `A` vers `B` utilise le texte `A` pour calculer la clé qui est affichée dans la fenêtre `B`. L'opération `Formater` est une opération vide provoquant un formatage des données lors de l'appel à `Exécuter`.

A.4 - Utilisation de l'interface Xwindow

A.4.1 - Installation de l'interface

L'exécutable de l'interface est créé dans le répertoire `vigenere/` lors de la compilation complète des programmes sources. Pour que l'interface puisse fonctionner correctement, il est nécessaire de lui fournir des fichiers de données fréquentielles sur les langues en les plaçant dans le même répertoire que l'exécutable.

A.4.2 - Utilisation de l'interface

Le fonctionnement de l'interface Xwindow est calqué sur celui de l'interface `HTML`. Seules quelques fonctionnalités telles que la lecture ou l'enregistrement ont été ajoutées. En sélectionnant une fenêtre il est en effet possible d'y lire le contenu d'un fichier, ou bien d'enregistrer le contenu de cette fenêtre.

Annexe B

Les informations ci-dessous sont un extrait des informations obtenue par `gprof` pour le décryptage d'un texte français de 3000 lettres chiffré avec une clé de longueur 4. Seules les informations intéressantes ont été conservées par souci de lisivilité.

```
                              called/total       parents
index  %time    self descendents  called+self    name     index
                              called/total       children

                0.00        3.62      1/1            auto_decipher [3]
                0.00        0.15      1/1            auto_install [23]
-----------------------------------------------
[3]     50.7    0.00        3.62      1          auto_decipher [3]
                0.00        2.48      1/1            _minter [6]
                0.00        0.88      1/1            _offset [11]
                0.00        0.26      1/1            _doubloon [19]
-----------------------------------------------
                0.07        0.79   3432/10764        shift_search [12]
                0.15        1.68   7332/10764        _frequency_fill [10]
[5]     37.6    0.22        2.47  10764          ic_mutual [5]
                0.27        1.73 1395264/1395264     _index_shift [9]
                0.47        0.00 1395264/1397630     _sqr [18]
-----------------------------------------------
                0.00        2.48      1/1            auto_decipher [3]
[6]     34.8    0.00        2.48      1          _minter [6]
                0.00        2.48      6/6            kh_compute [7]
-----------------------------------------------
                0.00        2.48      6/6            _minter [6]
[7]     34.8    0.00        2.48      6          kh_compute [7]
                0.00        2.47      6/6            _analyse [8]
-----------------------------------------------
                                     82              _analyse [8]
                0.00        2.47      6/6            kh_compute [7]
[8]     34.6    0.00        2.47      6+82       _analyse [8]
                0.00        1.86     47/47           _frequency_fill [10]
                0.01        0.59     47/47           _find_solutions [15]
                0.00        0.01     47/47           _equation_order [56]
-----------------------------------------------
                0.00        1.86     47/47           _analyse [8]
```

```
[10]    26.0    0.00        1.86       47          _frequency_fill [10]
                0.15        1.68     7332/10764        ic_mutual [5]
                0.01        0.02      188/300          ic_compute [40]
-----------------------------------------------
                0.00        0.88        1/1            auto_decipher [3]
[11]    12.3    0.00        0.88        1          _offset [11]
                0.00        0.86        6/6            shift_search [12]
                0.01        0.01        6/7            cipher_compute [51]
-----------------------------------------------
                0.00        0.86        6/6            _offset [11]
[12]    12.1    0.00        0.86        6          shift_search [12]
                0.07        0.79     3432/10764        ic_mutual [5]
-----------------------------------------------
                                     6992              _find_solutions [15]
                0.01        0.59       47/47           _analyse [8]
[15]     8.4    0.01        0.59       47+6992     _find_solutions [15]
                0.02        0.57    76135/76135        _test_solution [16]
                                     6992              _find_solutions [15]
-----------------------------------------------
                0.02        0.57    76135/76135        _find_solutions [15]
[16]     8.3    0.02        0.57    76135          _test_solution [16]
                0.08        0.11    76135/76135        _choice_spread [21]
                0.06        0.05    76135/76135        _choice_project [27]
                0.06        0.05    76135/76135        _choice_maximum [26]
                0.05        0.00    70232/73266        result_insert [38]
                0.03        0.00    76135/77974        result_newoffset [44]
                0.01        0.02    76135/76135        _choice_quality [46]
-----------------------------------------------
                0.00        0.26        1/1            auto_decipher [3]
[19]     3.6    0.00        0.26        1          _doubloon [19]
                0.00        0.25        1/1            search_key_length [20]
                0.00        0.01        3/9            ic_coincide [53]
-----------------------------------------------
                0.00        0.25        1/1            _doubloon [19]
[20]     3.5    0.00        0.25        1          search_key_length [20]
                0.00        0.12        1/1            _allpgcd [25]
                0.00        0.09        2/2            _compute [30]
                0.01        0.03        3/5            set_destroy [36]
-----------------------------------------------
                0.01        0.03     6446/30747        dsl_next [43]
                0.01        0.05    11064/30747        dsl_find [34]
                0.01        0.06    13237/30747        dsl_member [33]
[22]     2.4    0.03        0.14    30747          _find [22]
                0.08        0.00  288449/290355        _lcompare [32]
                0.07        0.00  148498/204225        _scompare [31]
-----------------------------------------------
                0.00        0.15        1/1            main [2]
[23]     2.1    0.00        0.15        1          auto_install [23]
                0.00        0.15        1/1            shift_install [24]
-----------------------------------------------
```

```
                0.00      0.15       1/1          auto_install [23]
[24]     2.1    0.00      0.15       1        shift_install [24]
                0.10      0.00      15/15         _stat [29]
                0.00      0.05      11/11         _load [37]
----------------------------------------------
                0.00      0.12       1/1          search_key_length [20]
[25]     1.7    0.00      0.12       1        _allpgcd [25]
                0.00      0.06    3757/6680       set_select [28]
                0.00      0.04    3757/6393       set_member [35]
                0.00      0.02    3934/4023       set_forward [48]
----------------------------------------------
                0.00      0.04    2548/6680       _compute [30]
                0.00      0.06    3757/6680       _allpgcd [25]
[28]     1.5    0.00      0.11    6680        set_select [28]
                0.00      0.04    6680/13237      dsl_member [33]
                0.00      0.04    2264/2343       dsl_insert [42]
                0.00      0.03    4416/11064      dsl_find [34]
-----------------------------------------------
                0.00      0.09       2/2          search_key_length [20]
[30]     1.3    0.00      0.09       2        _compute [30]
                0.00      0.04    2548/6680       set_select [28]
                0.00      0.02    2548/6393       set_member [35]
                0.00      0.02       2/5          set_destroy [36]
                0.00      0.01     375/375        set_insert [65]
----------------------------------------------
                0.00      0.00     164/13237      set_addition [68]
                0.00      0.04    6393/13237      set_member [35]
                0.00      0.04    6680/13237      set_select [28]
[33]     1.1    0.00      0.08   13237        dsl_member [33]
                0.01      0.06   13237/30747      _find [22]
----------------------------------------------
                0.00      0.01    2343/11064      set_destroy [36]
                0.00      0.02    4220/11064      set_member [35]
                0.00      0.03    4416/11064      set_select [28]
[34]     0.9    0.00      0.06   11064        dsl_find [34]
                0.01      0.05   11064/30747      _find [22]
----------------------------------------------
                0.00      0.02    2548/6393       _compute [30]
                0.00      0.04    3757/6393       _allpgcd [25]
[35]     0.8    0.00      0.06    6393        set_member [35]
                0.00      0.04    6393/13237      dsl_member [33]
                0.00      0.02    4220/11064      dsl_find [34]
----------------------------------------------
                0.00      0.02       2/5          _compute [30]
                0.01      0.03       3/5          search_key_length [20]
[36]     0.8    0.01      0.04       5        set_destroy [36]
                0.00      0.02    2343/2343       dsl_delete [52]
                0.00      0.01    2348/6455       dsl_next [43]
                0.00      0.01    2343/11064      dsl_find [34]
----------------------------------------------
```

```
                0.00        0.00       11/73266        _offset [11]
                0.00        0.00       24/73266        _minter [6]
                0.00        0.00     1716/73266        shift_search [12]
                0.05        0.00    70232/73266        _test_solution [16]
[38]     0.7    0.05        0.00    73266          result_insert [38]
                0.00        0.00    17539/17539        _compare [290]
-----------------------------------------------
                0.00        0.04     2264/2343         set_select [28]
[42]     0.5    0.00        0.04     2343          dsl_insert [42]
                0.02        0.00    35193/204225       _scompare [31]
                0.01        0.00     2343/2343         _level [62]
                0.00        0.01     2343/2343         _cell [64]
-----------------------------------------------
                0.00        0.01     2348/6455         set_destroy [36]
                0.00        0.02     4023/6455         set_forward [48]
[43]     0.5    0.00        0.04     6455          dsl_next [43]
                0.01        0.03     6446/30747        _find [22]
-----------------------------------------------
                0.00        0.00       11/77974        _offset [11]
                0.00        0.00       24/77974        _minter [6]
                0.00        0.00     1716/77974        shift_search [12]
                0.03        0.00    76135/77974        _test_solution [16]
[44]     0.4    0.03        0.00    77974          result_newoffset [44]
-----------------------------------------------
                0.00        0.02     3934/4023         _allpgcd [25]
[48]     0.3    0.00        0.02     4023          set_forward [48]
                0.00        0.02     4023/6455         dsl_next [43]
-----------------------------------------------
                0.00        0.00        1/7            auto_decipher [3]
                0.01        0.01        6/7            _offset [11]
[51]     0.2    0.01        0.01        7          cipher_compute [51]


Index by function name

 [3] auto_decipher
 [5] ic_mutual
 [6] _minter
 [7] kh_compute
 [8] _analyse
[10] _frequency_fill
[11] _offset
[12] shift_search
[15] _find_solutions
[16] _test_solution
[19] _doubloon
[20] search_key_length
[23] auto_install
[24] shift_install
[25] _allpgcd
```

```
[28] set_select
[30] _compute
[33] dsl_member
[34] dsl_find
[35] set_member
[36] set_destroy
[38] result_insert
[42] dsl_insert
[43] dsl_next
[44] result_newoffset
[48] set_forward
[51] cipher_compute
```

Bibliographie

[1] J. Hennessy D. Patterson. *Organisation et Conception des ordinateurs.* Dunod, 1994.

[2] Stanley H. Lipson Francine Abeles. Some victorian periodic polyalphabetic ciphers. *Cryptologia*, April 1990.

[3] Ole Immanuel Franksen. Babbage and cryptography. or, the mystery of admiral beaufort's cipher. *Mathematics and Computers in Simulation, North-Holland*, 1993.

[4] Andrew Hodges. *Alan Turing : The Enigma.* Vintage, Random Century, London, 1983.

[5] A. Kerckhoff. *La cryptographie Militaire.* Baudoin, Paris, 1883.

[6] Francine Abeles Stanley H. Lipson. The matrix cipher of c. l. dogson. *Cryptologia*, January 1990.

[7] D. Stinson. *Cryptographie, théorie et pratique.* International Thomson Publishing, 1996.

[8] N. Zimmermann. *Alan Turing ou l'enigme de l'intelligence.* Edition Payot.