Cryptographie classique

Paul Roziere

octobre 5, 2017

Les fichiers référencés sont accessibles dans le répertoire `public` du dépôt gitlab, ou à partir de `http://www.pps.univ-paris-diderot.fr/~roziere/crypto/Cr_class/`

1 Outils

1.1 Programmes standards

`recode` : : par exemple `recode -f latin1..flat <fichier>` élimine les accents dans <fichier>, si ce dernier est au format iso-latin1 ; `recode` fonctionne en filtre :

`cat <fichier>|recode -f latin1..flat`
`bc -l` : : calculatrice programmable en ligne de commande, -l pour charger la librairie mathématique

1.2 Programmes ad hoc

Les utilitaires ne prennent pas en compte, en général, les lettres accentués, les espaces, signes de ponctuations etc. Les majuscules sont traitées comme les minuscules. Les utilitaires fonctionnent en "filtre" (ils prennent par défaut l'entrée standard). Le résultat est affiché sur la sortie standard (on peut la rediriger sur un fichier avec " > "). Vous les trouvez dans le répertoire indiqué.
Les exemples supposent que les binaires sont présents dans le répertoire local `./` .
`./veryflat` : : `veryflat` <fichier> élimine accents, espaces, signes de ponctuations ...
`./testFreqs` : : `testFreqs <fichier>` calcule les fréquences de chaque lettre dans le fichier en entrée. Autre exemple (le répertoire données ayant été rapatrié dans le répertoire local) :

```
cat donnees/*/*.txt|recode -f latin1..flat|./testFreqs
```

`./testFreqs2` : : idem + les fréquences des bigrammes (suites de deux lettres)
`./subst1` : : `subst1 motclef <fichier>` codage par substitution
`./vigCode` : : `vigCode motclef <fichier>` codage de Vigenère (de César si `motclef` n'a qu'une lettre)
`./vigDecode` : : idem (décodage)
`./coincidence` : : `coincidence n fichier_chiffré` Indices de coincidences modulo i pour 1 <= i <= n
`./coincidence2` : : `coincidence2 n fichier_chiffré` Indices de coïncidence mutuelle avec décalages modulo n

1.3 Alphabet

Dans la suite, on code les lettres par leur position dans l'alphabet, numérotée à partir de 0 :

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

2 Données

Divers textes (dans le domaine public) au format txt (avec accents, iso-latin1) sont accessibles dans le répertoire `donnees` (`donnees/*/*.txt`).
Inutile de recopier ces textes (voir ci-dessus l'exemple pour `testFreqs`).

3 Cryptogrammes

Ils sont tous dans le répertoire `cryptogrammes`.

4 Exercices.

Exercice 1
On suppose que l'on connait un couple texte clair/texte chiffré, et le système cryptographique utilisé. Le texte est chiffré avec une clef de 128 bits. Le nombre d'opérations pour le chiffrement est estimé à environ 1000 instructions. Un PC actuel peut effectuer environ 50 000 millions d'instructions par seconde (MIPs). Estimer le temps demandé pour une recherche de la clef par force brute.
Exercice 2 (longueur des clefs)
Pour évaluer la sécurité d'un chiffrement symétrique, on essaye d'estimer le temps et l'investissement financier nécessaire pour retrouver la clef connaissant un couple (texte clair, texte chiffré), suffisamment long pour déterminer la clef de façon unique (attaque "texte clair connu"). Pour beaucoup des chiffrements modernes, il s'agit du temps nécessaire pour retrouver la clef par recherche exhaustive (attaque "force brute"). Quand c'est le cas on définit un *niveau de sécurité* qui est la longueur de la clef si l'attaque par force brute est la seule possible, et qui est estimée en fonction des faiblesses du chiffrement dans les autres cas. Ces attaques sont hautement parallélisables, on peut s'attendre à diviser le temps par deux en doublant les moyens financiers. Le code DES (niveau de sécurité et clef de 56 bits) a été cassé en 1998, en 56 heures et pour un coût de 250 000 $, en construisant une machine spécifique. En adaptant la loi de Moore à la situation on a :
Loi de Moore *Le coût d'une attaque donnée est divisé par 2 tous les 18 mois.*
Même si c'est assez subjectif, on pense que que le niveau de sécurité de 56 bits était suffisant en 1982 (détails dans l'article de Lenstra cité plus bas).

1. En utilisant ces données, estimer l'année limite de protection en fonction du niveau de sécurité (supérieur à 56) et la fonction réciproque.
2. calculer l'année limite de protection pour un niveau de sécurité de 80 bits, 112 bits (triple DES à deux clefs), de 128 bits, le niveau de sécurité nécessaire pour une protection jusqu'en 2008, puis juqu'en 2027.

Ces estimations sont subjectives et elles ne prennent pas en compte des progrès éventuels des techniques de cryptanalyse, des révolutions techniques inattendues. Elles n'ont pas grand sens sur une durée trop longue, au delà de quelques dizaines d'années. Les longueurs de clefs des chiffrements symétriques actuels comme AES (128 ou 256 bits) sont largement au delà des limites de validité de ces estimations.
Voir (pas pendant cette séance !) pour des détails l'article *key lengths* de Arjen K. Lenstra dans le "Handbook of Information Security" (2004).
`http://infoscience.epfl.ch/record/164539/files/NPDF-32.pdf`
Des recommandations pour les longueurs des clefs émanant de divers organismes sont accessibles à partir du site `http://www.keylength.com/`
Exercice 3 (chiffrement de César).

1. Utilisez l'utilitaire `testFreq1` pour une analyse de fréquences des textes proposés. Sauvez le résultat (remarque ces résultats dépendent du choix de textes constitué pour les produire. Ils ne prétendent pas être universels. Ils suffisent pratiquement pour les exercices qui suivent).
2. Pour quelle valeur du décalage le chiffrement de César est-il involutif? Vérifiez avec `vigDecode` (utilisez la commande `echo`).
3. Décryptez le texte `cesarCrypt` (obtenu par chiffrement de César).

Exercice 4 (chiffrement par substitution).

1. Exécutez `echo abcefghijklmnopqrstuvwxyz | ./subst1 clef` puis créez un fichier texte de votre choix auquel vous appliquerez `subst1 clef <fichier>`. Expliquez le fonctionnement de `subst1`.
2. Utilisez, les statistiques obtenues à la première question de l'exercice 1, les utilitaires `testFreq1` et `subst1` pour décrypter le cryptogramme `substCrypt`. Le cryptogramme a été fabriqué avec `subst1` pour un mot clef de nettement moins de 26 lettres, on peut accélérer le décryptage en remarquant que la substitution n'est pas choisie vraiment de façon aléatoire. On peut également se servir de `testFreq2` (sur les textes de références et sur le cryptogramme proposé).

Exercice 5 (chiffrement par permutation). Le texte à décrypter est le suivant :

`emmasartatiqehtmnuorsefiqmeetauiltcapaipstlaoiantcloyropzgzzeizz`

1. Comment peut-on se douter qu'il n'est pas obtenu par un chiffrement par substitution, et qu'il pourrait être obtenu par permutation.

2. Le texte crypté a été obtenu par un chiffrement en colonnes (permutation répétée sur des blocs de taille fixe), après avoir été éventuellement complété, de façon que la taille du texte à chiffrer soit un multiple du nombre de blocs (bourrage). Quelles sont les tailles de bloc (nombre de colonnes) possibles?
3. Le texte a été complété avec la même lettre répétée un nombre suffisant de fois. Servez-vous de ceci pour décrypter le texte (de quelle lettre peut-il s'agir, et, en fonction de celle-ci, quelle peut être la taille de bloc?). (un bourrage mal fait peut grandement aider au décryptage).

Exercice 6 (chiffrement de Vigenère, analyse par recherche de répétitions).
Le texte clair est le suivant :
`ilditquequelquesoitcequelonditilferacequildit`
Utiliser l'utilitaire `vigCode` pour coder ce texte suivant les mots clefs `def defi defin`, et repérer à chaque fois les répetitions de mots de 3 lettres dans le texte chiffré. Que remarquez vous?
Exercice 7 (chiffrement de Vigenère, analyse par indice de coïncidence).

Le texte chiffré `cosetteCrypt` at été obtenue à partir du texte clair `cosette` par chiffrement de Vigenère (pour information, texte original `cosette.ori`).

1. Reconstituez la clef de chiffrement (vérifiez avec les outils `vigCode`, `vigDecode`).
2. Utilisez sur le fichier cosetteCrypt l'utilitaire `coincidence`, pour divers entiers, expliquez le résultat (consultez également les résultats de l'analyse des textes proposés à l'exercice 1).
3. Utilisez sur le fichier cosetteCrypt l'utilitaire `coincidence2`, pour l'entier longueur de la clef, expliquez le résultat.

Exercice 8 (chiffrement de Vigenère, analyse par indice de coïncidence).

1. Comment peut-on écarter les chiffrements par substitution ou par permutation pour l'un des cryptogrammes `vigCrypt vig2Crypt vig3Crypt`?
2. Décryptez ces cryptogrammes.

Exercice 9 (chiffrement de Vigenère, mot probable et clef non aléatoire).

1. Décoder le fichier `cosetteCrypt` par le code de Vigenère en utilisant le mot clef "thenardier", puis les mots clefs obtenus par permutation circulaire ("henardiert", etc), jusqu'à faire apparaître le mot clef utilisé pour coder `cosetteCrypt`. Expliquez.
2. On sait que le fichier `meteoCrypt` est un bulletin meteo, codé par chiffrement de Vigenère, et qui contient le mot clef "temperature". Utilisez ceci pour décrypter très rapidement ce fichier.

Exercice 10 (chiffrement de Vigenère, mot probable et clef non aléatoire)
Décoder le message chiffré suivant (fichier `decembreCrypt`), sachant qu'il est codé par la méthode de Vigénère, et que le message initial contient le mot `decembre`.

`uypsuihmyujgtnxajsxlcnmqemxlrkmwipcnodwamrmyuwuflnugfhibnojwa`

Décrypter l'un des cryptogrammes `vigCrypt1`, ..., `vigCrypt6, vigCrypt7` en utilisant les indices de coïncidence et les indices de coïncidence mutuels. Puis rédiger la solution de cet exemple, en expliquant la méthode de décryptage. Quelqu'un qui n'a jamais entendu parler des indices de coïncidence doit comprendre votre texte (vous devez les définir en particulier). Par contre il ne s'agit pas de faire un manuel des outils utilisés.