UNIVERSITE DE BATNA 2

FACULTÉ DES MATHMATIQUES ET INFORMATIQUE

N° .....

Thèse pour l'obtention du grade de

Docteur en Sciences

Spécialité : Informatique

Présentée et soutenue par

Tahar MEKHAZNIA

Analyse cryptographique par les méthodes heuristiques

Thèse dirigée par

Prof. Abdelmadjid Zidani, Directeur de Thèse, Université de Batna, Algérie

Prof. Mohamed El Bachir Menai, Co-Directeur de Thèse, King Saud University, Ryadh, KSA

Soutenue le 12 février 2017

JURY

Président	*Professeur*	BILAMI Azeddine
Rapporteurs	*Professeur*	ZIDANI Abdelmadjid
	Professeur	MENAI Mohamed El Bachir
Examinateurs	*Professeur*	AMIRAT Abdelkrim
	Professeur	MOKHATI Farid
	Professeur	LAOUAR Mohamed Ridda

Résumé

La *cryptanalyse* est l'art de l'étude des chiffrements et leurs concepts annexes, de les casser sans connaissance au préalable des algorithmes de chiffrements et des clés utilisées. Son principe réside dans l'utilisation d'outils mathématiques adéquats pour la réussite des attaques. La *force brute* étant l'attaque la plus sure; elle tente toutes les possibilités sans restriction mais, nécessite en contrepartie d'avantages de ressources et donc, moins de succès en pratique. D'autres alternatives moins gourmandes sont disponibles en littérature : la cryptanalyse *linéaire* et *différentielle*, capables de briser une variété de chiffrements. Néanmoins et, compte tenu de leur conception spécifique, paraissent inefficaces à l'égard des cryptosystèmes modernes. Ce problème est classé comme *NP-Difficile*, a été longuement cible de diverses attaques; les résultats, apparus depuis quelque temps demeurent jusqu'à lors restreints, spécialement lors de la manipulation de larges instances où la consommation de ressources accroît avec la taille du problème.

D'un autre côté, les metaheuristiques regroupent un ensemble de méthodes et techniques destinés à explorer des espaces hétérogènes en vue de résoudre des problèmes d'optimisation difficiles. Ceci est dû essentiellement à leur capacité de converger rapidement avec un usage modérée de temps de calcul. Faisant partie des metaheuristiques, les algorithmes évolutionnaires sont dotés d'un potentiel efficace pour le traitement des grandes instances et paraissent aptes à produire des solutions approximatives pour une large variété de problèmes d'optimisation avec une consommation acceptable de ressources.

De ce fait, le travail présenté se focalise sur l'analyse des caractéristiques des chiffrements et améliore les outils de cryptanalyse dans le but de favoriser les attaques en optimisant les ressources. Cette analyse englobe la manière de recherche de clés par analyse des statistiques de fréquences de caractères, stratégies de paramétrage de l'environnement et diversification des données de tests; l'outil applicatif utilisé est un ensemble d'algorithmes heuristiques basés sur la recherche locale, le branchement et certaines metaheuristiques à population. Le résultat renvoi une synthèse quant à la classification de ces algorithmes selon leur efficacité à résoudre de tels types de problèmes. La modélisation intègre diverses propriétés liées aux chiffrements, notamment les outils de reconnaissance des langages de chiffrements, le type des chiffrements et les propriétés des clés utilisées. Le code d'expérimentation élaboré permet de gérer un nombre consistant de paramètres: clés, textes de chiffrements, tables statistiques de fréquence de caractères, ainsi que les paramètres liés à l'environnement: taille de données, temps exécution, valeurs de paramètres de diverses metaheuristiques utilisées. L'ensemble de ces tâches a été intégré au sein d'un outil de recherche de clés automatisé permettant de gérer les diverses situations et de contourner l'intervention humaine durant tout le cycle de traitement. Ce concept de synthèse, absent dans la majorité des travaux similaires en littérature, permet en grande partie d'améliorer les conditions d'expérimentations et favoriser la convergence de la solution.

Enfin, le travail est couronné par une synthèse de comparaison de performances entre diverses techniques de recherche utilisées aidant ainsi les cryptanalystes à mieux choisir leurs stratégies d'attaques en regard de chaque type de chiffrement ciblé.

Mots clés : Cryptanalyse, metaheuristiques, cryptographie classique, cryptographie symétrique

Sommaire

Figures et illustrations

Tables

Algorithmes

Notations

L	langage littéraire donné
$\mathcal{A}$, *Alpha*	alphabet de caractères de L
$\mathcal{A}_a$	alphabet de a caractères
P	ensemble de textes clairs
p	texte clair donné de P
p_i	i-eme caractère ou bit du texte p
Plain[n]	tableau contenant une chaine claire de caractères ou de bits de taille n
s_t	séquence s de t caractères ou de bits de p
Seq[t]	tableau contenant la séquence s_t
C	ensemble de textes chiffrés
c	texte chiffré donné de C
c_i	i-eme caractère ou bit de c
Cipher[n]	tableau contenant une chaine chiffrée de caractères ou de bits de taille n
K	ensemble de clés
k_t	clé donnée de taille t appartenant à K
k_i	i-eme caractère ou bit de k
T_k	transformation associée à la clé k
$\xi=(P,C,k,T,T')$	cryptosystème ou protocole d'échange
$c=T_{k(p)}$	transformation permettant d'obtenir un chiffré c à partir d'un texte clair c en utilisant la clé k
$p=T'_{k(c)}$	Transformation inverse
I_p	indice de coïncidence d'un texte p
S	espace de recherche ou ensemble de solutions à un problème donné
s_i	i-eme solution de S
s^*	solution optimale de s
$V(s)$	voisinage de la solution s
$F(s)$	fonction de cout ou fonction objectif de la solution s
$O(n)$	grand O de n
A*	algorithme *path-finding*

Remerciements

Je tiens en premier lieu exprimer ma profonde reconnaissance à Monsieur Abdelmadjid Zidani, Professeur à l'université de BATNA, Algérie pour le suivi régulier, le cadre scientifique rigoureux et l'esprit d'autocritique qu'il a su m'apporter. Je lui suis très reconnaissant pour m'avoir favorisé un cadre de recherche agréable m'ayant permis d'achever ce travail dans les meilleures conditions.

Je tiens également à remercier Monsieur Mohamed El Bachir Menai, Professeur à l'université King Saud University, Riadh, KSA,pour avoir orienté ma recherche et pour l'intérêt qu'il m'a témoigné tout au long de ce parcours.

J'espère avoir été à la hauteur de leurs espérances.

J'adresse également mes remerciements les plus sincères au Président pour l'honneur qu'il me fait en acceptant de présider le jury de cette thèse.

Mes vifs remerciements vont aux membres du jury qui ont accepté de consacrer une part de leur temps pour examiner mon travail ;

Merci à celles et ceux qui auront à lire tout ou une partie de ce manuscrit et qui y trouveront un quelconque intérêt;

Enfin, un grand merci est adressé également à tous ceux qui m'ont octroyé part de leur soutien et encouragements afin arriver au terme de cette thèse ;

Tous ceux qui m'ont incité même involontairement à faire mieux, veuillez trouver ici le témoignage de ma profonde gratitude.

Publications

Mekhaznia and Zidani, A new approach of known plaintext attack with Genetic Algorithm, *Wseas Transactions on Computers*. (paper in press)

Mekhaznia and Zidani, Swarm intelligence algorithms in cryptanalysis of Simple Feistel Ciphers, *Int. Journal of Information and Communication Technologies*, Inderscience publishers. http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijict

Mekhaznia and Zidani, Bio-inspired algorithms for attack of block ciphers, *Recent advances in computer science*, Vol. 32. .pp 108-113. ISSN 1790-5109. www.inase.org/library/2015/.../COMPUTERS-15.pdf

Mekhaznia and Zidani, Wi-Fi security analysis, Procedia Computer Science. Direct Sciences. *In proceeding of the AWICT'15*. Vol. 73. pp 172-187. http://www.sciencedirect.com/science/article/pii/S1877050915034705

Mekhaznia and Zidani, BAT algorithm for Cryptanalysis of Feistel cryptosystems, *Int. Journal of Intelligent Systems and applications in Engineering*. Vol. 3.No 2. pp 82-85. http://ijisae.atscience.org/article/view/1065000184

Mekhaznia and Menai. Cryptanalysis of classical ciphers with ant algorithm. *Int. J. of Metaheuristics (Intersciences),* 3 (3) 2014. http://www.interscience.com/info/inarticle.php?artid=65159

Mekhaznia and Zidani. Swarm intelligence algorithms in cryptanalysis of Simple Feistel Ciphers. *IT4OD '14,* 19-20/10/2014. Tebessa, Algeria. http://abgattal.info/IT4OD14/IT4OD%202014_Proceedings.pdf

Mekhaznia, and Zidani. Wolf Pack Algorithm for Cryptanalysis of Symmetric Cryptosystems. *META'14,* 27-31/10/2014, Marrakech, Morocco. http://meta2014.sciencesconf.org/conference/meta2014/pages/Program_META_15.pdf

Mekhaznia and Zidani. Algorithme hybride basé sur l'intelligence des essaims pour la cryptanalyse du chiffrement Feistel. *ICA2IT'14,* 10-12/3/2014, Ouargla, Algerie. http://manifest.univ-uargla.dz/documents/Archive/Proceding/proceedings%20de%20les%20seminaire/Faculte%20nouvelles%20%20technologie%20d%20information%20et%20de%20la%20comunication/10-11-12-03-2014.pdf

Mekhaznia and Zidani. An Hybrid BAT algorithm for attack of Feistel Cryptosystems. *ICAT'14,* 12-15/8/2014, Antalya, Turkey. http://conference.atscience.org/#program

Mekhaznia. Nature inspired heuristics for attack of simplified DES algorithm. *SIN'13,* 26-28/11/2013, Aksaray, Turkey. http://www.sinconf.org/sin2013/

Mekhaznia. Formalisation d'empreintes digitales à l'aide d'outils basés sur le modèle de mixture gaussienne. *CNDM'13,* 25-26/11/2013, Tebessa, Algerie.

Mekhaznia, Menai and Zidani. Algorithmes biologiques pour la cryptanalyse du chiffrement Feistel. *COSI'13,*9-11/6/2013, Alger, Algerie. http://www.isima.fr/cosi/cosi2013/acceptedPosters.php

Mekhaznia, Menai and Zidani. Nature inspired heuristics for cryptanalysis of Feistel ciphers. *ICIST'13,* 23-24/3/2013, Tanger, Morocco. http://ardeigroup.e-monsite.com/medias/files/conference-programme-icist-13.pdf

Mekhaznia and Zidani. BAT Algorithm for cryptanalysis of Data Encryption Standard Cryptosystem. *ICSENT'13,* 21-23/12/2013, Hammamet, Tunisia.

Mekhaznia, Menai and Zidani. Metaheuristiques pour la cryptanalyse du chiffrement par substitution et transposition. *ICIST'12,* 24-26/3/2012, Sousse, Tunisia.

Mekhaznia. Natural heuristics for cryptanalysis. *JISR (Dline),*2(3)2012.

Mekhaznia, Menai and Zidani. Group swarm optimization for cryptanalysis of classical ciphers. *META'12,* 27-31/10/2012, Sousse, Tunisia.

Mekhaznia, Zidani and Menai. Cryptanalysis of ciphertext substitution using ACO algorithms. *ICMOSS'10,* 29-31/5/2010, Tiaret, Algeria.

Abstract

Cryptanalysis is the art and science of studying cryptosystems and ciphers in order to detect, analysis and repairs their weakness. It denotes also the concept by which, an unauthorized user gain access to the hidden content of an encrypted data without necessary knowing the key or the algorithm which normally used for decryption. This leads, for example, to convert a ciphertext into a plaintext without using the decryption key. The use of cryptanalysis on ciphertexts is known as an attack. The cipher will be broken and its security becomes obvious when an attack is successful. So, the design of high secure cryptosystems presents a challenging area for researchers in the discipline of computer and data security, especially with the increasing of communication networks where it becomes difficult in practice, to keep secret encryption keys that must be communicated between the sender and the receiver through public networks. The principle of cryptanalysis includes the correct mathematical tools necessary to provide the right attack. This fact allows the evaluation of the cryptosystems efficiency and therefore, performs more robust algorithms.

In another way, the metaheuristics, a branch of computational intelligence have got significant importance in determining efficient solutions of different real world problems; This is mainly due to their effectiveness to handle cryptanalysis problem and appears an efficient way to break complex ciphers. Swarm intelligence algorithms, a branch of heuristic techniques, are well-known metaheuristics which demonstrate a potential in determining efficient solutions of combinatory problems such cryptanalysis. Various researches shown that algorithms based swarm intelligence have successful potential to handle wide instances and may be adapted to produce approximate solutions for variety of optimization problems. They use an intelligent system that offers an independence of movement of agents which tend to replace the preprogramming and centralized control.

The purpose of this work is to provide, a detailed study about the performance of metaheuristics for cryptanalysis of some variant of classic and symmetric cryptosystems. Experiments were accomplished in order to study the effectiveness of these algorithms in solving the considered problem and underlined encountered problems.

1. Introduction

Depuis toujours, l'homme a éprouvé la nécessité d'avoir des secrets personnels ou d'échanger des informations privées avec d'autres personnes d'une manière confidentielle, il se servait donc d'outils permettant de garder ses confidences hors d'atteinte de yeux indiscrètes : signes et symboles intelligibles, figures ou couleurs, usage d'expressions ou phrases convenues d'avoir un sens spécifique qui diffère de l'ordinaire, etc. La progression de ces outils primitifs à travers le temps, a permis de concevoir des règles de sécurité plus efficaces et plus logiques qui ont donné naissance à la cryptographie.

Etymologiquement, la cryptographie désigne la manière de la protection de l'information. Son histoire était aussi longue que celle de l'homme car elle est née conjointement avec la communication et ce, il y a 4000 ans[1] en Egypte ancienne. A cette époque, elle n'était pas un outil de sécurité mais plutôt un moyen de contact solennel réservé à la haute classe car les communautés étaient en grande partie pauvres et illettrés et avaient des moyens de communication limités.

A l'origine, la cryptographie était plutôt un art : le secret des communications était primitif pendant des siècles[2] ; il est basé sur de concepts de sténographie qui visent à dissimuler l'existence

1. Le premier support chiffré connu est une tablette d'argile qui remonte à 1600 av. J.-C. retrouvée en Irak sur laquelle est gravée sa recette secrète en supprimant des consonnes et en modifiant l'orthographe des mots réalisée apparemment par son fabriquant.
   L'écriture sur les crânes rasés des esdaves du roi de Babylone date en 600 av. J.-C.
   Les scribes hébreux emploient l'algorithme *Atbash* pour transcrire le livre de Jeremiah en 500 av. J.C.
   Les grecs utilisaient le *scytale* en 487 av. J.-C

2. Le premier système cryptographique est dédié à *Polybe*, historien grec et ce, il y a 150 av. J.-C. C'est un procédé de chiffrement utilisant un système de transmission basé sur un carré de 25 qui utilise l'alphabet tout en combinant deux lettres en un, soit le I/J ou le V/W. La technique se distingue de la substitution classique par la conversion des lettres en chiffres et la représentation de chaque lettre par deux éléments séparés. En 50 av. J.-C. *Jules César* utilisa une substitution modeste de l'alphabet basée sur le décalage pour les communications gouvernementales.

du message lui-même, ce qui nécessite de l'imagination, de l'intuition et de l'invention[3]. Son utilisation était restreinte à certaines catégories de population ayant la faculté d'imaginer et de développer de telles idées et éprouvant le besoin de les utiliser, notamment la diplomatie ou l'armée ; les concepts utilisés étaient basiques et spécifiques, reflétant le peu de moyens d'échanges d'informations existant à l'époque. Ce procédé s'est révélé inefficace si le message a été découvert et qu'il fallait souvent changer de stratégie. Au fil de l'histoire, les techniques cryptographiques ont subi des améliorations vers de nouvelles techniques permettant de préserver le contenu des communications notamment dans le cas où celles-ci étaient interceptées et ce par l'usage séparément d'une *clé* et d'un *algorithme de chiffrement*.

Au cours du siècle dernier et en particulier vers les années 70, la cryptographie et, conjointement au développement de l'outil de calcul a connu une évolution profonde la transformant en science largement utilisée dans les échanges de données. La tâche des concepteurs de *systèmes de chiffrement* (ensemble de données, clé et algorithme) est désormais plus difficile quand il s'agit de préserver la confidentialité et l'intégrité des données échangées via les réseaux publics. Les systèmes de chiffrement conçus doivent être en mesure de résister à toute tentative consciente d'attaques; les concepteurs se sont servis donc de méthodes issues de branches voisines notamment la théorie des nombres, les courbes elliptiques, la recherche opérationnelle ainsi que les certains outils mathématiques pour élaborer des algorithmes de chiffrement efficaces: c'est alors que débuta l'âge de la *cryptanalyse*.

La cryptanalyse est donc une manière simple de révéler le contenu d'un texte chiffré sans avoir connaissance de l'algorithme de chiffrement ni de la clé utilisée. Bien qu'elle paraisse comme un outil destructeur, c'est plutôt une discipline ardue et excitante car elle permet de renseigner sur les propriétés cachées des systèmes de chiffrements et de leurs faiblesses aidant ainsi les concepteurs à élaborer d'avantage de procédés plus robustes. Elle s'est basée en premier lieu sur le hasard et l'intuition quant au choix des clés utilisées, mais vite développée en science utilisant des concepts mathématiques adéquats notamment les techniques d'optimisation, la programmation linéaire et des éléments d'analyse statistique comme outils de choix des clés afin de concevoir des attaques réussies moyennant des ressources optimisées.

Les techniques de cryptanalyse sont nombreuses et dépendent en grande partie des éléments dont dispose l'attaquant, à savoir l'algorithme de chiffrement, des échantillons de textes clairs et chiffrés et dans la mesure du possible, la nature de la clé et sa longueur. Néanmoins, et en cas général, seul le texte chiffré est à disposition de l'attaquant. Il doit donc user de cette information uniquement pour générer certaines clés permettant de découvrir des fragments de texte clair sans détails supplémentaires, autrement, la *recherche exhaustive* de clés parait la seule issue qui garantit la solution ; elle est sûre mais consomme d'avantage de ressources et n'est utilisée qu'en alternative de rechange en absence d'autres moyens plus bénéfiques et ce, pour des instances de taille raisonnable, étant donné que le problème de la cryptanalyse est classée dans la catégorie des problèmes difficiles où il est pratiquement impossible de le résoudre en un temps raisonnable avec des méthodes exactes. Pour cette raison, diverses autres méthodes ont été proposées ; la plupart entre elles nécessite la connaissance de certains fragments de textes clairs et chiffrés

---

En 200 Av.J.-C., apparaît Le *papyrus* de Leyde, un ancien manuscrit de l'alchimie. Il utilisa un algorithme de chiffrement pour cacher les parties importantes de certaines recettes.

3. En Chine on écrivait les messages sur une soie très fine glissée dans une petite boule recouverte de cire, avalée ensuite par le messager ; en Italie on utilisait une encre absorbée par la coquille d'un œuf dur, …

comme données de départ, telle la cryptanalyse différentielle (Biham & Shamir, 1993b) basée sur la l'analyse d'une perturbation de résultat causée par une différence entre diverses données, la cryptanalyse linéaire (Matsui, 1993) qui consiste à l'analyse de l'approximation affine constatée dans le résultat par rapport aux données d'entrée ou le *slide attack* (Biryukov & Wagner, 1999) permettant de surmonter la complexité relative au nombre de tours d'un chiffrement symétrique. Ces alternatives paraissent capables de briser une grande variété de chiffrements, néanmoins, et compte tenu de leur niveau de puissance réduit, restent inefficaces envers une large classe de systèmes cryptographiques modernes.

Au cours des années récentes, la recherche en cryptanalyse s'est orientée progressivement vers les algorithmes dits : *intelligents* conçus à base des techniques heuristiques. Cette démarche a été encouragée par l'efficacité ces techniques quant aux attaques de grandes instances représentées par les chiffrements dont la clé est assez longue (BinAhmad & BinMaarouf, 2006) (Grosek & Zajak, 2009) (Urszulan & Dworak, 2014). Les techniques heuristiques polyvalentes ou metaheuristiques ont été les mieux sollicitées dans ce domaine; elles sont inspirés en général de phénomènes naturels et consistent à explorer un vaste espace de recherche en quête de la solution optimale. Elles possèdent la faculté de considérer uniquement les chemins jugés intéressants qui mènent au but et de contourner les minimas locaux. Les metaheuristiques inspirées de la nature (Beni & Wang, Swarm Intelligence in Cellular Robotic Systems, 1989), simulant le comportement des êtres vivant en collectivité, caractérisée par une auto-organisation régie par certaines règles communes inspirées par l'intuition biologique, permettent un libre mouvement de chaque individu sans nécessité d'un contrôle centralisé.

Vu leur consommation acceptable de ressources et leurs solutions avantageuses, les metaheuristiques ont été largement utilisées pour résoudre des problèmes difficiles évitant ainsi l'explosion combinatoire[4] liée aux méthodes exactes, malheureusement, aucun argument relatif à l'optimalité de la solution générée par une metaheuristique ne peut être avancé, étant donné que, la recherche devient vaine si dans un espace d'exploration, un croisement entre la solution locale et globale se produit (Olamaei, Niknam, & Gharehpetian, 2008). En cryptanalyse, ces techniques demeurent l'outil préféré des attaques en l'absence d'alternatives plus efficaces; la première implémentation d'attaques aux chiffrements ayant utilisé les algorithmes génétiques remonte aux années 93 (Spillman, Janssen, Nelson, & Kepner, 1993) et depuis, la recherche en cet axe ne cesse de progresser en passant par les méthodes de recherche locale (Forsyth & Savafi, 1993) (Clark, 1998), techniques de branchement (Tariq & Faez, 2014) jusqu'aux metaheuristiques à population dont le détail est présenté au chapitre 4.

Les résultats des recherches obtenus étaient en général distincts et restreints à leurs environnements d'expérimentation; les données utilisées sont générées d'une manière intuitive ou inspirées à base d'autres travaux similaires. Il est donc difficile de les reproduire ou de les vérifier en l'absence de modèle mathématique adéquat. Nous pensons dans ce contexte que la meilleure manière de justifier les données choisies ou les valeurs des différents paramètres d'implémentation est de procéder à une expérimentation permettant de couvrir une large gamme de valeurs de paramètres environnementaux et données du problème, à savoir : type et taille de la base de tests utilisée, variables spécifiques à chaque metaheuristique utilisée et enfin, un

4. L'espace de solutions du problème du voyageur de commerce par exemple, croit en (n-1)!, où n est le nombre de villes à visiter. Avec seulement 50 villes, il faudra évaluer 49! trajets, soit 6.08×10^{62}, ce qui dépasse rapidement les capacités de calcul de n'importe quel ordinateur. C'est l'explosion combinatoire.

paramétrage varié de l'environnement d'expérimentation permettant ainsi de sélectionner des données offrant des résultats optimums. Le détail des divers paramètres est présenté au chapitre 7.

Ainsi et, à travers de ce travail, nous essayons de reproduire des tests similaires de celles présentés en littérature en utilisant un environnement comportant un ensemble hétérogène de chiffrements. Les algorithmes d'attaques ont été implémentés à base d'une variété de metaheuristiques. Nous pensons par ce fait, couvrir une large gamme de données et contribuer en partie à déterminer le meilleur paramétrage des variables environnementales permettant de tirer profit d'une attaque, en autres termes, de répondre à la question pertinente :

Afin de révéler le secret d'un chiffrement donné moyennant un minimum de ressources, quelle est la meilleure technique d'attaque à adopter et dans quelles conditions ?

Dans le souci d'apporter des éléments de réponse à la question, deux niveaux de tests expérimentaux ont été accomplis : le premier test consiste à essayer de fixer les meilleures valeurs de paramètres de chaque metaheuristique utilisée, ce qui nous permet de répondre à la question dérivée :

Quelles valeurs de paramètres permettant à une metaheuristique de mieux performer ?

Le deuxième niveau de tests se focalise au classement des metaheuristiques selon leur performance quant aux chiffrements utilisés. Ce niveau de tests nous permet de répondre également à la deuxième question dérivée:

Sur quel chiffrement une metaheuristique performe le mieux ?

L'environnement de l'expérimentation utilisé consiste en un ensemble de chiffrements produit par diverses techniques classiques et symétriques de cryptographie, à savoir : la substitution monoalphabétique et polyalphabétique, la transposition et le chiffrement par blocs et pour le volet symétrique, nous utilisons l'algorithme DES avec certaines de ses variantes. L'outil d'attaque est inspiré de plusieurs metaheuristiques sélectionnées des différentes classes existantes, à savoir : la recherche locale, le branchement et les metaheuristiques à population. Les expérimentations nous permettent certainement de clarifier les diverses insuffisances liées aux attaques des chiffrements et de contribuer à l'amélioration des outils de cryptanalyse automatique dans le but d'optimiser les ressources. La conclusion du travail présente une synthèse des tests réalisées et discute les diverses idées et perspectives.

1.1 Problématique

Bien que certains algorithmes de chiffrements présentent une résistance acceptable aux attaques, ils ne le seront certainement pas dans quelque temps, voir dans certains environnements. Leur robustesse va manifestement céder face aux techniques d'attaques basées sur des outils statistiques et mathématiques qui surgissent dans la littérature de temps à autre, auxquelles on ajoute la puissance de calcul qui permet de favoriser les attaques même envers les algorithmes les plus complexes. En d'autres termes, la cryptographie offre toujours une sécurité relative qui cédera certainement tôt ou tard.

Une technique de chiffrement est dite *fiable* (ou d'une force relative) si elle utilise un algorithme complexe et dispose d'un large espace de clés. Un *cryptanalyste* ne connaissant pas la

bonne clé, doit théoriquement tenter tout l'espace considéré en utilisant une recherché *exhaustive*. Selon la norme de sécurité 1997[5] et, tenant compte de la puissance de calcul disponible, une technique est dite robuste si elle utilise des clés de plus de 80 bits, c'est-à-dire, un espace de 2^{80} clés différentes ou utilisant un algorithme du type *masque jetable*[6]. Malheureusement, la mise en application d'une telle technique n'est pas toujours évidente vu sa complexité en ressources qui retarde les échanges de flux et cause des anomalies pratiques en cas d'usage de dispositifs temps réel. En plus, les attaques fructueuses peuvent profiter de certaines erreurs de conception, réalisation ou d'installation de systèmes de chiffrements tels les attaques par *mot probable* ou *paradoxe des anniversaires* ou par une analyse mathématique des chiffrements telles les attaques par *rencontre au milieu*.

L'efficacité d'une attaque dépend de la quantité et la qualité des données dont dispose le cryptanalyste. Elle est basée sur la connaissance des caractéristiques des cryptosystèmes, telles l'algorithme de chiffrement, la langue de texte en clair, des fragments de textes clairs ou chiffrés. En général, le cryptanalyste est en mesure de récupérer d'avantages de textes clairs à l'aide de certaines clés spécifiques, sinon, la recherche exhaustive semble la solution la plus sollicitée. Elle garantit un résultat sûr mais utilisée uniquement comme dernier recours lorsqu'il parait impossible de profiter de la faiblesse d'un cryptosystème. Pour cette raison, plusieurs méthodes d'attaques ont été proposées, la plupart d'entre elles est basée sur de textes clairs connus ou de textes clairs choisis. Certaines d'entre elles sont aléatoires : attaque par mot probable[7], ou par dictionnaire[8]. D'autres sont dédiées : *analyse fréquentielle* ou *indice de coïncidence*[9] et certaines autres sont à caractère analytique destinées aux chiffrements modernes tel que *l'attaque de Boomerang* (Wagner, 1999), basée sur l'analyse de la perturbation de sortie résultant affecté par une différence connue dans l'information d'entrée, l'*attaque de l'homme-du-milieu* qui vise à analyser le rapprochement affine relatif à un texte clair et son correspondant chiffré .

Ainsi et, comme tout problème combinatoire relevant de l'industrie, la communication, la biologie ou purement théorique, la cryptanalyse en faisait partie et, considéré comme NP-Difficile, où la détermination d'une solution exacte prend plus de temps qu'il soit raisonnable d'en chercher une telle solution. En d'autres termes, aucun algorithme polynomial[10] n'est en mesure de fournir une solution exacte pour tous les problèmes de ce type. Donc, impossible pratiquement de faire recours aux méthodes exactes.

Concrètement, l'attaque exhaustive d'un chiffrement utilisant une clé de taille n, nécessite la génération de n ! clés distinctes obtenues par les différentes permutations des n caractères ou bits de la clé. Chaque clé produite désigne une solution du problème et associée à une fonction

5. http://csrc.nist.gov/publications/

6. Le masque jetable ou Chiffre de Vernam est un cas particulier du chiffre de Vigénère ou la dé est aussi longue que le texte, ce qui réduit à néant toute tentative de cryptanalyse.

7. Basée sur l'idée de l'existence d'un mot, phrase ou expression donnée dans le texte chiffré. Cette supposition (si elle est correcte) permet de connaître une partie du texte en chiffré et en clair et permet cependant de deviner la dé. Le mot probable peut être une date, un nom ou une signature.

8. Consiste à essayer de déchiffrer un mot par usage d'une liste de mots disponibles. La technique est utilisée pour casser les mots de passe. La liste des mots (appelée dictionnaire) est choisie à base des habitudes et du comportement de la personne attaquée et pourrait contenir un prénom, un animal favori, une couleur, un numéro d'assurance, etc.

9. Méthodes explicitées en chapitre 3.

10 Un algorithme en mesure de résoudre un problème de la classe NP en un temps polynomial est équivalent à prouver l'égalité P=NP. Inversement, ce type d'algorithme est inexistant.

objectif représentant sa valeur et doit satisfaire une série de contraintes relatives à la nature du problème, notamment la qualité du texte déchiffré. En supposant que le traitement d'une solution nécessite une moyenne 100n unités de temps machine, le temps nécessaire à accomplir une telle attaque est illustré par le tableau (1.1) relatif à certains calculateurs réputés puissants.

n	n!	Blue Gene (367 teraflops[11])	Tianhe-2 (33,86 petaflops[12])	DuoCore Ordinaire 2.5 Ghz/ 10Gflops
5	120	$32,7x10^{-12}$ secondes	$35,4x10^{-14}$	$12x10^{-7}$
26	$\approx 1204x10^{26}$	$10,9x10^{12}$ secondes $\approx$ 3170 siècles	$11,9x10^{11}$ secondes $\approx$ 300 siècles	$4x10^{18}$ secondes $\approx 1.3\ x10^{9}$ siècles
64	$\approx 1.27x10^{120}$	$>10^{8}$ siècles	……	……

Tab 1.1. Estimation du temps de recherche exhaustive dans un espace de solutions.

Il parait cependant quasi impossible d'accomplir de telles tâches en pratique ; par conséquent, la résolution de tels problèmes doit être confiée aux techniques approchés ou metaheuristiques qui permettent, en un temps raisonnable de fournir des solutions de qualité acceptable ou 'bonnes' solutions sans garantie d'optimalité (Clark, 1998).

Ces techniques exploitent généralement des processus aléatoires dans l'exploration de l'espace de recherche au lieu d'une recherche exhaustive afin de faire face à l'explosion combinatoire engendrée par l'utilisation des méthodes exactes. En plus de cette base stochastique, les metaheuristiques sont le plus souvent itératives où le même processus de recherche est réitéré lors de l'exploration. Leur efficacité provient justement de leur capacité à éviter les minima locaux et par conséquent, réduire l'espace de recherche au profit d'une dégradation de la fonction objectif. Ce phénomène a été décrit par (Bauer, 1997) comme suit:

"*The exhaustion attack, although by itself alone rather insignificant, is in combination with other, likewise automatic attacks, the fundamental method of intelligent cryptanalysis*."

En cryptanalyse, cette idée s'explique par le fait qu'une modification élémentaire dans le corps de la clé (telle la permutation de deux caractères ou deux bits) enclenche un changement proportionnel au sein du texte déchiffré. En d'autres termes, il existe une relation entre la clé et le texte chiffré ou, dans la terminologie de (Shannon, 1949), un niveau bas de confusion. Partant de ce constat simple, il se ensuit qu'une clé partiellement correcte, donne lieu à un déchiffrement correct en partie également (Clark J. A., 2003) et par conséquent, une approximation de la clé est suffisante pour produire un texte lisible pouvant être amélioré manuellement.

Bien que certaines techniques heuristiques telles le Branch-&-Bound ou BFS soient en mesure de résoudre des problèmes NP-Complets, elles sont rarement utilisées pour les instances de large taille et, où la valeur optimale de la fonction objective ne peut être prédéfinie. Ainsi, les

11. 1 teraflop=10^{12} opérations par seconde

12. 1 petaflop=10^{15} opérations par seconde

metaheuristiques font usage des caractéristiques du problème considéré pour produire de bonnes solutions et, où la notion : bonne est d'un sens relatif par rapport à d'autres solutions trouvées et indépendant de la solution réellement optimale. A titre d'exemple, si une metaheuristiques est utilisée pour rechercher une clé pour un chiffrement donné et si une clé trouvée permet de fournir un déchiffrement clair, elle sera considérée comme bonne. Si le déchiffrement est lisible en partie, la solution pourrait quand même être considérée comme bonne s'il y'en aura pas d'amélioration durant tout le processus de recherche.

Les metaheuristiques ont connu un développement significatif durant les trois dernières décennies (Sharvani, Cauvery, & Rangaswamy, 2009). Ceci est dû principalement à leur l'efficacité quant au traitement les problèmes combinatoires. Diverses recherches (Felix & Manoj, 2007) (Dadhich, Gupta, & Yadav, 2014) ont montré que les algorithmes basés sur le comportement des essaims, une catégorie des metaheuristiques inspirées de la nature, ont un potentiel consistant pour le traitement de larges instances et peuvent être adaptés pour produire des solutions approximatives pour une grande variété de problèmes d'optimisation y compris la cryptanalyse (Yean Li, Samsudin, & Belaton, 2005). Elles utilisent un système intelligent qui offre une indépendance de mouvement des agents qui tend à remplacer la préprogrammation et le contrôle centralisé.

1.2 Metaheuristiques en cryptanalyse

Depuis leur création, les metaheuristiques étaient toujours considérées comme un outil d'attaque efficace en cryptanalyse des chiffrements classiques. Les premières recherches dans ce domaine remontent aux années 80 (Peleg & Rosenfeld, 1979); ont proposé l'algorithme de relaxation pour attaque aux chiffrements par substitution ; les résultats étaient limités vu le champ réduit des expérimentations. Le même algorithme a été repris par (Hunter & McKenzie, 1983) (Carroll & Robbins, 1987) avec une expérimentation plus poussée. En 1993, (Ramesh, Athithan, & Thiruvengadam, 1993) ont introduit la notion de dictionnaire de référence qui consiste à garder intact tous les mots générés lors d'un déchiffrement ayant la même syntaxe que ceux du dictionnaire. Au cours de la même année, (Spillman, Janssen, Nelson, & Kepner, 1993), (Matthews, 1993) et (Forsyth & Savafi, 1993) ont proposé respectivement les algorithmes génétiques pour la cryptanalyse des chiffrements par substitution et transposition et le recuit simulé pour l'attaque aux chiffrements par substitution simple ; les résultats étaient relativement positifs mais limités par la puissance du calcul disponible à l'époque. Ces attaques ont été reprises sous une autre forme par (Jackobsen, 1995) en 1995. En 1994, (King, 1994) présenta pour la première fois une attaque aux chiffrements par substitution polyalphabétique en utilisant l'algorithme de relaxation ; (Clark & Dawson, 1997) ont utilisé, pour l'attaque du chiffrement par substitution, une approche des algorithmes génétiques basée sur une stratégie permettant le traitement de plusieurs parties du problème simultanément.. Durant les années 2000, (Clark, 2001) (Clark & Jacob, 2001) (Clark, Russell, & Stepney, 2003) (Clark, Russell, & Stepney, 2003) (Dimovski & Gligoroski, 2003) ont démontré que les metaheuristique peuvent être utilisées comme outil de conception des protocoles et fonctions booléennes utilisées en cryptanalyse. Les résultats de la plupart des travaux ainsi cités étaient limités vu la faible puissance de calcul disponible à l'époque et très variée due à la divergence liée à l'environnement d'expérimentation.

Durant la dernière décennie, la recherche s'est focalisée vers les metaheuristiques à population inspirée de la nature ; diverses recherches ont été élaborées : (Russel, Clark, & Stepny, 2003) ont

utilisé une approche basée sur les colonies de fourmis pour l'attaque du chiffrement par transposition ; Ce même chiffrement a été objet d'attaques diverses par des metaheuristiques; (Uddin & Youssef, 2006a) (Uddin & Youssef, 2006b) ont utilisé, pour l'attaque du chiffrement par substitution des approches basées sur les essaims particulaires et les colonies de fourmis ; (Verma, Dave, & Joshi, 2007) (Picek & Golub, 2011) (Luthra & Pal, 2011) (Omran, Al-Khalid, & Al-Saady, 2010) (Toemeh & Arumugam, 2007) ont montré que les méthodes basées sur l'évolution biologique telle la sélection génétique, peuvent être utilisés comme outils de résolution d'une variété de problèmes complexes dont la cryptographie ; (Omran, Al-Khalid, & Al-Saady, 2011) ont repris le travail de Clark et Dawson en élargissant la base de tests ; (Mekhaznia, Menai, & Zidani, 2012) ont proposé une version modifiée de l'algorithme des essaims particulaires pour l'attaque de certains chiffrements classiques ;(Dureha & Kaur, 2013) ont proposé une variante générique des algorithmes génétiques pour l'attaque de divers chiffrements classiques; les expérimentations ont été satisfaisants quant à la révélation des positions de caractères et également à la distinction entre les chiffrements par substitution et transposition.; (Heydari, Shabgani, & Heydari, 2013), (Sadiq, Ali, & Kareem, 2014) et (Mishra & Kaur, 2015) ont proposé des approches d'attaque aux chiffrements par transposition utilisant respectivement les *algorithmes génétiques*, *Cuckoo Search* , *Hill Climbing* et le *recuit simulé*; Les résultats expérimentaux montrent que les approches considérées peuvent être efficaces pour la détermination de la clé optimale.

1.3 Insuffisance des travaux existants

Les diverses approches ainsi mentionnées ont été en général, appliquées avec succès à la cryptanalyse de la plupart des chiffrements classiques. Cependant, il est nécessaire de remarquer que la majorité des implémentations réalisées ont été testées à base d'expérimentation restreinte ou spécifique en utilisant notamment des chiffrements modèles qui ne reflètent pas le standard des données du monde réel. Ceci est justifié par le fait que la plupart des bases de tests ont été élaborées ou sélectionnées par les auteurs selon leur angle de vue relative au problème considéré et ce, en l'absence de benchmarks dédiés à ce genre de problèmes ; (Clark J. A., 1998), dans sa thèse, utilisa pour ses tests des passages extraits de « *2000 Leagues Under the Sea* » de « Jules Verne » seulement, alors que (Delman, 2004) a utilisé certains passages de la littérature anglaise, entre autres : « *Genesis* » et « *The Holy Bible* » de «King James, version » et « *The Time Machine* » de « Herbert George Wells». (Banks, 2008) utilisa certains extraits de la littérature anglaise, entre autres « «*Hamlet* », « *Alice's Adventures in Wonderland* », « *The Prince* » sans aucune justification quand à ces choix. La majorité des autres recherches ne mentionnent quasiment jamais leur sources de tests ni la taille des extraits choisis. Donc, les résultats d'expérimentations obtenus à l'issu des données mentionnées sont considérées comme spécifiques à leurs environnements respectifs et par conséquent, difficiles à reproduire ou à vérifier.

D'un autre côté, la majorité des travaux publiés se sont appuyés sur l'analyse de fréquence d'apparition de caractères au sein des textes déchiffrés comme le seul moyen d'évaluation de la validité d'un décryptage. Ce principe a permis d'élaborer une variété fonctions objectif basées sur les *n-grams* et dotées d'un certain nombre de paramètres dont il parait difficile de les évaluer en l'absence de modèle standard de référence. Cependant, les implémentations proposent diverses valeurs de paramètres basées en majorité sur l'intuition. Le détail de ces travaux est illustré au §5.2.5 et §6.2.2

Concernant les algorithmes de chiffrement, les travaux décrits dans des publications antérieures s'adressent en grande partie à des attaques aux algorithmes de substitution ou de transposition simples. Peu d'attention a été donnée aux chiffrements classiques complexes tels la substitution polyalphabétique. Les tests mentionnés au §7.5 montrent que ce chiffrement est le plus difficile à casser dans sa catégorie, donc relativement délaissé par la communauté scientifique. Les techniques d'attaques étaient partagées entre les méthodes de recherche locale telle la Recherche Tabou et Recuit simulé et les metaheuristiques à population où les algorithmes génétiques et essaims particulaires étaient les plus sollicités tandis que les colonies de fourmis étaient moins préconisées dans ce domaine.

Les résultats relatifs à l'usage de l'une ou l'autre de ces techniques mentionnent dans la majorité des cas, que la dite technique peut être appliquée à ce type de problème ou qu'elle est prometteuse ou potentiellement apte à fournir des résultats acceptables quant à la cryptanalyse d'un tel chiffrement sans toutefois fournir des arguments valides surtout que de telles conclusions ne peuvent être prononcées qu'à base d'une synthèse de comparaison non pas entre diverses techniques hétérogènes, mais également entre environnements de tests différents.

1.4 Objectifs et Contribution

Les divers travaux énumérés ont permis d'aborder la plupart des algorithmes de chiffrements classiques et une partie importante des chiffrements symétriques par des attaques utilisant une grande catégorie des techniques de recherche existantes. Cependant, la contribution ainsi présentée n'est pas de procéder à d'attaques similaires mais plutôt de répondre à la question : quelle est la meilleure technique d'attaque et pour quel chiffrement et dans quel environnement ?

Parmi les éléments de réponse proposées pour ce fait, celle d'aborder des attaques basées sur la nature du chiffrement lui-même, c'est-à-dire et, au lieu de procéder à une attaque à l'aide de clés générées d'une manière exhaustive, choisir des clés qui répondent à certaines spécificités du chiffrement et par conséquent, orienter la recherche vers un objectif ou les alternatives de choix se rétrécissent au cours des itérations. L'objectif du travail consiste donc à une analyse fréquentielle des chiffrements sous leur aspect linguistique, ce qui permet de savoir si le résultat obtenu au cours du traitement concorde en partie avec une langue donnée. Cet acte permet en outre, de fournir une idée sur la qualité de la clé utilisée pour le déchiffrement : s'il fallait la remplacer entièrement par une autre ou modifier certains de ses caractères seulement et, pour ce dernier cas : quels sont les caractères qu'il faut modifier ou échanger au sein de la clé et à quels emplacements.

De même, le but du travail se focalise en partie dans l'amélioration de certains outils de cryptanalyse et plus précisément, une recherche de clés par analyse des statistiques de fréquences de caractères d'une manière permettant l'optimisation de ressources. L'outil applicatif utilisé est un ensemble d'algorithmes basés sur certaines classes de techniques de recherche locale, de branchement et certaines metaheuristiques à population. Le résultat renvoi une synthèse quant à la classification de ces algorithmes selon leur efficacité à résoudre de tels types de problèmes.

Le choix ainsi proposé est basé sur le fait que toutes les techniques de chiffrements concernant des textes littéraires s'accordent sur le principe de l'analyse de fréquences comme étant le seul moyen de prouver la qualité d'un résultat de déchiffrement en l'absence d'autres outils de synthèse ou de comparaison. Ces outils, s'ils existent, peuvent être la clé de chiffrement ou le

texte correct qu'on essaye d'atteindre comme objectif au cours d'une attaque. Or ces deux éléments sont disponibles seulement en travail de laboratoire. Dans le monde réel, on dispose uniquement d'un texte chiffré (un message chiffré récupéré d'une transmission sur un réseau) qu'on essaye de le dévoiler ; donc, l'algorithme de chiffrement doit être reconnu à base du texte chiffré lui-même et pas d'autre chose. De même, l'implémentation doit comporter un outil de reconnaissance de la langue littéraire du texte tel l'indice de *coïncidence* ou la méthode de *Kazizki* et un autre outil de détermination de la longueur de de la clé du chiffrement telle la méthode de *Friedman* qui s'avèrent bénéfiques pour les attaques aux chiffrements à clés variables qui s'avèrent exceptionnellement difficiles à briser à base de la seule analyse de fréquence.

Concernant l'environnement de tests qui comporte un nombre assez consistant de paramètres ; données relatives à la nature du chiffrement : type et taille de la clé utilisée ; données relatives à la technique d'attaque et les données générales (taille du texte utilisé, tables statistiques de fréquences, données relatives à la fonction objectif,..etc.) ainsi que les paramètres d'exécution notamment le temps alloué à chaque essai ; des tests préliminaires vont être opérés afin de fixer les meilleurs paramètres environnementaux tel la taille de la population d'une metaheuristique, la taille des textes utilisés, les meilleurs coefficients de poids de la fonction objectif, .., etc. Ce travail de synthèse, absent dans la plupart des travaux similaires en littérature, permet en grande partie d'améliorer les conditions d'expérimentations et favoriser la convergence de la solution. L'ensemble de ces étapes est intégré au sein d'un outil de recherche automatisé afin d'éviter une intervention humaine au cours du processus de traitement permettant ainsi un gain de ressources.

Enfin, l'objectifs essentiel du travail est de produire une synthèse de comparaison de performances entre diverses techniques de recherche issues des principales catégories de metaheuristiques, à savoir les techniques de recherche locale, de branchement et les méthodes à population en utilisant une grande variété de données de tests représentant la plupart des structures linguistiques échangées sur les réseaux (textes littéraires, dialogues, extraits de médias écrite, messages publics, ..etc.). La performance concerne surtout le paramétrage relatif à chaque chiffrement utilisé dans le but de réduire la consommation de ressources ainsi que le nombre d'itérations maximal représentant un seuil de confiance au-delà duquel aucune performance de la fonction objectif n'est attendue.

1.5 Structure de thèse

Le premier chapitre de ce travail permet d'avoir une vue globale sur le contenu de la thèse : après une brève introduction de la cryptologie, il tente de présenter la problématique liée à la cryptanalyse des chiffrements, son utilité et conséquences.

Le deuxième chapitre présente une description générale des concepts cryptographiques et leurs caractéristiques ainsi que les algorithmes et structures de chiffrements classiques et symétriques les plus utilisées, leurs avantages et faiblesses. Ce chapitre permet d'avoir une vision globale sur la cryptographie et sur son utilité.

Le troisième chapitre, expose certaines techniques célèbres de cryptanalyse, les plus marquantes, avec un développement détaillé des caractéristiques des langages littéraires, analyse de fréquences de caractères et corpus ainsi que des exemples reflétant des cas réels. Le but de ce chapitre est de comprendre le fonctionnement des attaques aux chiffrements ainsi que l'importance de l'analyse linguistique liée au domaine de la cryptanalyse.

Le quatrième chapitre est consacré à l'étude des metaheuristiques ; Nous sommes intéressés dans un premier temps à l'état de l'art des méthodes d'optimisation en général : description, classement, caractéristiques. Dans un deuxième temps, nous avons développé certains algorithmes de recherche locale, de branchement et quelques metaheuristiques à population. Ce chapitre permet de donner une vision globale sur les metaheuristiques et leur importance quant à la résolution de problèmes combinatoires.

Les chapitres 5 et 6 sont consacrés à une mise en œuvre expérimentale. Il illustre le cadre de modélisation des techniques metaheuristiques pour la cryptanalyse. Le but de ce chapitre des d'offrir un modèle efficace de présentation des données liées aux techniques de recherche et une manière quant à leur adaptation pour la construction de structures dédiées à l'environnement de la cryptanalyse.

Le dernier chapitre est dédié à l'expérimentation proprement dite. Il illustre l'environnement et les paramètres des tests, les données et les critères de leurs choix. Il présente également les divers résultats d'attaques obtenus avec des synthèses de comparaison, analyses et commentaires. Ce chapitre permet d'évaluer les expérimentations réalisées et ce, en regard des travaux similaires en littérature.

La conclusion apporte une synthèse du travail élaboré et quelques idées en perspective.

2. Cryptographie

2.1 Introduction

La *cryptographie*[1] est la science qui englobe les diverses techniques de protection de données en les transformant sous une forme inintelligible. Elle désigne les diverses techniques ou algorithmes de chiffrement et de déchiffrement. Une technique de chiffrement typique à un texte lisible (connu sous le nom de *texte clair*) et d'une certaine donnée secrète (connue sous le nom de *clé*) comme entrée, produit une forme brouillée (ou chiffrée) du texte d'origine (connu sous le nom de *texte chiffré* ou *chiffrement*). Le *déchiffrement* consiste en l'opération inverse produisant le texte clair d'origine en utilisant le texte chiffré et la clé. En général, la plupart des techniques cryptographiques utilisent la même la clé pour le chiffrement et le déchiffrement Un système de chiffrement ou *cryptosystème* englobe les divers données (textes clairs et chiffrés, algorithme et clé) permettant d'accomplir le chiffrement et le déchiffrement. A base de ce principe et, connaissant un texte chiffré, seules les personnes possédant la bonne clé auront accès au texte clair correspondant et toute tentative de déchiffrement avec une clé autre que celle ayant servi au chiffrement est dite *attaque* ou *cryptanalyse* (Menezes, Oorschot, & Scott, 1996).

Depuis toujours et partout où l'homme a semé une civilisation, la cryptographie était présente pour assurer la protection des données secrètes sous ses diverses états : stockage, traitement ou échanges entre les individus. Les premières traces de la cryptographie remontent à la civilisation ancienne (Vaudenay, 2006). Au fil du temps et, avec le développement civique de l'humanité, les techniques cryptographiques sont devenues de plus en plus complexes. La demande à ces

1. Du grec κρυπτος: caché et γραφειν (Γραπειν): écrire

techniques s'est intensifiée non seulement pour protéger les secrets propres mais, également pour divulguer ou détourner les secrets d'autrui. Ce principe est similaire à l'activité d'un sociologue qui essaye de comprendre et prévoir le comportement des autres avant de les inférer ou d'un médecin qui doit diagnostiquer son patient avant de lui administrer un produit. Actuellement, la cryptographie ne s'intéresse pas à l'homme lui-même, mais à ses idées encodées sur une machine. Une technique de chiffrement ne peut être considérée comme fiable qu'après avoir subi diverses tentatives d'attaques non réussis. Cependant, la cryptanalyse ne peut être vue comme une action ou démarche destructive mais plutôt, un moyen permettant d'améliorer la sécurité des chiffrements par la mise en surface de ses défauts et points faibles aidant ainsi la recherche à concevoir de systèmes de cryptographiques plus robustes.

Les techniques de chiffrement ont connu un développement sérieux au cours du vingtième siècle, la deuxième guerre mondiale en était le facteur le plus influent dans ce processus. Elle a enclenché une avancée énorme en matière de développement scientifique où l'accès à la connaissance était à la portée de tout le monde et les supports d'écriture étaient également disponibles. La prolifération des systèmes de communication a cassé la monopolisation de la cryptographie du domaine militaire et a encouragée le développement d'avantages de techniques plus efficaces et répondant mieux à la sécurité des échanges. Néanmoins, les meilleurs techniques demeurent toujours secrètes et réservées aux milieux fermés de la recherche ou de l'industrie au sein des pays développés.

Aujourd'hui, la cryptographie a regagné la majorité des domaines économiques, notamment le secteur bancaire, le commerce, l'administration, la communication. La progression a été favorisée par l'invention d'applications et équipements nouveaux, mobiles en majorité, suscités par l'utilisation des réseaux publics et la manipulation de quantités importantes d'informations numériques. Cette évolution pose donc le problème de la confidentialité et l'intégrité des données; leur protection n'est plus garantie surtout avec l'évolution de la puissance de calcul qui, utilisée par les attaquants pour briser une majorité de chiffrements reconnus robustes.

2.2 Historique

Les premiers techniques cryptographiques fussent leur apparition au 6è siècle Av. J.-C. avec la méthode d'*AtBash*[2], celle de *Cesar*[3] au 1[er] siècle av. J.-C. Il a été objet d'améliorations qui avaient donné naissance de nouvelles techniques plus consistantes notamment le *Rot13*[4] et la méthode de *Vigénere*[5]. Puis, l'évolution et, pour de raisons inconnues s'est figée une longue période de l'histoire.

Quinze siècles plus tard, les égyptiens ont de nouveau contribué dans le domaine de la cryptographie en publiant des écrits relatifs aux techniques de substitution et de la transposition.

2. Technique basique de chiffrement par substitution employée par les hébreux 500 ans av. J.-C. Elle consistait à un échange de chaque caractère à partir du début du texte par son opposé à partir de la fin. Malgré basique et faible, la technique était acceptable à son temps vu le manque d'outils mathématiques adaptés à son cryptanalyse.

3. Jules César (100-44) av. J.-C., homme politique et écrivain romain. Inventa le chiffre de César qui consiste à un décalage avec rotation des caractères de texte d'un certain nombre de pas, généralement 3.

4. Cas particulier du chiffre de César où le décalage est de 13 positions. Son but est d'utiliser le même principe du chiffrement et du déchiffrement car l'alphabet comportait 26 caractères.

5. Technique de chiffrement polyalphabétique développée plus bas dans ce chapitre.

En 1467, Batista Alberti[6] était l'auteur de la substitution polyalphabétique puis, en 1518 était le tour de Jean Tritheme qui publia un livre sur la cryptographie où il annonçait le premier système de sténographie[7]. En 1553, Giovan Batista Belaso utilisa, et pour la première fois la notion de mot de passe[8]. Dix ans plus tard, Giambattista Della Porta inventa les premières règles quant à la substitution polygrammique[9]. En 1793, Thomas Jefferson qui inventa son cylindre de 26 roues[10]. Ce mécanisme a été encore réinventé par Bazaries sous le nom de M-94[11].

Au 19è siècle, c'était l'ère de la cryptographie analytique ou cryptanalyse : Le chiffre de Vigenere a été cassé par Charles Babbage et Friedrick Kasiski quant à Etienne Bazarie, il a cassé le Grand chiffre de Luis XIV[12]. En 1883, Auguste Kerckhoffs publia ses règles quant aux systèmes cryptographiques modernes qui demeurent une référence jusqu'à nos jours[13].

Depuis là, l'évolution de la cryptographie s'est lancée dans tous les sens favorisée par le développement économique : le commerce en ligne, la téléphonie, la gestion des transmissions et ce, avec des centaines d'applications standards ou dédiées, néanmoins les grands axes des systèmes cryptographiques demeures stables, à savoir la cryptographie classique, et moderne avec ses deux branches : symétrique et asymétrique.

Actuellement, les chiffrements sont largement utilisées dans les institutions bancaires, le transport, la téléphonie mobile et au sein des secteurs militaires et étatiques. Les années à venir vont, peut-être tourner vers la cryptographie quantique qui possède des caractéristiques inviolables. En attendant, l'usage des outils existants paraît satisfaisant jusqu'à lors.

2.3 Caractéristiques

2.3.1 Principes

L'émetteur d'un message veut bien qu'aucune tierce personne non autorisée ne doit prendre connaissance du contenu de ses transmissions ni de les modifier ou les échanger à d'autres. Il veille également à ce que ses données transmises doivent toucher le destinataire et que ce dernier ait connaissance de ces données d'une manière intégrale sans altération et qu'il ne doit pas nier de

6. Leon Battista Alberti(1404-1472), écrivain, philosophe et architecte de la renaissance. Développa en 1460 son chiffre polyalphabétique basé sur plusieurs alphabets désordonnés évitant ainsi les attaques par analyse de fréquence.

7 En latin, Johannes Trithemius(1462-1516), abbé allemand proposa un traité de sténographie. Il fut accusé de magie et courut le risque d'être brulé avec son livre

8. Giovan Battista Bellaso (1505-), cryptographe italien mort à une date inconnue.

9. Giambattista della Porta (ou Giovanni Battista Della Porta) (1535-1615), physicien, opticien, philosophe, cryptologue et alchimiste italien. écrit De Futivis Literarum Notis, traitant les chiffres, la cryptanalyse et les caractéristiques linguistiques qui favorisent le déchiffrement. Inventa la première substitution bigrammique qui consiste à remplacer deux lettres sont représentées par un seul symbole. Il inventa également le premier chiffre polyalphabétique et fut le premier à classer les deux principes cryptographiques: la substitution et la transposition.

10 Thomas Jefferson (1743-1826), homme d'état, agronome et architecte American. Inventa son cylindre composé de 26 roues identiques montées sur un même axe comprenant les lettres aggravées en désordre. Le message clair est reconstitué par les cylindres. Le message chiffré est récupéré sur une autre ligne.

11. Étienne Bazeries (1846-1931), cryptanalystes. Inventa le cylindre de Bazeries basé sur le cylindre de Jefferson. L'invention a été utilisée par l'armée américaine sous le nom de M-94 et améliorée en M-138-A.

12. Enigme de « «l'homme au masque de fer ».

13. Le principe d'Auguste Kerckhoffs est détaillé au chapitre 3

les avoir reçues. D'un autre côté, le destinataire doit pouvoir reconnaitre la qualité du message reçu et si ce dernier est en bon état ou altéré et associer son émission à une personne ou entité bien connue sans toutefois que cette dernière nie l'avoir transmis.

A base de ces critères, un système cryptographique doit être en mesure d'assurer les divers services suivants :

- La *confidentialité* : englobe l'ensemble des dispositions permettant de restreindre l'accès aux données aux seules personnes autorisées. Ce service concerne les informations échangées ou enregistrées dans des dispositifs publics ; il est assuré par le contrôle d'accès ou le chiffrement contre la divulgation ;
- L'*intégrité* : service permettant de protéger une donnée contre toute tentative d'altération ou de modification lors de sa transmission, traitement ou stockage. Similaire à la confidentialité, ce service est assuré par un code d'accès avec interdiction d'opérations d'écriture ou de modification. Au sens large, l'intégrité désigne la manière de pouvoir confirmer qu'une information est conforme à son état d'origine et qu'il n'y a pas eu d'erreurs liées à la transmission, la falsification et qu'elle est valide à l'utilisation.
- L'*authenticité :* service permettant d'associer à chaque information émise sur le réseau l'identité de son propriétaire. L'authentification permet au destinataire de la dite information d'être assuré que son émetteur est bien la personne qu'il prétend être. Ce service peut être assuré par une information relative à l'identité de l'émetteur, reconnue par le destinataire et émise par le propriétaire de l'envoi et que lui seul est capable de la produire. En général, l'authenticité peut être accomplie par un support physique : adresse IP, carte à puce ou empreinte comme elle peut être soft telle la signature numérique, un code d'accès, etc.
- La *non-répudiation* : désigne l'assurance d'une transmission entre deux personnes ; que l'émetteur a bien émis son message à destination du destinataire et que dernier a bien reçu le dit message et que son correspondant est bien à l'origine de cet envoi. La non-répudiation peut être confirmée par un chiffrement avec clés communes ou par certificat numérique.
- *Non duplication* : désigne la manière de protection de données contre la copie ou duplication. Ce principe permet de responsabiliser l'auteur du document transmis et préserver ses droits. Ce service est associé parfois au *marquage électronique*, un procédé de modification invisible d'un produit, image par exemple afin d'identifier les copies pirates.
- *Anonymat* : service permettant de garder l'identité de l'émetteur transparente au public afin de préserver ses secrets. Ce service parait utile lors de transmission de données sensibles ou lors des transactions bancaires, élections, etc.
- *Le time stamping* : service permettant de dater la création, la modification ou l'émission d'une information. Un tel procédé permet de gérer la validité des données transmises et engager les responsabilités des entités.

2.3.2 Terminologie

Au cours de ce travail, les termes relatifs à la cryptologie les plus utilisés sont définis ainsi :

- *Alphabet A*: ensemble fini de symboles (lettres alphabétiques ou bits 0 et 1) utilisés pour écrire les messages.
- *Texte clair p* **:** (ou PlainText), chaîne de caractères ou bits composée de lettres de l'alphabet *A* et dont on veut en général les conserver en secret. On note *P* l'ensemble de tous les textes clairs.
- *Texte crypté c*: (ou CipherText), chaîne de caractères composée de lettres de l'alphabet *A*, correspondant à un message clair, et dont la diffusion à des entités non autorisées ne doit pas dévoiler pas d'information sur le texte clair. On note *C*, l'ensemble de tous les textes cryptés.
- *Chiffrement* (ou Cryptage) : Processus de transformation d'un texte clair en une forme incompréhensible pour toute personne non autorisée. Il désigne parfois le texte chiffré lui-même.
- *Déchiffrement* (ou Décryptage): transformation inverse qui permet de retrouver le texte clair à partir du texte chiffré équivalent et ce, à l'aide de la clé du déchiffrement.
- *Clé k:* donnée permettant de construire les fonctions de chiffrement et de déchiffrement. Sans connaissance de cette clé, le déchiffrement sera impossible. On note K, l'ensemble de toutes les clés. Certains procédés de chiffrement utilisent une même clé pour le chiffrement et le déchiffrement ; d'autres en utilisent une seule. La clé doit être gardée secrète en général.
- *Cryptanalyse*: procédé de reconstitution du texte clair à partir d'un texte chiffré sans au préalable, connaitre la clé ou l'algorithme de chiffrement.
- *Cryptogramme*: Texte chiffré, résultat d'une opération de chiffrement. Par abus de langage, un texte chiffré peut être désigné par chiffrement ou chiffré.
- *Cryptologue*: Auteur ou concepteur de cryptogrammes.
- *Signature s*: chaîne de caractères associée à un texte donné (et aussi possiblement à une entité) et le caractérisant. On note *S* l'ensemble des signatures.
- *Transformation Tk*: fonction qui associe à un texte clair ou chiffré, une autre donnée (message clair, crypté ou signature). En *général*, ce sont des fonctions qui dépendent de clés.
- *Protocole*: description de l'ensemble des données nécessaires pour mettre en place le mécanisme de cryptographie : ensemble de *textes* clairs et chiffrés, des clés possibles, des transformations, etc.
- *Vérification de création* ou d'existence d'une information par une personne autre que le créateur de cette information, sans forcément dévoiler cette information. L'accusé de réception permet de vérifier qu'une information est bien arrivée à son destinataire.

2.4 Systèmes cryptographiques

2.4.1 Définition

Un système cryptographique (appelé cryptosystème) est l'ensemble des techniques (algorithmes ou méthodes de chiffrement), documents (textes clairs et chiffrés, clefs) et outils matériels (par

exemple un processeur particulier, un encodeur ou un calculateur spécifique type *enigma*[14] ou *CipherUDD*[15] qui permettent le chiffrement de données (messages, fichiers, etc.) d'une façon particulière comme montré par la figure 2.1[16] :

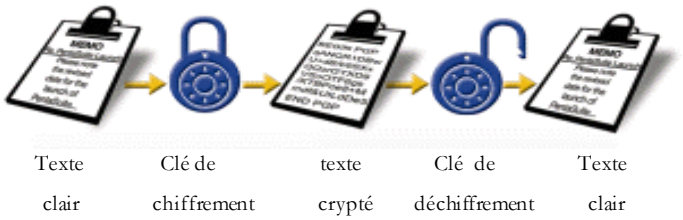

Fig. 2.1 : Principe de chiffrement

Selon (Schneier, 1994), « *Un cryptosystème est un terme utilisé en cryptographie pour désigner un ensemble composé d'algorithmes cryptographiques et de tous les textes en clairs, textes chiffrés et clés possibles*»

Un cryptosystème est composé des éléments suivants (Stinson, 2006):

- Espace P des messages clairs p sur un alphabet A,
- Espace de C des textes chiffrés c sur un alphabet B (égal à A ou différent),
- Espace K clés k,
- Un ensemble T de transformations de chiffrement. Chaque transformation est indexée par une clé selon la relation :

$$T_k : p \in P \rightarrow c \in C \tag{2.1}$$

- Un ensemble T^{-1} de transformations de déchiffrement selon la relation :

$$T^{-1}_k : c \in C \rightarrow p \in P \tag{2.2}$$

Bien entendu, la condition $T_k(T^{-1}_k(p))=p$ doit être vérifiée. Le cryptosystème est noté par (P,C,k,T,T^{-1}).

L'efficacité de cette procédure repose sur la clé k qui doit être gardée secrète. La transformation T, appelée algorithme de chiffrement est rendue publique tel mentionné dans le principe de *kerckhoffs*.

14. Machine de chiffrement électromécanique à rotors inventée par *Arthur Scherbius* et *Richard Ritter* en 1918. Elle a été utilisée en Allemagne durant la 2ème guerre mondiale aux fins militaires.

15.Système matériel de chiffrement conçu en format de lecteur de mémoires Flash ou disque dur. Le produit est conçu par *Addonics* ; une firme américaine de commercialisation de produits de sauvegarde.

16. www.pentaware.com (consulté le 22/7/2014).

2.4.2 Objectifs

Un système cryptographique fiable doit être en mesure de transférer des données entre deux ou plusieurs entités en respectant les règles suivantes :

- Veuille à la sécurité des données qui ne doivent pas être déchiffrées que par le destinataire et avoir l'habilité de se rendre compte de toute tentative d'attaque (réussie ou non) au niveau des canaux d'échanges.
- Pouvoir identifier les différents acteurs d'un échange de données notamment les serveurs ou points d'accès ayant participé à l'acheminement de messages et retracer le chemin de passage de paquets en cas d'anomalie.
- Etre en mesure de chiffrer et déchiffrer rapidement un volume de données acceptable, voire en temps réel pour certaines applications industrielles.
- Résister aux diverses attaques connues et dans la mesure du possible, avoir une sécurité prouvée.

En général, aucun chiffrement n'est en mesure de réunir simultanément toutes les spécificités indiquées, néanmoins, le choix est basé sur un compromis temps-sécurité adapté à chaque circonstance.

2.4.3 Evolution

Le développement des cryptosystèmes était jusqu'aux années 70 basé sur des méthodes symétriques où la même clé est utilisée pour le chiffrement et le déchiffrement. Ce mécanisme pose le problème de la transmission des clés entre les utilisateurs lors de l'échange d'informations via les réseaux publics.

En 1976, ce problème a été résolu par l'introduction du concept de la cryptographie à clé publique où seule la clé de déchiffrement est secrète, ce qui évite le transfert de clés (Diffie & Hellman, 1976). Le premier cryptosystème RSA à clé publique fût introduit en 1978 (Rivest, Shamir, & Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, 1978) et, malgré sa résistance à la cryptanalyse, il n'a pas marqué la fin de la cryptographie symétrique car celle-ci détient toujours une mise en œuvre plus rapide et un niveau de sécurité acceptable. L'évolution des cryptosystèmes à clé publique a permis de créer de nouveaux concepts, notamment la signature numérique et l'authenticité de l'émetteur.

Actuellement, les cryptosystèmes sont classés en deux catégories (fig.2.2) : classique (comprenant entre autre, la substitution et la transposition) et moderne relatif aux techniques complexes symétriques et asymétriques. Les techniques asymétriques emploient des clés publiques basées sur les nombres premiers, tandis que les techniques symétriques utilisent des clés privées du même principe de la cryptographie classique, néanmoins, elles sont plus longues et nombreuses avec des algorithmes de chiffrement plus complexes.

Cependant, la cryptographie classique demeure un référentiel de développement des techniques à clés secrètes. Elle est également utilisable dans certains environnements à risque modéré et ce, grâce à la simplicité de ses algorithmes et sa rapidité de chiffrement.

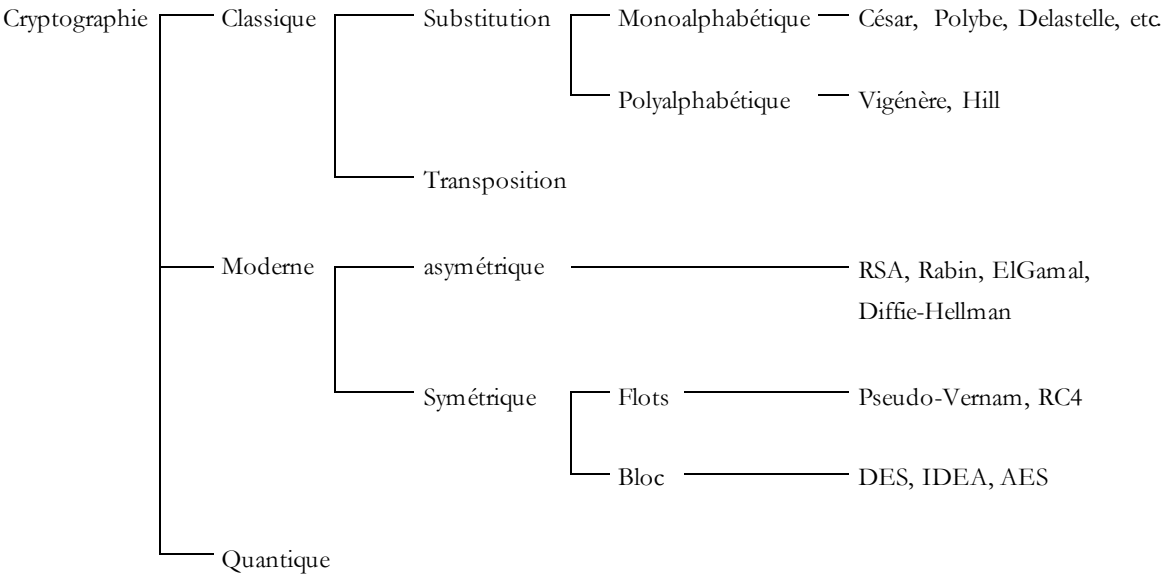

Fig. 2.2 : Classification des systèmes cryptographiques

2.4.4 Sécurité

a. Principe de kerckhoffs

L'idée énoncé par (Kerckhoffs, 1883) (Kahn, 1996) était destinée à résoudre les difficultés liées aux transmissions par Télégraphe[17]. Eventuellement cet outil permet l'échange de données secrètes entre diverses entités indépendantes. Il parait donc difficile de construire un algorithme de chiffrement pour chaque personne et même si c'est le cas, de tels algorithmes ne peuvent être gardés secrets car il *existe toujours une tierce personne qui connaît le mécanisme d'un chiffrement*. S'il n'est pas divulgué par erreur ou volé, il peut toujours être connu par une personne malveillante, le concepteur de l'algorithme ou son vendeur. Cependant et, pour des raisons de standardisation, il paraît plus logique de construire un système de chiffrement unique qui peut être configuré par chaque utilisateur par ses propres paramètres afin que la partie secrète de l'algorithme soit minimale. L'idée de Kerchoffs n'est donc pas un algorithme ou une technique d'attaque mais plutôt un outil de mesure de la difficulté calculatoire d'un cryptogramme quant à l'usage massif de données. ; Son principe consistait à ce que la sécurité d'un système doit reposer sur un paramètre connu plus tard sous le nom de *clé secrète*.

b. Chiffre de Vernam

Appelé également *Masque jetable* et classé dans la catégorie des chiffrements par flots (voir chap.5), le chiffrement de *Vernam* (Vernam, 1926) consiste à chiffrer un texte p de avec une clé k aléatoire de même taille que le texte à l'aide d'un XOR de la manière suivante

[17]. L'idée de kerckhoffs stipule que « *La sécurité d'un cryptosystèmes ne repose pas sur le secret du cryptosystème mais seulement sur la clef qui est un paramètre facile à changer, de taille réduite (actuellement de 64 à 2048 bits suivant le type de code et la sécurité demandée et donc assez facile à transmettre secrètement.* »

$$c_i = p_i \oplus k_i \quad (2.3)$$

avec i, le i-ème élément du texte considéré qui peut être un caractère ou bit.

La mise en œuvre d'un tel chiffrement est délicate. La fabrication et le stockage de clés aléatoires était toujours un problème dans la mesure où chaque clé est utilisable une seule fois. La manière de transfert sécurisé de clés entre les utilisateurs est également difficile. De même, la taille de la clé nécessite une connaissance préalable de la longueur du texte à chiffrer ce qui requiert l'élaboration de clé au même temps que le texte.

En revanche, si la manière du choix de la clé k est respectée, qu'elle est aléatoire au sens du mot et utilisable une seule fois, ce chiffrement offre une sécurité totale au sens de du théorème de Shannon (Shannon, 1949) relatif à la théorie de l'information, c'est-à-dire que la clé ou le texte chiffré c n'offrent aucune indication sur la nature du texte clair.

2.4.5 Stratégies d'attaques aux chiffrements

Un système cryptographie est, selon l'idée de Kerckhoffs supposé public à l'exception de la clé. L'attaquant est censé connaître l'algorithme du chiffrement, les détails de la fonction de combinaison utilisée et avoir en même temps accès à divers données claires et leurs correspondantes chiffrés et le secret du chiffrement réside bien entendu dans les conditions initiales constituant la clé. En pratique, chaque attaque est unique et spécifique à son environnement ; ce qui ressemble à un acte plutôt artisanal qu'industriel dans le sens où son résultat reste imprévisible même si elle est exécutée deux fois dans les mêmes conditions.

a. Attaques aux données

Le but d'une attaque est de trouver la bonne clé de chiffrement, ce qui est équivalent parfois à trouver le chiffré lui-même. En général, elle sera interrompues dès que le sens du chiffré peut être assimilé de la même manière qu'un jeu de mots croisés. A base de cette situation, l'attaquant peut agir de différentes manières :

- S'attaquer à un texte chiffré seul : c'est le cas le plus fréquent et également le plus dur, car le texte chiffré ne fournit pas assez d'informations susceptibles de retrouver la clé ou le texte clair. En général, pour la cryptographie classique, on fait recours à certaines autres caractéristiques liées au langage littéraire utilisé dans les chiffrements telle l'analyse de fréquence. Cette même idée peut être utilisée pour l'attaque de chiffrements symétriques ayant un espace réduit de clés.

  Les chiffrements symétriques ayant un large espace de clés (64 et plus) sont résistants à ce genre d'attaques. L'attaquant peut toujours faire recours à d'autres techniques standards notamment l'attaque par dictionnaire, paradoxe des anniversaires et l'homme-du-milieu.

- L'attaquant dispose de certains textes claires et d'autres chiffrés ou pouvoir choisir soi-même des échantillons de textes clairs et procéder à leur chiffrement. Cette alternative est plus efficace et peut casser tous les chiffrements classiques ou presque et également un grand nombre de chiffrements symétriques. Les techniques utilisées dans ce contexte se rapportent à la cryptanalyse linéaire et différentielle.

b. Attaques à la conception

Eventuellement, la première cible d'une attaque concerne les chiffrements et les clés. Dans d'autres cas, elle peut viser les algorithmes de chiffrements en espérant de découvrir certaines failles. En général, un algorithme est dédié à son environnement d'utilisation : type et taille de données, gestion de la mémoire, protocole d'échange, etc. Diverses institutions et, dans le but de minimiser les charges liées à la conception dédiée, ne se privent pas de récupérer des sources libres (fonctions de chiffrement, hachage, signatures) et de les adapter à leurs environnement sans vérification approfondie, ce qui a pour effet, peut-être, de générer des failles qui, exploitées par les concepteurs des logiciels pour surpasser la barrière de sécurité.

En pratique, un algorithme fiable doit être en mesure de gérer les diverses situations critiques qui peuvent se présenter lors de l'installation ou de l'exécution, notamment :

- Tests réussis sur différents types de données particulièrement les grandes ou petites valeurs, données spécifiques, etc ; et que l'algorithme ne plante pas lors d'une exécution limite et qu'il permet de gérer convenablement la mémoire conjointement à d'autres applications et sur divers types de machines.
- Paramètres de protection fiables quant aux codes d'accès, de lecture de fichiers.
- Pouvoir s'adapter à l'environnement de son installation : matériel, interface, protocoles, antivirus, etc.
- Gestion correcte d'erreurs quant à d'arrêt prématuré dû à une erreur d'exécution ou acte volontaire de l'utilisateur avec protection de code, fermeture de fichiers et suppression des données temporaires.

Du coté exploitation, Les failles relatives aux données sont également variées, la plus célèbre est celle liée aux S-boxes de l'algorithme DES (voir chapitre 4) – si on ose l'appeler ainsi- car et jusqu'à lors personne ne sait si elles renferment des trappes susceptible d'être exploitées par son concepteur pour contourner la clé de chiffrement. Les éléments à considérer dans ce contexte se résument par les points suivants :

- Gestion de partage de clés : Le transfert de clés constitue un majeur problème vulnérable aux attaques. Ce problème s'accentue si le nombre d'utilisateurs est important. A noter que pour deux utilisateurs, il faut échanger une clé ; pour trois utilisateurs, il faut trois clés. Pour un nombre supérieur, il faut autant de clés qu'il y a de paires non ordonnées d'utilisateurs, à savoir $n(n-1)/2$.

  Le transfert d'une clé par un canal ordinaire peut à la limite, passer inaperçue mais si ce nombre augmente, il faut utiliser des canaux privés. En cas général, l'usage des protocoles d'échange parait la meilleure solution. Les protocoles sollicités dans ce contexte relèvent de la cryptographie asymétrique, le plus célèbre est dû à (Diffie & Hellman, 1976).

- Espace de clés : La force d'un système de chiffrement est étroitement liée à la taille de l'espace de clés. Cet espace doit contenir suffisamment d'éléments afin de résister à l'attaque exhaustive. Il est admis qu'une cardinalité de 2^{80} est suffisante pour prévenir la plupart des attaques, soit 10^{24} clés environ. Ce principe met en cause la totalité des chiffrements classique et une grande partie des chiffrements symétriques ayant des clés de 64 bits.

- Gestion des nombres pseudo-aléatoires : prévoir des fonctions permettant de générer des nombres aléatoires corrects et de pouvoir détecter leur réutilisation.

2.5 Cryptographie classique

2.5.1 Introduction

Les techniques de chiffrement classiques concernent la période précédant l'âge des calculateurs. Elles étaient, jusqu'aux années 70 utilisées comme le seul outil disponible de chiffrement. Basées sur la notion de caractère, pour cacher un texte, il suffit de le désordonner, de substituer ses caractères par d'autres ou simplement de les transposer selon un rangement particulier. Les meilleurs systèmes classiques itèrent ces opérations plusieurs fois. Malgré leur simplicité, ces techniques offrent une confidentialité acceptable.

A la naissance des calculateurs, Ces techniques ont été placées à *usage restreint* ; le caractère est désormais remplacé par le bit, et les divers algorithmes de chiffrements ont migré vers la cryptographie symétrique.

Malgré leur simplicité, les opérations de chiffrement classique demeurent la pierre angulaire de la cryptographie moderne ; elles sont utilisées dans la conception des fonctions non linéaires Sboxes et réseaux de substitution-permutations utilisées dans la plupart des chiffrements symétriques.

2.5.2 Schémas cryptographiques classiques

a. Cryptographie par substitution monoalphabétique

- Description

C'est le plus simple chiffrement imaginable ; il consiste à remplacer chaque caractère p_i du texte clair P de taille n appartenant à l'alphabet A par un autre caractère de l'alphabet de substitution B qui en général, n'est autre qu'un réarrangement de l'ordre lexicographique de l'alphabet A et ce, à l'aide d'une transformation T_k définie de la manière suivante :

$$c=T_k(p)=k(p_1)k(p_2)\ldots k(p_n) \text{ avec } p=p_1p_2\ldots p_n \tag{2.4}$$

avec c, le texte chiffré; p_i, le i-ème caractère du texte clair. k étant la clé de chiffrement. Connaissant celle-ci, on déchiffre le message crypté avec la transformation inverse définie par la relation suivante :

$$T^{-1}_k(c)= k^{-1}(c_1)k^{-1}(c_2)\ldots k^{-1}(c_i)=p \tag{2.5}$$

Malgré sa simplicité, la méthode offre une impression de sécurité relative, car il est quasi impossible de casser un texte en utilisant la force brute étant donné qu'il existe 26! ($\approx 4.10^{26}$)

possibilités distinctes de choix de clés si on utilise un alphabet de 26 caractères (Menezes, Oorschot, & Scott, 1996).

Son inconvénient réside dans la clé qui parait assez longue et difficile de s'en retenir ou de la transmettre sans risque.

- Exemple

Soit le passage suivant :

> « UN PETIT ROSEAU M'A SUFFI POUR FAIRE FREMIR L'HERBE HAUTE ET TOUT LE PRE ET LES DOUX SAULES ET LE RUISSEAU QUI CHANTE AUSSI. »[18]

En utilisant une clé *k* définie par :

Texte clair	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
Texte codé	W	X	E	H	Y	Z	T	K	C	P	J	I	U	A	D	G	L	Q	M	N	R	S	F	V	B	O

Le texte chiffré sera donc :

> «RA GYNCN QDMYWR U'W MRZZCN GDRQ ZWCQY ZQYUCQ I'KYQXY KWRNY YN NDRN IY GQY YN IYM HDRV MWRIYM LRC EKWANYAN WRMMC. »

- Algorithme

Le texte *p* ainsi que la clé *k* peuvent être représentés par des tableaux *k[2,26]* et *p[n]*. Le texte résultat sera également représenté par un tableau de taille n. L'algorithme illustrant le chiffrement est donné par le pseudo-code suivant :

Fonction 2.1. Subst_mono(p, c)

```
Pour i ← 1 à n faire
    j ← 1
    TantQue p[i] ≠ k[1,j]
        j ← j+1
    FintantQue
    c[i] ← m[k[2,j]]
FinPour
```

18. Extrait de Régnier, Henri de, « Odelette 1 », Les jeux rustiques et divins, Paris, Crès, 1925 [1897].

- Variantes

Parmi les chiffrements connus utilisant la substitution simple, on note : le *carré de Polybe*[19], le *chiffre de Delastelle*[20], le *chiffre des Templiers*[21], le *chiffre de Pigpen*[22], le chiffrement de *César*, le chiffrement *Affine*, etc.

- Faiblesse

Le chiffrement par substitution est facile à casser par l'analyse de *fréquence de caractères*, cependant il demeure un composant essentiel dans la construction des *sboxes* utilisées dans les chiffrements symétriques modernes.

b. Cryptographie par substitution polyalphabétique

- Définition :

L'idée revient à Léon Battista[23], la substitution polyalphabétique est une technique basée sur plusieurs permutations utilisées successivement l'une après l'autre d'une manière cyclique, c'est à dire, utiliser un chiffrement par substitution monoalphabétique en changeant à chaque fois le caractère de remplacement de la manière suivante :

$$c = T_{k1,k2,\ldots,kt}(m)=k_1(p_1)k_2(p_2)\ldots k_{i[t]}(p_i)\ldots k_{n[t]}(p_n) \quad (2.6)$$

avec c, le texte chiffré; p_i, le i-ème caractère du texte clair; k_1, k_2, …, k_t des transformations au nombre de t, et i[t] désigne i mod t si (i mod t ≠ 0) et t si (i mod t ≠ 0).

L'ensemble des permutations s'obtient à l'aide d'une clé de taille déterminée inférieure à la taille de l'alphabet qui indique à chaque permutation le nombre de décalages à réaliser.

L'analyse fréquentielle dans ce type de chiffrement est inefficace.

- Exemple :

En utilisant une transformation T_{k1k2k3}='123' de longueur 3 qui indique que le premier caractère du texte clair sera décalé d'une position, le caractère suivant de deux positions et le troisième de 3, le quatrième d'une position, etc. Le texte « CRYPTOGRAPHIE » sera chiffré en « DTBQVRHTDQJLF »

---

19 Ce chiffre a été utilisé par les *nihilistes* russes. Inventé par *Polybe* vers 150, il consiste en une table 5x5 contant la dé (sans répétition de caractères) suivie du reste des caractères de l'alphabet avec omission du J ou du W. Le message crypté consiste en un ensemble de chiffres (de 11 à 55) relatifs aux coordonnées de chaque caractère du texte dair.

20 Inventé par *Felix-Marie Delastelle*, Ce chiffre combine une substitution des lettres du texte dair par des symboles, puis affectés par une transposition, puis retour aux lettres par une deuxième substitution.

21 Relatif à l'icône de l'ordre du Temple, ce chiffre consiste à remplacer les lettres par des symboles du type de la croix de Malte.

22 Appartenant à *Corneille Agripa*, ce chiffre consiste en une substitution des lettres par des symboles du type traits et points.

23. Leon Battista Alberti (1404-1472). Artiste en architecture et renaissance italien, compositeur et poète à qui on a attribué le concept du *cadrant chiffrant*.

- Algorithme:

Pour une clé *k* de taille *t* et un texte clair de taille *n*, la variante la plus simple du chiffrement polyalphabétique est illustré par le pseudo-code suivant :

Fonction 2.2. Subst_Polyalpha(p, c, k)

```
Pour i ← 1 à n faire
    c[i] ← p[k[i%t]]%26
FinPour
```

- Variantes

Les chiffrements utilisant la substitution polyalphabétique sont : le chiffre de *Vigéneire*, le Chiffre de *Hill*[24], la machine *Enigma*[25], Chiffre de *Bellaso*[26], Chiffre de *Porta*[27], etc.

- Force et faiblesse

Pour décrypter ce chiffrement, il est nécessaire de connaître la longueur de la clé. Ceci peut être déduit par une recherche de répétitions de caractères dans le message chiffré en utilisant certains utilitaires, notamment *l'indice de coïncidence*, la méthode de *kazizki* ou le test de *Friedman*.

c. Cryptographie homophonique

- Définition :

Technique fût apparue au XVI^{ème} siècle[28]. Elle consiste à remplacer chaque caractère de l'alphabet *A* par un symbole de l'alphabet *B* choisi parmi un sous-ensemble de symboles appelés *homophones*, c'est-à-dire une permutation non bijective *T* entre *A* et *B* illustrée par la relation suivante :

$$c = T_k(p) = k_{\{p1\}}(p_1)\, k_{\{p2\}}(p_2) \ldots k_{\{pi\}}(p_i) \ldots k_{\{pn\}}(p_n) \qquad (2.7)$$

avec *c,* le texte chiffré; p_i le i-ème caractère du texte clair; $k_{\{i\}}(i)$, la permutation du caractère *i* par un symbole appartenant au sous ensemble $\{i\}$ de symboles relatifs à ce caractère, et *n* désigne la taille du texte à chiffrer .

24. Chiffre polyalphabétique publié en 1929 par S.Hill Utilise le principe de chiffrement par blocs du même type que le chiffrement de Playfair.

25. Machine de chiffrement polyalphabétique utilisée en Allemagne Nazi durant la 2eme guerre mondiale.

26. Chiffrement polyalphabétique basé sur des mots dés et adapté à l'alphabet italien. Conçu par Giovanni Battista au IVX siède.

27.Système polyalphabétique inventé par Giovanni Batista della Porta en 1563 qui consiste de changer d'alphabet à chaque lettre.

28. Le premier exemple connu est un texte manuscrit attribué à Michele Steno (1331-1413), doge de Venise, datant de 1411, où il se contente d'associer plusieurs symboles aux voyelles. Source : http://www.apprendre-en-ligne.net/crypto/homophone/ (Consulté le 01/11/2014).

Chaque caractère de *A* possède son propre sous-ensemble d'homophones dont la taille est proportionnelle à sa fréquence[29]. L'ensemble *B* doit contenir en général, 100 symboles distincts, car on a besoin de 100 possibilités (somme des fréquences de caractères de *A*).

Car	Fréq.	Homophone	Car	Fréq.	Homophone	Car	Fréq.	Homophone
E	13	Π B Q ≠ J ∈ Ω ≤ π S É ⊇ θ	U	4	⁄ ? ∩ ⌂	V	1	%
A	9	τ → ₴ ∪ ρ < γ ⇒ ∏	C	4	▲ ◊ ♀ ₡	Z	1	∅
I	8	α ≤ μ ¬ ε √ φ ‼	M	3	♠ ώ ♥	W	1	₳
R	8	ϙ ⇐ Ϛ ≡ B Δ ∩ Ж	D	2	☼ ₪	J	1	₡
S	7	Ж ↓ З ∂ ∅ К ┌	P	2	♪ ‰	X	1	Э
N	7	⊗ Ш ⊕ Λ ⊆ Э Ю	H	2	♫ ₹	Q	1	←
T	7	Я ↑ ϗ £ β # ∑	G	2	Ł 2	F	1	@
O	6	∧ И ∨ ϑ ⇔ ћ	B	2	₳	K	1	∞
L	5	⊃ Ç ♦ æ	Y	1	Ⱶ			

Tab.2.1. Chiffrement homomorphique

L'affectation de ces symboles aux caractères du texte clair peut être réalisée d'une manière aléatoire. Il n'y a pas de restrictions quant au choix des symboles de l'ensemble *B* pourvu qu'ils appartiennent à la table ASCII supporté par le format de données utilisé dans l'implémentation de l'utilisateur et également de son correspondant; On utilise par exemple, des nombres de deux positions 00 à 99 mais la taille du message crypté s'allonge au double.

On peut également utiliser des caractères ASCII divers comme illustré par le tableau 3.1[30]. Une autre alternative moins compliquée consiste à l'emploi l'alphabet ordinaire (telle la substitution monoalphabétique) auquel on intègre le même alphabet en format minuscule, les chiffres et quelques caractères de ponctuation.

- Exemple :

Ainsi, avec le tableau de substitution précédent, le texte :

«LARESSEMBLANCECACHEBIENDESDIFFERENCES»

peut-être chiffré de la manière suivante :

«Ç∏≡J3KΠώ₳ə< ⊗♀Q◊ ∪◊₹≠ ₳≤∈⊗₪ √É ☼μ@@≠Ж⊃◊B↓»

ou également par :

29. Ce choix est connu sous le terme : *renversement de fréquence*. Utilisé pour empêcher la cryptanalyse par analyse fréquentielle.

30. Marc Lesson, « Atelier mathématique : Chiffre homophonique ». Source : http://www.lifl.fr/~wegrzyno/enigma/enigma/documents/homophonique.pdf (Consulté le 08/05/2014).

$$« \supset \rho B \Omega \varnothing \partial \Pi \phi \acute{A} Ç \gamma \oplus q B \Diamond \Rightarrow \Diamond \♫ \leq \acute{A} \alpha J \Pi \ ☼ \varphi S \ ☼ \neg @ @ Q \cap \theta ♀ \Pi \varnothing »$$

et ce, après suppression des caractères de séparation et ponctuation.

- Algorithme :

Soient *k[n,s]*, une matrice de correspondance avec *n* la taille de l'alphabet *A* et *s* la probabilité maximale d'apparition de tout caractère de *A*. Pour chaque ligne de *k*, on insère les caractères de remplacement au nombre de *s* ; si ce nombre est inférieur à *s*, on répète certains caractères afin d'atteindre ce nombre ; et *p[n]*, un texte clair de taille *n*, la fonction de substitution de tout caractère p_i par un symbole de la matrice de correspondance est donnée par le pseudo-code suivant :

Function 2.3. Subst_homo(m, c, k)

Pour *i* ← *1 à n faire*
    c[i] ← *k[p[i],rand(s)]*
FinPour

où *c[n]* étant le texte chiffré correspondant à *p[n]*.

- Force et faiblesse :

Ce chiffrement résiste à l'analyse fréquentielle de caractères, il est plus sûr qu'un chiffrement par substitution monoalphabétique, néanmoins son cadre de sureté est simplement relatif car ses défauts apparaissent sous deux formes :

- Il n'est pas fiable à l'analyse par *bigrammes* car, certains caractères de faible fréquence notamment le Q,U apparaissent souvent ensembles en littérature, et la répétition de certains autres caractères tels W, X et Y est rare. En se penchant sur de longs chiffrements, on pourra identifier les caractères de remplacement ainsi les propriétés de certains syllabes, notamment : le Q ou U qui sont d'ordinaire suivis par une voyelle.
- L'ensemble B n'est pas toujours facile à construire. Il ne doit pas contenir de redondances et doit être facile à implémenter. De même, il est considéré comme une clé de chiffrement qui est bien entendu assez longue à retenir ou à communiquer.

Du côté avantages, le chiffrement homographique, malgré classique, a été repris par la recherche actuelle pour subvenir certains besoins relatifs aux architectures modernes en matière de sécurité de données, notamment le *Cloud Computing*[31] et l'*Internet of Things*[32]. La nouvelle approche consiste à procéder au traitement direct sur les données cryptées sans recours au préalable à l'opération de déchiffrement. L'avantage est bien entendu est le gain de ressources de stockage et temps de calcul (Gentry, Fully Homomorphic Encryption Using Ideal Lattices, 2009).

31. Accès en service libre aux ressources partagées distantes.

32. Echange d'informations et données provenant du monde physique

d. Cryptographie par transposition

- Définition :

Consiste à réordonner les caractères d'un texte de taille n (subdivisés en b blocs de d caractères) sans toutefois les substituer à d'autres[33] afin de construire des *anagrammes*[34] et ce, conformément à une transformation définie comme suit :

$$c = T_k(p) = k_1(p_{1,1})\, k_2(p_{1,2}) \ldots k_d(p_{1,d})\, k_1(p_{2,1}) \ldots k_d(p_{b,d}) \text{ avec } k(p_{i,j}) \in p \quad (2.8)$$

où k_i désigne le *i-eme* caractère de la clé k qui représente la nouvelle position du *i-eme* caractère de chaque bloc p dans le texte chiffré c, et d, la taille du bloc.

La transformation k qui référence également la clé, peut être opérée de différentes manières et ce, après le rangement du texte à chiffrer dans une matrice rectangulaire. La permutation en spirale (Tab. 2.2) par exemple consiste à relever les caractères de la matrice par un ordre spiral qui commence par un caractère du bord et se termine au centre de la matrice construisant ainsi le texte chiffré.

Texte clair p					Texte chiffré c			
C	R	Y	P		N	O	S	N
T	O	G	R		R	E	A	T
A	P	H	I		C	R	Y	P
E	P	A	R		R	I	R	A
R	T	R	A	→	O	I	T	I
N	S	P	O		S	T	P	P
S	I	T	I		O	G	H	A
O	N	←			R	P		

Tab.2.2. Transposition en spirale

La ligne de parcours peut être en biais (en diagonale), en dents de scie (méthode de *Rail Fence*), carré magique (type table de *karnaugh*) ou plutôt selon la marche de cavalier sur échiquier.

Afin de limiter la taille de la clé, le texte à chiffrer est divisé en blocs fixes de taille d. On utilise généralement une matrice où la longueur de la ligne correspond à la taille la clé. La permutation est appliquée de la même manière pour chaque bloc, on parle ainsi de *transposition*.

33. En un mot, les méthodes de transposition sont une salade des lettres du texte dair .par Etienne Bazeries(1846-1931).

34.La manière d'échange de l'ordre de caractères d'un mot ou une phrase afin d'obtenir une nouvelle structure ayant un sens. Tel l'exemple : « Albert Einstein » : « rien n'est établi ».

- Exemple

En utilisant une clé « JOUR ». Le chiffrement consiste à réarranger les colonnes de la matrice du texte clair selon l'ordre alphabétique des caractères de la clé, à savoir 1243, l'exemple précédent sera chiffré comme suit :

1	2	3	4	← Ordre →	1	2	4	3
C	R	Y	P		C	R	P	Y
T	O	G	R		T	O	R	G
A	P	H	I		A	P	I	H
E	P	A	R	→	E	P	R	A
R	T	R	A		R	T	A	R
N	S	P	O		N	S	O	P
S	I	T	I		S	I	I	T
O	N	←			O	N		

Tab.2.3. Transposition par colonnes

Ainsi, le texte chiffré sera « CRPYTORGAPIHEPRARTARNSOPSIITON »

Les cases vides du tableau peuvent être ignorées ou remplacées par un caractère rarement utilisé « x » par exemple.

- Algorithme

Dans ce chiffrement, la clé $k=\{i_1,i_2,..,i_d\}$ avec i_j indice du caractère j au sein de chaque bloc de taille d (d>1). L'ordre des indices est défini selon la permutation k. La permutation est opérée conformément aux règles générales suivantes :

- Découper le texte à crypter en blocs identiques de taille *d*. (l'insérer, au préalable dans une matrice de largeur d).
- Si le dernier bloc est de taille inférieure à *d*, le compléter au terme de taille *d* par un caractère quelconque.
- Pour chaque bloc, permuter les caractères selon l'ordre défini dans k.

L'algorithme correspondant est illustré par le pseudo-code suivant :

Function 2.4. Transpo(p, c)

```
Pour i ← 1 à n-d pas d faire
    Pour j ← 1 à d faire
        c[i+k[j]] ← p[i+j]
    FinPour
FinPour
```

- Variantes :

Le chiffrement par transposition peut être réalisé de différentes manières. Autre que la permutation de l'ordre des caractères, il existe d'autres variantes, notamment la double *transposition*[35], le chiffre de *Ubchi*[36], le chiffre *digrafide*[37], la transposition *abrupte*[38], etc.

- Force et faiblesse :

Le chiffrement par transposition conserve les caractères du texte clair, ce qui rend l'analyse statistique des fréquences de caractères inefficace. Par conséquent, la technique offre un niveau acceptable de sécurité sous réserve que la permutation s'opère selon des règles rigoureuses.

La faiblesse de la méthode, tel tout système de chiffrement, réside dans la règle de permutation (clé) qui, en principe connue par l'émetteur et le destinataire et doit être souvent échangée au préalable à travers un réseau public.

2.6 Cryptographie symétrique

2.6.1 Introduction :

La cryptographie moderne fût ses premiers pas aux années 70 grâce à *Horst Feistel* qui a mis au point son premier algorithme de chiffrement par bloc : *Lucifer*[39], développé plus tard pour générer plusieurs variantes notamment le célèbre *DES*[40]. Conjointement à cette invention, *Diffie* et *Hellman* ont mis au point leur concept de chiffrement à clé publique (Diffie & Hellman, 1976) qui porta leurs noms respectifs. Durant la même période, apparut le célèbre l'algorithme *RSA*[41] (Rivest, Shamir, & Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, 1978) imaginé et mis en pratique par ses auteurs. Ces algorithmes ont mis fin aux problèmes liés à la distribution et transport de clés.

Durant les années 90 et, avec le développement des réseaux et protocoles modernes, de nouveaux systèmes de chiffrements ont vu le jour : *l'IDEA*[42] en 1990 ; Le *PGP*[43], une application

35. Chiffrement utilisant deux transpositions successives avec une ou deux dés différentes, sur les lignes ou les colonnes ou les deux.

36. C'est une succession de deux transpositions en utilisant la même dé de chiffrement ; c'est-à-dire, l'insertion du texte dans une matrice de k colonnes. La récupération du texte chiffré se fait par colonne en suivant l'ordre des permutations de la dé k.

37. C'est un chiffre tomogrammique utilisant deux alphabets désordonnés. Les caractères du texte clair sont groupés par bigrammes et transformés en nombres de trois chiffres puis permutés selon un ordre défini par la dé. Le résultat est retransformé en caractères et donne les bigrammes chiffrés.

38. Dans ce modèle, on ignore certaines cases de la matrice lors de l'insertion du texte clair. Après permutation, les caractères seront insérés dans la matrice cible dans un ordre normal. Cette technique a pour effet de créer un décalage de entre les cases des deux matrices et brouille les tentatives de la cryptanalyse.

39. Algorithme utilisant le chiffrement par blocs.

40. *Data Encryption standard*, algorithme de chiffrement symétrique utilisant des blocs de 64 bits. Son premier standard a été publié en 1977 par FIPS.

41. Algorithme de chiffrement à dé publique basée sur les nombres premiers breveté par MIT en 1983.

42. *International Date Encryption Standard*, algorithme de chiffrement symétrique successeur de DES. Conçu par Xuejia Lai et James Massey en 1991 et breveté en suisse par mediacrypt.

43. *Pretty Good Privacy*, application de chiffrement à dé publique offrant divers services, notamment l'authentification et la compression.

de chiffrement de *Zimmermann* destinée au grand public. En 1991, le *Blowfish*[44] et en 1993, l'implantation de la première fonction de hachage initié par *Rivest* en 1990 (Rivest R. L., 1992). Dans cette même période, les premiers résultats de la cryptographie quantique de *Benett* et *Brassart* relatifs à la distribution de clés ont vu le jour (Benett, F, Brassard, Salvail, & Smolin, 1992).

Ainsi, la cryptographie moderne s'est répartie en deux grandes classes : symétrique et asymétrique. En cryptographie asymétrique le chiffrement et le déchiffrement se fait par de clés différentes. Une telle clé est construite à partir de deux sous-clés : publique, connue par tous les utilisateurs et une privée propre à chacun d'eux. En cryptographie symétrique, tous les utilisateurs possèdent la même clé qui servira pour le chiffrement et le déchiffrement. L'usage d'un tel algorithme nécessite l'échange préalable de la clé entre les utilisateurs à travers un canal sécurisé ou spécifique.

Dans ce qui suit, notre étude se limite à la cryptographie symétrique où l'analyse et l'attaque peut être réalisée par les heuristiques.

2.6.2 Définition

Le *chiffrement symétrique*, appelé également chiffrement à *clé secrète* ou *clé privée* est principalement lié aux services de la confidentialité. Il est extrêmement répandu à cause de son efficacité, sa simplicité d'implémentation et ses performances comparables au chiffrement asymétrique et fonctions de hachage.

La structure d'un chiffrement symétrique est basée sur des algorithmes itératifs comportant un certain nombre d'étages. Au niveau de chaque étage, intervient la même transformation paramétrée par une clé distincte de celle des autres étages.

Similaire à la cryptographie classique, le chiffrement symétrique génère un bloc de texte chiffré c à partir d'un bloc de texte clair p de même taille à l'aide d'une transformation T en utilisant une clé k. (relation 2.9). La restauration du texte p à partir du chiffré c se fait par application de la même transformation au chiffrement et de la même clé k, d'où le nom symétrique prend son sens.

$$c=T_k(p)=k(p_1)k(p_2)\ldots k(p_n) \text{ avec } p=p_1p_2\ldots p_n \tag{2.9}$$

Le majeur inconvénient des systèmes de chiffrement symétrique est la manière d'échange et de la distribution de clés. Ceci est accentué lors de l'échange de données avec plusieurs utilisateurs.

2.6.3 Modes de chiffrement

a. Chiffrement par blocs :

Dans ce chiffrement, un texte de taille p est préalablement découpé en blocs de longueur fixe de b bits et traités séparément. Si le dernier bloc ayant une taille inférieure à b, il sera ajusté à cette taille par une règle de formatage convenue. Ce chiffrement est fréquemment utilisé et

44. Algorithme de chiffrement symétrique libre utilisant des blocs de 64 bits conçu par Bruce Schneider en 1993.

permet une meilleure sécurité. Les algorithmes concernés sont également plus connus (*DES*, *AES*, *Skipjack*., *IDEA*, *RC6*, *Blowfish*,..). La plupart de ces algorithmes emploie des blocs de 64 bits et des clés de taille égale, mais ce n'est pas toujours le cas ; cela dépend du niveau de sécurité envisagé : un chiffrement de moins de 40 bits est déclaré faible au sens du principe de *Kerchkoffs* ; un chiffrement de 56 bits (tel le standard DES) est qualifié de acceptable, c'est-à-dire vulnérable aux attaques moyennant de grandes ressources ; donc, ne peut être envisagé pour le chiffrement de données sensibles. Les chiffrements utilisant de blocs de 128 bits (tel le standard AES) sont reconnus temporairement robustes.

b. Chiffrement en chaine

Appelé également chiffrement par *flot* ou par *flux*, ce chiffrement considère qu'un texte est une suite finie de bits où chaque bit est traité séparément des autres. Le chiffre de *Vernam* (§2.4.4) en est un exemple. En pratique, on utilise une clé de taille fixe permettant de générer une suite chiffrante qui, additionnée au texte clair, retourne une suite aléatoire permettant de masquer toute possibilité de reconnaissance du texte clair.

Pour un texte p de longueur n, le chiffrement c de p en utilisant une clé k est défini par la relation (2.10)

$$c=T_k(p)=k_1(p_1)k_2(p_2)\ldots k_n(p_n) \text{ avec } p=p_1p_2\ldots p_n \tag{2.10}$$

Le plus célèbre exemple d'un tel chiffrement est l'algorithme RC4 (Rivest R. , 1992), utilisé dans la conception de plusieurs protocoles notamment le SSL, Netscape, WEP, WPA, etc.

2.6.4 Structure

Les chiffrements symétriques sont itératifs, c'est-à-dire basés sur des algorithmes qui consistent à itérer une certaine transformation, appelée *fonction de tour*, exécutée un certain nombre de fois sur le même bloc de données en utilisant de différentes clés. La clé principale utilisée est appelée *clé maître*. Pour chaque tour, on utilise une nouvelle clé, appelée *sous-clé* extraite de la clé maître à l'aide d'un algorithme de *cadencement de clés* de façon à ce que tous les sous clés générées soient différentes. La relation 4.6 illustre le résultat chiffré c_i obtenu à partir d'un bloc clair p_i en utilisant la fonction du tour associée généralement à une addition logique et d'un nombre r de sous clés.

$$c_i^k = p_i^{k1} \oplus p_i^{k2} \oplus .. \oplus p_i^{kr} \tag{2.11}$$

Les chiffrements symétriques sont basés sur le concept de *confusion* et *diffusion* définis par Shannon. La confusion consiste à rendre une information intelligible à l'aide d'une fonction non linéaire ; la diffusion, quant à elle permet de répartir une information dans plusieurs positions du corps du bloc résultat afin que sa modification génère une avalanche de changements dans toute la structure. Ce concept est réalisé par divers réseaux de calcul tel le réseau substitution-permutation, réseaux de Feistel ou Sboxes.

a. Réseaux de substitution-permutation

Un algorithme de *substitution-permutation* (SPN en abrégé) est constitué d'une suite de tours. Au niveau de chaque tour (fig. 2.3), les bits du bloc de données sont soumis à une série de substitutions et de permutation d'une manière séparée ou combinée avec la clé de chiffrement utilisée.

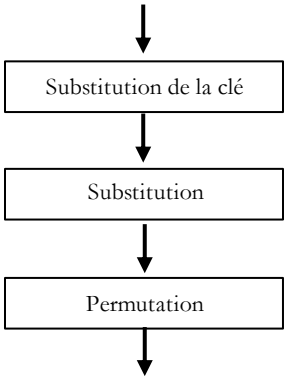

Fig. 2.3 : Tour d'un réseau SPN

Cette construction est utilisée dans plusieurs certains chiffrements symétriques notamment l'AES (Daemen & Rijmen, 2000), SERPENT (Anderson, Biham, & Knudsen, 2000), PRESENT (Bogdanov, et al., 2007), PUFFIN (Cover & Thomas, 1991), etc.

b. Boites-S :

Connues sous le nom de *Sboxes*, une boite S est une fonction booléenne vectorielle agissant sur un petit nombre de bits. Elle permet d'apporter une confusion reliant les bits d'entrée et de sortie. Les Sboxes sont décrites par des tables de plusieurs sorties qui peuvent être structurées en tables statiques donc, occupent peu de place en mémoire mais gourmands en temps de calcul nécessaire aux multiples opérations d'accès à leur contenu.

La manière de conception des Sboxes demeure non publique ; Elle est basée sur la non linéarité et la diffusion permettant ainsi de résister aux diverses attaques notamment la cryptanalyse linéaire et différentielle.

c. Réseaux de Feistel

Appelés également schémas de Feistel (Feistel, 1973), les réseaux de Feistel sont des algorithmes de chiffrement itératifs basés sur les notions de *confusion* et *diffusion* agissant sur des demi-blocs de texte. Au niveau de chaque tour, le bloc de texte clair p est découpé en deux sous-blocs L et R de même taille; le résultat du tour i est un bloc chiffré c_i avec une sous clé k_i obtenue de la manière illustré par la relation 2.12.

$$c_i = \sigma_i(p_{L_i}, p_{R_i}) = \left(p_{L_{i-1}}, p_{R_{i-1}} \oplus f_i(p_{R_{i-1}}, k_i)\right) \tag{2.12}$$

L'avantage d'une telle transformation est qu'elle est réversible sans nécessité que la fonction *f* n'en soit ainsi. En effet, on remonte un tour par application de la même relation :

$$\sigma^2(L,R) = \sigma_i(L \oplus f(R,k_i) = (R, L \oplus f(R,k_i) \oplus f(R,k_i), L) = (L,R) \quad (2.13)$$

La transformation ainsi décrite est loin d'être aléatoire à moins d'utiliser un nombre acceptable de tours successifs (Patarin, 2004).

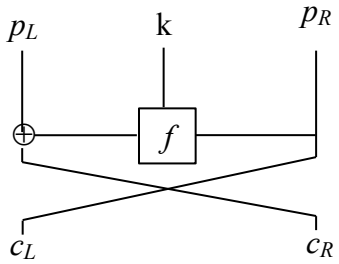

Fig. 2.4 : Tour d'un réseau de Feistel

Depuis sa création, les réseaux de Feistel ont été objet de plusieurs améliorations, notamment les réseaux généralisés (Nyberg, 1996) et les réseaux non équilibrés (Schneier & Kelsey, 1996). De même, les résultats théoriques relatifs à la sécurité des schémas de Feistel montrent que si la fonction *f* est pseudo-aléatoire, trois tours de ce schéma suffisent à produire une permutation pseudo-aléatoire (Michael & Charles, 1988). Cette conclusion a sollicité l'intégration de ces schémas comme composant principal de plusieurs cryptosystèmes symétriques tels Blowfish (Schneier, 1993), LUCIFER (Feistel, 1991), CAST (Carlisle, 1997) et également asymétrique, tel le procédé OAEP[45] (Bellare & Rogaway, 1995) utilisé comme entrée d'une permutation à trappe de la fonction RSA évitant ainsi les attaques à chiffrés choisis.

2.6.5 Data Encryption Standard

a. Définition :

Le *Data Encryption Standard*, connu par son abréviation *DES* (Standards, 1977) est le plus célèbre des algorithmes de chiffrement symétriques. Initialement conçu par IBM au cours des années 70, remanié par la NSA en 1976, le DES s'est vu devenir la référence en terme de chiffrement symétrique.

A l'origine, le DES se prénommait *Lucifer*, agissait sur des blocs de 128 bits et d'une clé de taille équivalente et, à cause d'une faille de chiffrement (Ben-Aroya & Biham, 1996), Il fût normalisé à sa taille actuelle par la NSA.

Basé sur les réseaux de Feistel, le DES opère sur des blocs de 64 bits avec des clés de 56 bits en utilisant une transformation *f* à 16 tours. Chaque tour utilise une sous clé de 48 bits générée à

45. Optimal Asymetric Encryption padding

base de la clé principale par une méthode qui dépend du tour considéré. Le chiffrement est couronné par deux permutations, initiale et son inverse en final.

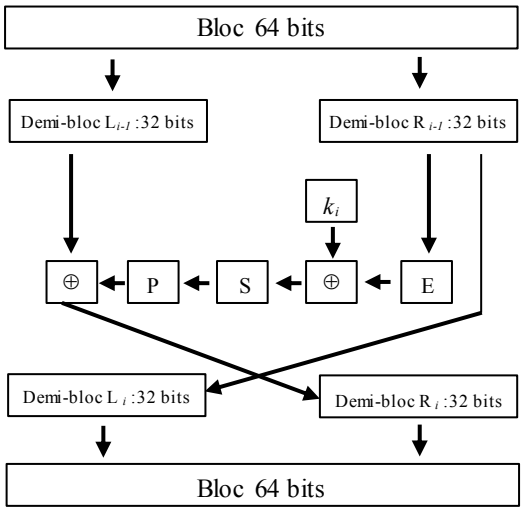

Fig. 2.5 : Tour de l'algorithme DES

Actuellement, et avec la puissance de calcul disponible, le DES n'est plus recommandé et considéré comme non sûr à cause de taille de la clé, jugée trop réduite et de la lenteur de son exécution. Il a été souvent objet d'attaques réussies (voir §2.6.5.d) et par conséquent, renforcé par une version complexe, le triple DES qui utilise deux clés (k_1, k_2), et une fonction de chiffrement *f*, dérivée de celle du DES par la formule $E_{k_1}(D_{k_2}(E_{k_2}(m)))$, ave E et D désignent les opérations de chiffrement et déchiffrement. Finalement, c'est l'AES qui a été adopté actuellement. Avec une clé de taille variable de 128 à 256 bits et répond aux même exigences que le triple DES. Il est considéré jusqu'à lors résistant à tous types d'attaques et a été également intégré dans les protocoles de chiffrement des réseaux mobiles.

b. Structure :

- Clé

La clé de chiffrement DES est codé sur 64 bits ; le dernier bit de chaque octet est un code de parité relatif aux autres bits du même octet. Le DES utilise 16 sous-clés différentes de 56 bits chacune, générées à partir de la clé principale. Au cours du chiffrement, la clé principale est réduite à 56 bits par suppression des bits de parité et d'une permutation dénommée *PC1* décrite par le tableau suivant :

Demi-bloc gauche							Demi-bloc droit						
57	49	41	33	25	17	9	63	55	47	39	31	23	15
1	58	50	42	34	26	18	7	62	54	46	38	30	22
10	2	59	51	43	35	27	14	6	61	53	45	37	29
19	11	3	60	52	44	36	21	13	5	28	20	12	4

Tab. 2.4 : Permutation PC1

Ensuite, pour chaque itération, les bits de la clé principale subiront des décalages circulaires à gauche d'un ou deux bits selon chaque itération de la manière suivante :

Itération	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Nbre de décalages	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tab. 2.5 : Décalage relatif aux sous-dés

A titre d'exemple, si la clé principale de 64 bits est :

k^{64}=0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010 1000 0000 1000 0000

Après suppression des caractères de parité et permutation à l'aide du tableau PC1, on obtient :

k^{56}= 1100 0000 0000 0000 0000 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000

La clé relative à la première itération subira un seul décalage à gauche de chacune de ses deux parties d'une position :

k_1^{56}= 1000 0000 0000 0000 0000 0000 0001 0100 0000 0000 0000 0000 0000 0000 0000

Ensuite, la clé k_i de l'itération i est réduite à 48 bits par suppression de certains bits et permutation des autres conformément à une autre permutation nommée *PC2* illustrée par le tableau suivant :

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Tab. 2.6 : Permutation PC2

En appliquant cette permutation, la clé en question sera :

k_1^{48}= 0000 1001 0000 0000 0000 0000 0000 0010 0000 0000 0000 0000

Pour calculer k_2 par exemple, le processus est repris à partir de k^{56}.

- Permutation initiale *IP*:

La permutation initiale *IP* permet de déplacer chaque bit du bloc de texte clair d'une position à une autre selon la table de permutation suivante :

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tab. 2.7 : Permutation initiale IP

Cette opération déplace par exemple le 58ème bit à la première position, le 50ème à la deuxième position, le 60ème à la neuvième position, etc. Cette nouvelle disposition permet de créer une confusion au sein des bits du texte clair et cacher toute structure linéaire ou algébrique dans le système.

- Fonction d'expansion *E*:

La fonction d'expansion *E* prend en entrée le demi-bloc R de 32 bits et va l'étendre à la taille de la clé (48 bits) avec des répétitions de certains bits en utilisant le tableau suivant :

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

Tab. 2.8 : Expansion E

Dans ce tableau, on trouve certaines répétitions, par exemple le 32ème bit du demi-bloc va être déplacé à la première position et également à la 47ème position. Le 25ème bit est déplacé simultanément à la 36 ème et 38 ème position.

- Fonction de sélection *S*:

La fonction *S*, référence aux S-boxes (Raphael, 2007) est non linéaire car elle prend en entrée un bloc de 48 bits et retourne un autre de 32 bits. Son principe est de découper le bloc d'entrée en 8 sous-blocs de 6 bits chacun. Les quatre bits 2 à 5 et les deux bits 1 et 6 de chaque sous-bloc *i* désignent respectivement la ligne et la colonne du S-boxe i. Ces coordonnées ciblent une valeur de 4 bits qui remplacera les six bits du sous-bloc i.

S_0															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	8	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_1															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_2															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	12	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_3															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_4															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_5															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	8	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_6															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	3	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_7															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tab. 2.9 : Sboxes

- Fonction de permutation P :

C'est une fonction classique de permutation qui change les positions des bits du bloc de données sans toutefois modifier sa longueur. Elle est réalisée à l'aide du tableau de transposition suivant :

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tab. 2.10 : Permutation P

- Permutation finale IP^{-1}:

La permutation finale procède à une nouvelle perturbation des bits du bloc afin d'annuler la permutation initiale par une transposition inverse donnée par le tableau suivant :

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tab. 2.11 : Permutation finale IP^{-1}

Cette permutation déplace le $40^{ème}$ bit à la première position, le $8^{ème}$ à la deuxième position, etc.

c. Algorithme :

L'algorithme DES prend en entrée une clé k de taille $t=56$ et un bloc de texte clair de taille $n=64$ et transforme le tout en un bloc c de 64 bits. La fonction 2.5 illustre les diverses transformations réalisées au niveau d'un tour.

Fonction 2.5 Tour_DES(p, c, k)

$p_0=(L_0,R_0)$
Pour $i \leftarrow 1$ à 15 faire
  $L_i=R_{i-1}$
  $R_i=L_{i-1} \oplus f(L_{i-1},k_i)$
FinPour

- Exemple

Soit le bloc p de données claires suivant :

P_1^{64}=0101 1110 0111 0100 1111 1000 1010 0000 0111 1110 1100 0010 0001 1001 0011 0011.

Il sera subdivisé en deux sous-blocs :

L_1^{32}= 0101 1110 0111 0100 1111 1000 1010 0000 et

R_1^{32}= 0111 1110 1100 0010 0001 1001 0011 0011

Le bloc droit R_1 sera étendu à la l'aide de la fonction d'expansion E qui deviendra :

R_1^{48}= 1001 0101 1001 0100 1111 1100 0010 1010 1011 1101 1010 0010

En utilisant la clé définie en §4.5.2.a

k_1^{48}= 0000 1001 0000 0000 0000 0000 0000 0010 0000 0000 0000 0000

La combinaison du bloc de données avec la clé donne :

$R_1^{48} \oplus k_1^{48}$=1001 1100 1001 0100 1111 1100 0010 1000 1011 1101 1010 0010

Chaque sous-bloc de 6 bits sera réduit à 4 en utilisant les Sboxes. Pour le premier par exemple, **1**00111**1** ; les bits en gras 11 désignent la ligne et les autres, la colonne de la S_0; c'est-à-dire S_0(11,0011) ; la valeur correspondante est 2, donc 0011 en binaire.

Le résultat, un sous-bloc de 32 bits ; il subira encore une nouvelle permutation selon le tableau 2.10 sans modification de sa longueur. Il sera ensuite X-oré avec l'autre demi-bloc R_1. Le résultat est transféré à R_2. Le L_2 n'est autre que R_1. Le processus reprend de la même manière pour l'itération suivante.

d. Force et faiblesse :

Les diverses transpositions et substitutions du DES sont publiques et fixes. La sécurité de l'algorithme toute entière repose sur la clé. Il existe 2^{56} valeurs possibles de cette clé, soit $7{,}2 \times 10^{16}$ possibilités, ce qui lui assure un certain seuil de sécurité acceptable[46]. De même, les Sboxes permettent de générer des blocs de données basées sur des transpositions déclarées non linéaires ; la manière de construction des Sboxes ne semble pas connue ; néanmoins et jusqu'à lors aucune recherche n'a permis de leur associer un quelconque comportement linéaire. Ceci est peut être justifié par l'exemple de la table 2.12 qui montre l'effet d'avalanche qui, pour une modification de 1% de la clé produira une modification d'au moins 40% du résultat :

Clé	22234512987ABB23	
Texte clair	0000 0000 0000 0000	0000 0000 0000 0001
Texte chiffré	4789 FD47 6E82 A5F1	0A4E D5C1 5A63 FEA3

Tab. 2.12 : Exemple illustrant la propriété de la diffusion

46. A l'exœption de l'usage de dés faibles : Il y en a 4, '01 01 ..', 'E0 E0 ..', 'FE FE,…' et '1F, 1F, ..' pour lesquelles le DES devient une involution et les dés semi-faibles, il y en en a 4 également.

Malgré que le DES est encore sollicité aujourd'hui dans divers secteurs, il est considéré comme vulnérable à l'attaque exhaustive[47].

e. Variantes

- DES 4 Tours :

L'algorithme DES à 4 tours (4DES en abrégé) (Laskari, Meletiou, Stamatio, & Vrahatis, 2005) est une version du DES réduite à 4 tours. Elle utilise une fonction *f* qui combine la substitution, la combinaison et l'addition logique et ce, en 4 tours seulement en utilisant les mêmes tables Sboxes de DES.

L'algorithme 4DES est moins complexe que celui de DES mais toujours difficile à briser.

Vu que l'attaque à la clé DES à 16 tours nécessite énormément de ressources, la plupart des travaux de cryptanalyse et, afin de valoriser leurs résultats, utilisent dans leurs expérimentations des versions DES à nombre de tours réduit. Le détail de ces travaux est présenté au §8.2.2.

- DES Simplifié :

Le *DES simplifié* (SDES en abrégé) est une variante de DES introduite par (Schaefer, 1996). Il emploie des propriétés similaires que celui de DES avec des données de taille réduite. La clé principale de cet algorithme est de 10 bits seulement de laquelle dérive deux sous-clés de taille égale à 8 bits par des permutations binaires et un décalage circulaire simple.

L'Algorithme SDES utilise des *blocs* de données de 8-bits et des clés de 10-bits. Il emploie les mêmes transformations telles le DES mais à échelle réduite, à savoir :

- Permutation initiale IP est définie par la table [2,6,3,1,4,8,5,7] c'est-à-dire que le second bit du bloc de données vient en première place, suivi du 5ème bit, ensuite le 2ème, etc.
- La fonction d'expansion E est définie par la table [4,1,2,3,2,3,4,1] permettant d'étendre la taille d'un sous-bloc de 4 bits à 8 avec une répétition désordonnée de certains d'entre eux.
- La fonction de transformation f_k est définie à base d'opérations de permutation et substitution en utilisant la clé de chiffrement k_i. Elle opère sur des sous-blocs de 4 bits de la manière suivante :

$$f_{k_i}(L,R) = (L \oplus f(R,k_i),R) \tag{2.14}$$

où *L*, *R* désignent respectivement les sous-blocs de données; k_i, la sous-clé (il y en en a 2) et *f*, une addition exclusive du sous-bloc avec la première clé.

- La fonction de sélection *S* utilise deux Sboxes réduites à 4 x 4 bits définies de la manière suivante :

47. Le DES a été cassée en 3 semaines par la fédération de machines sur Internet en 1997. Un an après, il a été cassé en 56 heures par machine dédiée (http://www.eff.org/descracker). En 1999, il a été également cassé par attaque groupée d'un réseau couplé de 100.000 PC en 22 heures.

S1				S2			
1	0	3	2	0	1	2	3
3	2	1	0	2	0	1	3
0	2	1	3	3	0	1	0
3	1	1	2	2	1	0	3

Tab. 2.13 : Sboxes réduites

Chaque quatre bits du sous-bloc généré constituent respectivement la ligne et la colonne au niveau d'une Sboxe, elles désignent une valeur de deux bits. Les deux valeurs extraites des deux Sboxes forment un sous-bloc est réduit à 4 bits.

- La fonction SW réalise une permutation simple des deux sous-blocs.

$$SW(L,R) = (R,L) \tag{2.15}$$

- Une deuxième transformation telle définie en (2.11) est appliquée au bloc généré en utilisant la deuxième clé k_2.
- Enfin, l'inversion finale IP^{-1} est accomplie en utilisant la table définie par [4,1,3,5,7,2,8,6].

En pratique, le SDES est utilisé à de fins de laboratoire. Il permet de simuler le DES (en cas d'implémentation ou de cryptanalyse) avec une consommation raisonnable de ressources.

2.7 Conclusion :

La cryptographie était jadis un art réservé à certaines communautés civilisées, transformée ce dernier siècle en science, favorisée par l'avancée en matière de technologie numérique, devenue un outil incontournable de sécurité au niveau de divers secteurs clés de l'économie, notamment les banques et les télécommunications. Actuellement, la cryptographie et, comme toute autre technologie est en évolution permanente. Ceci est dû à sa contribution d'une manière efficace à la sécurité et la confidentialité des informations.

La problématique liée à l'utilisation d'un système cryptographique apparaît au sens de pouvoir cacher des informations *facilement* mais pas d'une manière *sûre*. La naissance de la cryptographie moderne a permis de reformuler la question à la manière de pouvoir cacher des informations *facilement* et d'une manière *efficace* mais pas *indéfiniment*. La solidité d'un système cryptographique est mesurée par la durée de résistance aux attaques à laquelle on ajoute certains risques 'humains' liés à la manière d'échange de certaines données (algorithmes, clés et codes) à travers les réseaux publics.

A base de ce constat, il est intéressant d'avoir la conscience que la sécurité d'un système de chiffrement n'est jamais garantie car elle évolue dans le temps et qu'on ne peut exclure l'éventualité d'une tentative d'attaque fructueuse un jour ou l'autre.

3. Cryptanalyse

3.1 Introduction :

La cryptanalyse fait référence aux codes, chiffrements, clés et systèmes y afférant, leur étude et analyse afin de localiser leur faiblesse et ce, pour but de révéler leur contenus sans nécessité d'avoir connaissance des outils normalement requis pour ce fait tels les clés, protocoles ou les algorithmes. Ce concept est connu sous l'appellation de *casser* ou *briser* un cryptosystème. Ces termes sont employés conjointement avec le terme *faiblesse* d'un chiffrement qui, ayant pour but d'exploiter cette caractéristique et réduire le nombre de clés d'essais permettant de retrouver le texte clair correspondant au chiffrement et par conséquent, contourner l'usage de la force exhaustive. Cette faiblesse a été matérialisée par diverses techniques expérimentales ayant plus ou moins des résultats acceptables liés aux circonstances d'utilisation tel la *rencontre-du-milieu*[1], le *paradoxe des anniversaires*[2], etc.

La cryptanalyse utilise les outils mathématiques adéquats tels les statistiques, l'analyse de langages et exclus éventuellement toutes les techniques liées à la corruption notamment les *keyloggers*[3], le *phishing*[4] ou *l'ingénierie sociale*[5] bien que ces dernières sont souvent plus efficaces que les attaques classiques.

---

1. Manière d'interception illicite d'échange d'informations entre deux parties. Elle vise essentiellement les clés ou codes de chiffrement, les substituts à d'autres et les remis en réseaux. Les parties communicantes finissent par utiliser des clés connues par l'intercepteur.
2. Technique d'attaque visant certaines contraintes liées aux fonctions de hachage notamment l'absence de collisions réduisant ainsi le nombre de clés d'essais lors d'une cryptanalyse. Ce genre d'attaque est utilisé pour casser des techniques du type MD5, SHA-1, LM, etc.
3. Enregistreur de frappe : Dispositif matériel ou logiciel qui enregistre les frappes du clavier à l'insu de l'utilisateur. Utilisé par les malfrats comme outil de récupération des codes d'accès notamment lors de l'usage des distributeurs automatiques de billets.
4. Le «phishing» (ou « hameçonnage ») est une technique utilisée par les fraudeurs visant à soutirer des informations personnelles à leurs victimes en se faisant passer pour un tiers de confiance.

La cryptanalyse, bien qu'elle parait irrégulière au sens qu'elle est utilisée pour briser les chiffres et apprendre les secrets, elle permet de révéler leur faiblesse qui, prise en considération par les concepteurs de cryptosystèmes, leur permet de créer d'avantage de modèles plus robustes.

3.2 Cryptanalyse de chiffrements :

3.2.1 Définition :

En termes intuitifs, une attaque contre un cryptosystème est une manière de déterminer des informations permettant de mettre en clair le contenu d'un texte chiffré ou du moins comprendre son sens et ce, à partir du texte chiffré lui-même et peut-être certaines autres informations (dans la mesure du possible) relatives à environnement de chiffrement.

En général, la cryptanalyse exploite les failles d'un système : structure de l'algorithme de chiffrement, termes ou expressions pouvant être contenus dans le texte liées aux habitudes de son propriétaire ou utilise des outils mathématiques tels la différence entre divers textes chiffrés par un même procédé.

Malgré son évolution et sa persistance à l'ordre du jour dans le monde de recherche, la cryptanalyse ne peut être considérée comme une science exacte, elle est plutôt basée sur l'expérience, l'intuition et favorisée par les outils mathématiques et statistiques et la puissance de calcul informatique. Néanmoins, le terme '*chance*' était toujours lié à sa réussite

3.2.2 Principe :

Le *cryptanalyste* requiert un certain nombre d'informations avant d'aborder la procédure d'attaque. Il doit connaître bien entendu la langue avec laquelle le texte chiffré est écrit et le détail de l'algorithme de chiffrement. Ainsi, et selon le principe de *Kerchhoffs*, la seule donnée transparente est la clé de chiffrement qui doit être retrouvée et utilisée pour déchiffrer le message. Certaines autres données accordent un certain niveau de progression pour la tâche de l'attaque, notamment :

- La disponibilité d'un certain nombre de textes chiffrés par une clé unique utilisés en guise de référence pour aborder d'autres textes similaires : l'attaquant a la possibilité de retrouver la clé de chiffrement,
- La disponibilité d'un certain nombre de textes chiffrés ainsi que les textes clairs correspondants avec une ou plusieurs clés : l'attaquant a la possibilité de déterminer la manière (ou l'algorithme) permettant de déchiffrer n'importe quel autre message,
- La disponibilité d'un certain nombre de textes chiffrés avec la même clé et leurs correspondants en clair d'une manière permettant de connaître certaines spécificités du cryptosystème, notamment la longueur de la clé, les caractéristiques du langage utilisé.

---

5. Comprend les techniques d'escroquerie, de tromperie. Elle consiste à la manière de soutirer des informations de la part des utilisateurs par tous moyens disponibles, à savoir téléphone, courrier électronique, courrier traditionnel, messagerie instantanée ou contact direct. Par exemple, en se faisant passer pour un administrateur du réseau ou bien à l'inverse appeler l'équipe de support en demandant de réinitialiser le mot de passe en prétextant un caractère d'urgence.

- La disponibilité de certaines références ou spécificités quant au sujet du texte chiffré, son auteur ou les circonstances de son élaboration afin de localiser certaines expressions se rapportant au contenu du texte permettant de procéder aux attaques par *dictionnaire* ou *différentielle*.

En général, si l'algorithme de chiffrement est assez consistant et si la clé est suffisamment longue, l'attaque prometteuse ne peut être autre que la force brute dont les limites sont marquées par le temps nécessaire à la découverte de la clé correcte qui se chiffre par des dizaines d'années sur des machines d'une puissance proportionnelle. Donc, cette technique ne peut être envisagée qu'en cas désespérés ; il parait cependant mieux de recourir à certaines failles de l'algorithme telles mentionnées ci-dessus ou aux attaques spécifiques (différentielle, linéaire, statistique).

3.3 Techniques de cryptanalyse de chiffrements classiques

3.3.1 Attaque par dictionnaire

Au lieu de générer des clés d'essai aléatoires, *l'attaque par dictionnaire* permet de choisir des clés significatives, en autres termes, faisant partie d'une liste d'expressions prédéfinie reflétant les habitudes de l'auteur du chiffrement quant au choix de ses codes d'accès.

Ainsi, cette attaque est généralement utilisée conjointement avec une force brute pour casser les clés d'accès et mots de passe de taille réduite. Elle n'est pas efficace quant aux clés générés aléatoirement car les dictionnaires utilisés comportent généralement des expressions significatives : noms de personnalités, de marques, numéros divers, etc. Afin de rendre la technique fructueuse, des règles de transformations d'expressions seront alors utilisées augmentant ainsi le nombre de combinaisons possibles telles le mélange entre majuscules et minuscules, le remplacement d'un chiffre par un caractère similaire : « 5 » par « S » par exemple ou l'ajout de certains caractères au début ou en fin du mot.[6]

Ce genre d'attaque est généralement employé pour réduire d'une manière intelligente le nombre de clés d'essai sous réserve que le dictionnaire soit conforme à l'environnement du chiffrement utilisé.

3.3.2 Technique du mot probable

Consiste à supposer qu'une séquence de lettres correspondante à un mot donné est cachée dans le corps du chiffrement. La dite séquence (*cribbing* en anglais) est déterminée selon l'environnement de transmission tel l'envoi de messages à une heure déterminée où comportant le nom de son auteur, une formule de politesse, etc. Ce type d'attaque semble fonctionner sur la plupart des chiffrements classiques, notamment le chiffre de Vigenère, les substitutions homophoniques, ou encore le chiffre de Hill.

6. Des dictionnaires divers avec des applications associées sont disponibles sur le net tel *John-the-Ripper* ou *Cain-&-Abel*.

3.3.3 Analyse de fréquences

a. Définition

Consiste à l'analyse de la fréquence de caractères au sein d'un chiffrement donné. Ce principe permet de renseigner sur les propriétés du texte analysé, notamment le langage avec lequel il a été écrit ; le type du chiffrement utilisé, etc. Ce principe est employé fréquemment, sous certaines conditions, lors de l'attaque aux chiffrements afin de révéler certaines propriétés du texte chiffré pouvant conduire à son déchiffrement partiel.

L'application de cette technique consiste à une *analyse de fréquence* du texte chiffré en remplaçant bien entendu certains de ses caractères (de forte et faible fréquence) avec des caractères ayant des fréquences proches extraits de la table de fréquences relative au langage utilisé.

b. Fréquence de caractères

La fréquence d'apparition de caractères au sein d'un texte quelconque diffère d'un langage à un autre. L'idée a été démontrée d'après un examen de la fréquence d'apparition des caractères de l'alphabet de divers langages en se basant sur des échantillons de textes extraits d'un nombre de sources d'informations de la littérature et les médias.

L'idée originale de cette théorie provient de *Al-Kindi*[7]. Elle a été étendue à d'autres langages et a fait objet de plusieurs améliorations notamment l'omission de caractères à faible usage, la ponctuation ou les symboles dont la fréquence ne peut être objective tels les caractères spéciaux et les chiffres. L'étude s'est penchée également sur les groupes contigus de caractères ou *syllabes* dont la fréquence peut intéresser les cryptanalystes.

La littérature comporte diverses versions de tables de fréquences construites à base de statistiques provenant de sources diverses. La différence entre ses tables est due à la nature de la source de données utilisée pour l'analyse et de sa taille. La table 3.1 illustre un exemple de la fréquence des n-grams au sein de certaines langues latines.[8] :

Au sein de la plupart de ces langages, le caractère isolé (*1-gram*) le plus marquant est le « E » avec une fréquence qui dépasse les 10%, suivi du caractère « A » et « T » avec plus de 6%. Au-delà de ces deux caractères, on remarque une diversité d'apparition des autres caractères notamment les caractères « R » et « S » et les voyelles « I » et « O ». On trouve en fin de liste les derniers caractères de l'alphabet, le « W », « X », « Y » et « Z » avec fréquence approximative ne dépassant pas le 1%.

Les caractères contigus ou syllabes (*n-gram*) les plus fréquents diffèrent considérablement d'un langage à l'autre. Pour l'Anglais et le Français par exemple, les n-grams les plus fréquents sont illustrés dans les tableaux 3.2 et 3.3.

---

7. *AbuYusufyakoubIshaq Al kindi* (801-873):philosophe arabe ayant rédigé divers ouvrages sur l'arithmétique, la géométrie, l'optique et les algorithmes. Il publia son ouvrage sur la cryptanalyse (manuscrit sur le déchiffrement des messages cryptographiques).

8. http://en.wikipedia.org/wiki/Letter_frequency. (Consulté le 10/11/2014)

Lettre	Allemand	Anglais	Espagnol	Français	Italien	Portugais
A	6.51	8.167	11.525	7.636	11.74	14.63
B	1.89	1.492	2.215	0.901	0.92	1.04
C	3.06	2.782	4.019	3.260	4.5	3.88
D	5.08	4.253	5.510	3.669	3.73	4.99
E	17.40	12.702	12.681	14.715	11.79	12.57
F	1.66	2.228	0.692	1.066	0.95	1.02
G	3.01	2.015	1.768	0.866	1.64	1.30
H	4.76	6.094	0.703	0.737	1.54	1.28
I	7.55	6.966	6.247	7.529	11.28	6.18
J	0.27	0.153	0.443	0.613	0.00	0.40
K	1.21	0.772	0.011	0.049	0.00	0.02
L	3.44	4.025	4.967	5.456	6.51	2.78
M	2.53	2.406	3.157	2.968	2.51	4.74
N	9.78	6.749	6.712	7.095	6.88	5.05
O	2.51	7.507	8.683	5.598	9.83	10.73
P	0.79	1.929	2.510	2.521	3.05	2.52
Q	0.02	0.095	0.877	1.362	0.51	1.20
R	7.00	5.987	6.871	6.693	6.37	6.53
S	7.27	6.327	7.977	7.948	4.98	7.81
T	6.15	9.056	4.632	7.244	5.62	4.74
U	4.35	2.758	2.927	6.311	3.01	4.63
V	0.67	0.978	1.138	1.838	2.10	1.67
W	1.89	2.360	0.017	0.074	0.00	0.01
X	0.03	0.150	0.215	0.427	0.00	0.21
Y	0.04	1.974	1.008	0.128	0.00	0.01
Z	1.13	0.074	0.517	0.326	0.49	0.47

Tab. 3.1. : Fréquence de caractères de certains langages latins

c. Analyse de fréquence de caractères

La construction des tables de fréquences de caractères constitue un travail délicat. Il dépend de la façon adoptée pour le recueil et l'analyse des textes dédiés à cet effet. Evidemment, l'analyse des textes extraits de l'ancienne littérature ne peut donner des résultats similaire à l'analyse des textes scientifiques actuels ou de courriers élaborés à base de langage public de bas niveau ou plutôt des textes commerciaux comportant en grande partie, des chiffres, des abréviations et des caractères spéciaux. Cependant la nature du texte lui-même peut avoir un effet sur la qualité du résultat lui-même si on tient compte de certaines spécifiés, notamment :

- Le style de rédaction : un texte relatif à une description quelconque, un conte par exemple contient certainement plus d'adjectifs (donc, plus de « ent ») qu'un texte de dialogue ou de discussion où il y a d'avantage de verbes d'interrogation à la 2e personne du pluriel contenants significativement plus de « z » qu'il en soit nécessaire.

1-gram	2-gram	3-gram	4-gram	double	mots
E	TH	THE	TION	SS	THE
T	HE	AND	ATIO	EE	OF
A	IN	ING	THAT	TT	AND
O	ER	ION	THER	FF	TO
I	AN	TIO	WITH	LL	IN
N	RE	ENT	MENT	MM	IS
S	ON	ATI	IONS	OO	THAT
R	AT	FOR	THIS		BE
H	EN	HER	HERE		IT
L	ND	TER	FROM		BY
D	TI	HAT	OULD		ARE
C	ES	THA	TING		FOR
U	OR	ERE	HICH		WAS
M	TE	ATE	WHIC		AS
F	OF	HIS	CTIO		HE
P	ED	CON	ENCE		WITH
G	IS	RES	HAVE		ON
W	IT	VER	OTHE		HIS
Y	AL	ALL	IGHT		AT
B	AR	ONS	SION		WHICH
V	ST	NCE	EVER		BUT
K	TO	MEN	ICAL		FROM
X	NT	ITH	THEY		HAS
J	NG	TED	INTE		THIS
Q	SE	ERS	OUGH		WILL
Z	SI	ERI	THRI		OUT

Tab. 3.2. : Fréquence de caractères de la langue Anglaise (Mayzner, 1965)

- Le vocabulaire spécifique du document : si l'on parle de chemins de fer, il y aura beaucoup plus de « W » ; si ça relève à l'étude des rayons X, il y'en aura d'avantage de « x », ..etc.[9]

Ainsi, les tableaux de statistiques sont sujets à de nombreuses critiques et recommandations car elles sont élaborées à base de différentes sources de données.

9. Les exceptions sont abondantes tel le passage suivant : « De Zanzibar à zambie et au Zaïre, des zones d'ozone font courir les zèbres en zigzag zinzins » ou le roman de George Perec, « La disparition » publié en 1969 par Gallimard, Paris (Collection imaginaire). Avec ses 319 pages ne comportant pas une seule fois la lettre « e ». dont tiré l'extrait suivant : '*Un voisin compatissant l'accompagna à la consultation à l'hôpital Cochin. Il donna son nom, son rang d'immatriculation à l'Association du travail. On l'invita à subir auscultation, palpation, puis radio. Il fut d'accord. On l'informa : sourait-il ? Plus ou moins, dit-il. Qu'avait-il ? Il n'arrivait pasà a dormir ? Avait-il pris un sirop ? Un cordial ? Oui …*'

1-gram	2-gram	3-gram	4-gram	Répartition	Voyelles
E	ES	ENT	TION	SS	OU
S	LE	LES	MENT	EE	AI
A	DE	ION	EMEN	LL	IE
I	EN	DES	DELA	TT	EU
T	ON	EDE	ATIO	NN	UE
N	NT	QUE	IQUE	MM	UI
R	RE	EST	ELLE	RR	AU
U	AN	TIO	DANS	PP	OI
L	LA	ANT	POUR	FF	
O	ER	PAR	ESDE	CC	
D	TE	MEN	EDES	AA	
C	EL	DEL	ONDE	UU	
P	SE	ELA	IOND	II	
M	TI	SDE	IONS	GG	
V	UR	LLE	ANSL		
Q	ET	OUR	AIRE		
F	NE	RES	PLUS		
B	IS	SON	ILLE		
G	ED	TRE	QUEL		
H	OU	ONT	SONT		
J	AR	EUR	EDEL		
X	IN	ATI	QUES		
Y	IT	UNE	COMM		
Z	ST	CON	ENTD		
W	QU	EME	EURS		
K	NS	ANS	NTDE		

Tab.3.3. Fréquence de caractères de la langue Française. (Sacco, 1947)

Dans la littérature, on rencontre diverses approches d'analyse fréquentielle ; certaines regroupent les caractères portant le même diacritique, d'autres omettent les ligatures, la ponctuation ou encore les chiffres, d'autres fusionnent les styles parlés avec le langage écrit, etc. Cependant, aucune approche n'est parfaite car la fréquence d'apparition de caractères varie d'un échantillon à un autre. Le tableau 3.4 illustre certains échantillons de ce fait.

Intuitivement, plus le texte utilisé pour mesurer la fréquence de caractères est long et varié, mieux que c'est proche du réel. Il parait donc nécessaire de choisir un référentiel de données[10] composé d'un nombre suffisant de textes provenant de divers domaines de la littérature et

10. Cette source ne peut être un dictionnaire car ce dernier est dépourvu d'expressions plurielles ou de verbes conjugués ce qui influe sur le nombre de certains caractères qui apparaissent couramment au niveau des expressions linguistiques tels le 's' ou le 't'.

l'actualité afin de concevoir des tables de fréquences acceptables. Cette problématique a permis d'élaborer des modèles de données qui reflètent le standard des caractéristiques de chaque langage littéraire en structure et également en taille. Les résultats ont été baptisés *Corpus* (Robert, 2000) (Nelson, Sean, & Bas, 2000).

Car	([11])	([12])	([13])	([14])	([15])	([16])
A	8.16	8.04	8.12	8.55	8.3	8.56
B	1.49	1.54	1.49	1.60	1.5	2.12
C	2.78	2.71	2.71	3.16	2.7	4.74
D	4.25	4.14	4.32	3.87	4.1	3.13
E	12.7	12.51	12.02	12.10	12.6	11.42
F	2.22	2.30	2.30	2.18	2	1.47
G	2.01	1.92	2.03	2.09	1.9	2.3
H	6.09	5.49	5.92	4.96	6.1	2.76
I	6.96	7.26	7.31	7.33	6.7	7.94
J	0.15	0.11	0.10	0.22	0.2	0.15
K	0.77	0.67	0.69	0.81	0.8	0.84
L	4.02	3.99	3.98	4.21	4.2	5.52
M	2.4	2.53	2.61	2.53	2.5	3.22
N	6.74	7.09	6.95	7.17	6.8	6.41
O	7.5	7.60	7.68	7.47	7.7	7.12
P	1.92	1.73	1.82	2.07	1.6	3.27
Q	0.09	0.16	0.11	0.10	0.1	0.23
R	5.98	6.12	6.02	6.33	5.7	7.51
S	6.32	6.54	6.28	6.73	6.11	5.55
T	9.05	9.25	9.10	8.94	9.3	7.46
U	2.75	3.06	2.88	2.68	2.8	3.66
V	0.97	0.99	1.11	1.06	1	1.07
W	2.36	1.96	2.09	1.83	2.3	0.94
X	0.15	0.19	0.17	0.19	0.2	0.35
Y	1.97	2.00	2.11	1.72	2	2
Z	0.07	0.09	0.07	0.11	0.1	0.24

Tab. 3.4. Fréquence de caractères d'Anglais

11. http://commons.wikimedia.org/wiki/File:English_letter_frequency_%28alphabetic%29.svg (consulté le 1/08/2014)

12. http://scottbryce.com/cryptograms/stats.htm (consulté le 4/11/2014)

13. http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html (consulté le 28/10/2014)

14. http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies: (consulté le 21/05/2014)

15. http://www.sttmedia.com/characterfrequency-english (consulté le 1/11/2014)

16. http://www.rinkworks.com/words/letterfreq.shtml (consulté le 1/11/2014)

d. Corpus:

Ensemble de documents provenant de sources diverses appartenant à un langage littéraire donné. Contrairement aux corpus linguistiques dédiés, en cryptanalyse, un corpus doit être hétérogène afin de simuler le plus possible le langage naturel qui reflète la nature des données échangées à travers les réseaux, notamment les courriers électroniques, les forums ainsi que les messages de la téléphonie mobile.

Afin de répondre à ces exigences, un corpus dédié à la cryptanalyse doit répondre à certains critères, notamment :

- La *taille* : un corpus doit atteindre une taille critique afin de permettre des traitements fiables quant à l'élaboration des tables statistiques des n-grams.
- Le *langage* : un corpus doit appartenir à un langage précis. Néanmoins, il peut y avoir de tolérances quant aux textes issus du langage naturel tel le contenu des forums publics ou les différences des divers accents parlées d'un même langage.
- *L'évolution* : un langage peut évoluer au cours du temps, ceci est dû généralement à la civilisation et l'avancée scientifique qui produisent de de nouvelles procédures de vie ou d'interaction entre les communautés hétérogènes générée par les moyens de communication ou les guerres. Ainsi, un corpus élaboré à base de la littérature actuelle doit forcément être différent d'un autre produit à base de la littérature de l'histoire ancienne. Cependant et, afin d'être logique, un corpus doit prendre en compte la littérature d'un intervalle de temps assez vaste.
- *Spécificité* : un corpus doit englober un ensemble de thèmes spécifiques : science, littérature, médias, art ou tout autre domaine actif.

La construction d'un corpus dédié à la cryptanalyse est très délicate ; il n'y a aucun critère qui permet de mesurer son efficacité en l'absence d'un standard de référence car, un corpus n'existe pas en soi-même, mais dépend du positionnement théorique à partir duquel on l'envisage. Cependant, la problématique liée au recueil et à la sélection des données destinées à enrichir le corps du corpus doit prendre en considération certaines directives, à savoir :

- *Type et nature des données* : les données doivent être variées (écrites et orales), originaires de divers supports physiques (livres, journaux, rapports, courriers, affiches) et oraux (média, discussions réunions et meeting, conversations du quotidien, etc.),
- *Importance* et valeur des données : vu le volume gigantesque des informations disponibles, une sélection est donc nécessaire. Les données sélectionnées doivent satisfaire une certaine représentativité de l'actualité informative qui évolue continuellement et exige une mise à jour permanente du corpus,
- *Structure* des données : le contenu d'un corpus ne reflète pas des phrases compréhensibles ou syntaxiquement correctes ; c'est plutôt une amalgame de mots ou d'expressions les plus répandues dans un langage courant tels la grammaticale (articles, pronoms, verbes, etc.) ou lexicale (variété de synonymes) ou conjugale (diverses formes de verbes ou pronoms, etc.),
- *Pertinence* de données : vu le volume important de données similaires, la sélection doit être précédée d'une analyse qui procédera à un choix par élimination ou par un jeu de construction et destruction d'expressions afin de retenir les données les plus objectives.

En littérature, on rencontre une multitude de corpus ayant des caractéristiques plus ou moins différents. Les corpus les plus utilisés sont les suivants :

Pour la littérature française :

- Corpus Thomas Tempé[17]
- Corpus français [18]
- Corpus Wikipedia-FR[19]
- Corpus DEDE[20]
- Thesaurus synonymes[21]

Pour la littérature anglaise, on a :

- BrownCorpus[22]
- Concise Oxford Dictionary[23]
- Google Books Corpora[24]
- NYT Corpus[25]
- Leipzig English Corpora[26]

L'analyse statistique de l'apparition des caractères au sein de certains des corpus mentionnés donne des résultats sensiblement différents tels mentionnés dans le tableau 3.5. La différence qui apparaît dans les statistiques révélées au sein de chaque corpus est due essentiellement à la nature et la taille des données analysées. Afin d'ajuster cette problématique, certaines études statistiques ont adopté un ordre de classement de l'apparition de caractères au lieu d'un pourcentage et ce,

17. Regroupant plus de 1.5 millions de caractères provenant de diverses sources : littérature, messagerie électroniques et codes source divers. Source : http://web.archive.org/web/20080213211515/http://gpl.insa-lyon.fr/Dvorak-Fr/CorpusDeThomasTemp%C3%A9. (Consulté le 15/11/2014).

18. Base de 3 milliards de caractères extraits particulièrement de médias écrite, pages web et données d'encydopédie électronique. Source : http://wortschatz.uni-leipzig.de/ws_fra. (Consulté le 15/11/2014).

19. Extrait de l'édition française de l'encydopédie Wikipedia de juin 2008, le corpus contient plus de 1 milliard de caractères. Source : http://redac.univ-tlse2.fr/corpus/wikipedia.html.

20. Base de données de littérature regroupe une série d'artides du journal 'Le Monde', d'une taille dépassant les 240000 caractères. Source : http://www.cnrtl.fr/corpus/DEDE/. (Consulté le 15/11/2014).

21. Base de synonymes comprenant plus de 1 million de caractères répartis sur plus de 240 000 mots environs. Source : http://www.webcontentspinning.com/langage/. (Consulté le 15/11/2014).

22. Une base de plus de 4 millions de caractères composant des passages appartenant à une douzaine de domaines différents : presse écrite, ouvrages littéraires, politiques, scientifiques, romans, etc. Source : http://en.wikipedia.org/wiki/Brown_Corpus.

23. Dictionnaire de 240 milles entrées. Source : http://en.algoritmy.net/artide/40379/Letter-frequency-English.

24. Base de numérisation d'ouvrages Google. La version de 2011contient plus de 6500 milliards de caractères tirés de peu de 5.5 millions de livres. Source : http://googlebooks.byu.edu/# (consulté le 15/11/2014)

25. Le New York Times Corpus de janv à mars 1992, une base de presse de 70 millions de caractères. Source

26. Corpora multilingue extraite de la littérature. La base de l'anglais contient 4 millions de caractères extraites de la littérature. Extraite Source : http://corpora.informatik.uni-leipzig.de/

dans le but d'uniformiser les diverses résultats et de standardiser l'usage des corpus quels que soient leur source de données.

	Français		Anglais		
	Thomas Tempé	Corpus Français	Concise OD	NYT	Leipzig Corpora
A	7.246	9.38	8.167	8,43	8.55
B	0.855	1.54	1.492	1,57	1.6
C	3.094	1.49	2.782	3,33	3.16
D	3.481	4.70	4.253	3,80	3.87
E	13.980	10.15	12.702	11,98	12.1
F	1.012	2.03	2.228	2,12	2.18
G	0.822	2.86	2.015	1,98	2.09
H	0.699	2.09	6.094	4,68	4.96
I	7.144	5.82	6.966	7,22	7.33
J	0.517	0.61	0.153	0,22	0.22
K	0.046	3.14	0.772	0,77	0.81
L	5.177	5.28	4.025	4,04	421
M	2.816	3.47	2.406	2,62	2.53
N	6.732	8.54	6.749	7,20	7.17
O	5.121	4.48	7.507	7,35	7.47
P	2.867	1.84	1.929	2,13	2.07
Q	1.292	0.02	0.095	0,10	0.1
R	6.218	8.43	5.987	6,51	6.33
S	7.542	6.59	6.327	6,83	6.73
T	6.874	7.69	9.056	8,86	8.94
U	5.988	1.92	2.758	2,54	2.68
V	1.545	2.42	0.978	1,04	1.06
W	0.108	0.14	2.360	1,71	1.83
X	0.367	0.16	0.150	0,20	0.19
Y	0.292	0.71	1.974	1,76	1.72
Z	0.129	0.07	0.074	1,02	.11

Tab. 3.5. Fréquence de caractères de certains Corpus

Ainsi et, en Anglais par exemple, l'expression « etoin shrdlu » [*eh tay oh ! in shird loo*] est reconnue contenir les douze lettres de l'alphabet les plus fréquents selon leur ordre d'apparition. A quelques exceptions près, la plupart des corpus se concordent sur cette idée. De la même manière, on peut évaluer la queue qui devait contenir les caractères « kxvjqz ». Néanmoins et, en cryptanalyse classique, on ne tient pas compte de la queue de liste car la fréquence d'apparition de ces caractères est tellement faible qu'on a tendance à les confondre entre eux, ce qui perturbe la qualité du résultat. Dans ce contexte, certains corpus ont adopté cette idée et, au lieu d'affecter un score pour chaque caractère et, qui ne peut être significatif pour les caractères de faible fréquence,

ont préféré trier ces caractères selon leur ordre d'apparition dans les textes correspondants. Le tableau 3.6 illustre certains corpus et leur classement de caractères.

Corpus	Ordre de caractères
David Copperfield	etaoinhsrdlmuwycfgpbvkxjqz
Pride and Prejudice	etaoinhsrdlumcywfgbpvkzjxq
Wuthering Heights	etaonihsrdlumcyfwgpbvkxjqz
Vanity Fair	etaonhsirdlumcwfgypbvkjqxz
Gulliver's Travels	etoainshrdlmucfwygpbvkxjqz
Alice in Wonderland	etaoihnsrdluwgcymfpbkvqxjz

Tab. 3.6. Classement de caractères de certains corpus

3.3.4 Test de Friedman

Soit un texte p écrit dans un langage L. L'indice de coïncidence I_t est la probabilité pour que deux caractères tirés au hasard de p soient identiques.

a. Indice de coïncidence standard

Si le texte p est de taille n et, contenant tous les caractères d'un alphabet A_a avec $A \in L$ et a, la taille de A. Si le nombre de caractères de p est uniformément distribué, le nombre d'occurrences d'un caractère i quelconque de A est défini par la relation

$$n_i = \frac{n}{a} \tag{3.1}$$

Si est n est suffisamment grand, on peut supposer que $n(n-1) = n^2$ et de même pour n_i qui devient $\frac{n}{a}\left(\frac{n}{a}-1\right) = \left(\frac{n}{a}\right)^2$. Si A contient les caractères de l'alphabet courant, ce qui est le cas en général, on aura a=26. L'indice de coïncidence minimal du texte p sera défini par la relation suivante :

$$I_p = \frac{1}{n^2}\sum_{i=1}^{26}\left(\frac{n}{26}\right)^2 \sim \frac{1}{26} = 0.384 \tag{3.2}$$

Si, après une opération de cryptanalyse d'un texte c, cette valeur apparaît pour le texte résultat, c'est qu'il est dans l'une des situations suivantes : (Tilborg H. , 2005)

- Le texte initial p a été chiffré par la substitution monoalphabétique et la clé de déchiffrement utilisée est mauvaise car la fréquence d'apparition n'est pas la même pour tous les caractères : il fallait essayer d'autres clés.
- Le message a été chiffré par une substitution polyalphabétique.

b. Indice de coïncidence réel

En général, cet indice est dépendant du langage utilisé (telle la fréquence de caractères). Il sera donc défini par la relation $I_l = I_A + I_B + .. + I_Z$ où I_A est la probabilité de tirer aléatoirement deux 'A', I_B, la probabilité de tirer deux 'B' et de même pour les autres caractères de A.

Puisqu'il y a C_n^2 possibilités de tirer deux caractères identiques d'un texte de taille n, l'indice I_A du caractère 'A' sera donc :

$$I_A = \frac{C_{n_A}^2}{C_n^2} = \frac{\frac{n_A!}{2(n_A-2)!}}{\frac{n!}{2(n-2)!}} = \frac{n_A(n_A-1)}{n(n-1)} \tag{3.3}$$

L'indice du texte p contenant les 26 caractères de A devient :

$$I_p = \sum_{i=1}^{26} \frac{n_i(n_i-1)}{n(n-1)} \tag{3.4}$$

c. Test de Friedman

Similaire à la fréquence d'apparition de caractères qui varie d'un langage à un autre, l'indice de coïncidence I_L en est de même. Pour un texte assez grand, cet indice prendra une valeur donnée comme montré dans le tableau 3.7.

Allemand	0,0762	Français	0,0778
Anglais	0,0667	grec	0,0691
Arabe	0,0758	Russe	0,0529

Tab. 3.7. Indice de coïncidence de certains langages

L'idée de Friedman (Friedman W. , 1920), appelé également *test de Kappa* en référence à son aspect mathématique, repose sur ce principe : pour un chiffrement monoalphabétique, l'indice I_l propre à chaque langage ne change pas si on substitut un caractère par un autre caractère différent car chaque caractère est codé de la même manière. Il n'est pas de même quant au chiffrement polyalphabétique où il prendra une valeur minimale I_m largement inférieure à 0.04 car chaque portion du texte (égale à la longueur de la clé) est codée par un alphabet propre.

d. Longueur de la clé

En supposant que le chiffrement est polyalphabétique et, en plaçant le texte p dans un tableau de k colonnes, on aura bien entendu $\frac{n}{k}$ lignes. Chaque colonne correspond à une substitution simple par un caractère de la clé k.

La sélection de deux caractères identiques du texte peut être faite de deux façons différentes :

- Soient qu'elles se trouvent dans deux colonnes différentes, au nombre de

$$I_m C_k^2 \left(\frac{n}{k}\right)^2 = I_m \frac{k(k-1)}{2} \left(\frac{n}{k}\right)^2 = I_m \frac{(k-1)}{2k} n^2 \tag{3.5}$$

- Ou dans la même colonne, au nombre de

$$I_l C_{n/k}^2 = I_l \frac{n(n-k)}{2k} \tag{3.6}$$

L'indice réel du texte *p* aura pour équation

$$I_c = \frac{I_m \frac{(k-1)}{2k} n^2 + I_l \frac{n(n-k)}{2k}}{C_k^2} = \frac{I_m n(k-1) + I_l (n-k)}{k(n+1)} \tag{3.7}$$

Ce qui nous permet de déduire une valeur approximative de la clé

$$k = \frac{n(I_l - I_m)}{n(I_c - I_m) + (I_l - I_{cm})} \tag{3.8}$$

e. Exemple

Soit le passage suivant[27] :

"Certainly not; but if you observe, people always live forever when there is an annuity to be paid them; and she is very stout and healthy, and hardly forty. An annuity is a very serious business; it comes over and over every year, and there is no getting rid of it. You are not aware of what you are doing. I have known a great deal of the trouble of annuities; for my mother was clogged with the payment of three to old superannuated servants by my father's will, and it is amazing how disagreeable she found it. Twice every year these annuities were to be paid; and then there was the trouble of getting it to them; and then one of them was said to have died, and afterwards it turned out to be no such thing. My mother was quite sick of it. Her income was not her own, she said, with such perpetual claims on it; and it was the more unkind in my father, because, otherwise, the money would have been entirely at my mother's disposal, without any restriction whatever. It has given me such an abhorrence of annuities, that I am sure I would not pin myself down to the payment of one for all the world."

Ce texte ayant un indice de coïncidence égal à 0.066874, ce qui signifie qu'il est en anglais.

27. Source : Jane Austen, 'Sense and Sensibility', Novel 1811.

En chiffrant ce texte avec la substitution monoalphabétique avec la clé « azertyuiopqsdfghjklmwxcvbn », on aura le résultat chiffré suivant : (la ponctuation et espace ont été conservés pour de fins de clarté)

Etkmaofsb fgm; zwm oy bgw gzltkxt, htghst ascabl soxt ygktxtk citf mitkt ol af affwomb mg zt haor mitd; afr lit ol xtkb lmgwm afr itasmib, afr iakrsb ygkmb. Af affwomb ol a xtkb ltkogwl zwloftll; om egdtl gxtk afr gxtk txtkb btak, afr mitkt ol fg utmmofu kor gy om. Bgw akt fgm acakt gy ciam bgw akt rgofu. O iaxt qfgcf a uktam rtas gy mit mkgwzst gy affwomotl; ygk db dgmitk cal esguutr comi mit habdtfm gy miktt mg gsr lwhtkaffwamtr ltkxafml zb db yamitk'l coss, afr om ol adanofu igc rolaukttazst lit ygwfr om. Mcoet txtkb btak mitlt affwomotl ctkt mg zt haor; afr mitf mitkt cal mit mkgwzst gy utmmofu om mg mitd; afr mitf gft g y mitd cal laor mg iaxt rotr, afr aymtkcakrl om mwkftr gwm mg zt fg lwei miofu. Db dgmitk cal jwomt loeq gy om. Itk ofegdt cal fgm itk gcf, lit laor, comi lwei htkhtmwas esaodl gf om; afr om cal mit dgkt wfqofr of db yamitk, zteawlt, gmitkcolt, mit dgftb cgwsr iaxt zttf tfmoktsb am db dgmitk'l rolhglas, comigwm afb ktlmkoemogf ciamtxtk. Om ial uoxtf dt lwei af azigkktfet gy affwomotl, miam o ad lwkt o cgwsr fgm hof dbltsy rgcf mg mit habdtfm gy gft ygk ass mit cgksr.

Ce texte a un indice de coïncidence similaire texte clair correspondant : 0.066874.

En chiffrant ce texte avec la substitution polyalphabétique, clé « polycod », le résultat sera donc :

Rscrcwqam ymv; pxi wq wqi rqgppxs, stcajg oolojq nwyt tzpgjhg kscp hktfp gu oq pbyskhb ic mc rols hsco; oqs gsc kg ytfj qvcxi oyb jsdahsw, cbg wocbnm idfew. Cb dcbfgvm lh o gctm vtftmwg ejgtlggv; xh nmosv djpp cbg djpp gjhgm jccf, dcr efgfh xg ym iswiwye twg dt tr. Acx pfp lqh dlocc qt zwoe wqi dgs omkbj. X vltg yqdky y ifhph occz ru hsc vfrjpwc qt dcbfgvwhh; tzp om pdhsct kdh qwmiuhs ktrj hkt dlwosqi cq rjfht hz mnr vjdppcbqjoecf ghgjllvg en aj dchktf'd ukzo, pbo gv wv palxkbj wch bkgdvfpccpot gsc hcxcr tr. Vklrs ptgfb nslp vvhhs llpiliwpq ysut hz zg ddxr; llf hktb efgfh lod rjs wgcfzns ru uprvwqv we rq hkta; llf hktb zlg ci ivpk yov hotb vc kpjp bksg, pbo yhhhgklpfg li hfppsg die rq ph cc dsev wwwye. Om pdhsct kdh efgvs vxqv mh ww. Wsc gpqrbs hyu bri vpp qkq, hvp qcwg, lwef uifw dpprswjow anolbg zl kh; dcr tr yov ivp kqfh jbvgpr lc aj dchktf, mceoxhs, zrjsulwdc, vvh bcyca krjzo fcjh qspl gbwxfpja ow bm xmvvhg'g ogudrhow, ukhkdie ypm utgepkqwxcy ujowtjpp. Kh kpg rgxsq bs dsev dc omfqfutbnc qt dcbfgvwhh, hsyv W db gfpg W zdiwb pcw ewy kaghat omyb wd hsc robbsyr qt rcs qmt ooa hsc ycuar

Ce texte à un indice de coïncidence est égal à : 0.044344 , ce qui signifie qu'il est chiffré avec la substitution polyalphabétique.

f. Algorithme

Le pseudo-code illustrée par la fonction 3.1 utilise comme entrée un texte chiffré *Cipher* de taille *n*, composé de caractères appartenant à un alphabet *Alpha* (de taille 26). Ce texte est chiffré avec une clé *k* (désignée par sa taille) non connue mais ne dépassant pas *max_k*. Elle retourne l'indice de coïncidence du texte *Plain* et une valeur approximative de *k*.

La fonction détermine en une première phase le nombre d'occurrence *Num_occ* et l'indice de coïncidence *I* de chaque caractère texte *Cipher*. En une deuxième phase, elle calcule l'indice global du texte *Ic_ref* et par conséquent la longueur de la clé utilisée pour le chiffrement.

Fonction 3.1. Friedman_Test(Cipher, max_k, Alpha)

```
Ic_ref ← 0.038
Pour i ← 2 à max_k faire
    Pour j ← 1 à 26 faire
        Alpha[j].occurence ← 0
    FinPour
    Ic ← 0
    Num_occ ← 0
    Pour j ← 1 à n faire
        Alpha.[cipher[j]].occurence ← Alpha.[cipher[j]].occurence + 1
        Num_occ ← Num_occ+1
    FinPour
    Pour j ← 1 à 26 faire
        Si Alpha[j].occurence > 0
            Ic← Ic +Alpha[j].occurence * (Alpha[j].occurence-1)
        FinSi
    FinPour
    Ic ← Ic/ (Num_occ * (Num_occ-1))
    Si Ic > Ic_ref
        Ic_ref ← Ic
        k← i
    FinSi
FinPour
```

3.3.5 Test de Kasiski

a. Définition :

Au niveau d'un texte chiffré c de n caractères, soit une séquence t_p de p caractères de c qui apparaît plusieurs fois avec une distance multiple d'une constante d. Ce fait est dû à une des deux possibilités :

- La séquence provient de la même séquence du texte clair correspondant et bien entendu, chiffrée avec la même partie de la clé utilisée.
- La séquence pourrait, avec une faible probabilité, provenir de séquences différentes du texte clair et dont le chiffrement aurait été, par coïncidence le même.

Selon la première possibilité : si k est la longueur de la clé, avec ($p < k \leq d << n$) et pour que deux séquences soient codées avec la même partie de cette clé, il faut que k divise d. Le PGCD de k et d pourrait être la longueur de la clé considérée.

Chaque séquence t_{ik} de k caractères consécutifs (avec i=1, 2, .. , t/k) est en fait, une substitution monoalphabétique de clé k. Chaque caractère j (j=1, 2, .., k) de la clé a été utilisé pour chiffrer la séquence t^{j}_{ik} du texte t, avec i ∈ {1, 2, .., t/k-1}. En appliquant une analyse de fréquences à chaque séquence séparément, on pourrait déterminer le caractère correspondant de la clé.

L'idée de Kasiski repose sur ce principe : elle permet de rechercher les séquences identiques (de préférence : longs polygrammes) au sein d'un texte chiffré. Si trouvées, on relève les distances qui les séparent. Le diviseur commun de ces distances pourrait, dans la plupart des cas être la longueur de la clé de chiffrement.

b. Exemple :

Soit le texte suivant, résultat de chiffrement par substitution polyalphabétique du texte mentionné au § 3.3.4/e:

RSCRCWQAMYMVPXIWQWQIRQGPPXSSTCAJGOOLOJQNWYTTZPGJHGKSCPHKTFPGUOQPBYSKH
BICMCROLSHSCOOQSGSCKGYTFJQVCXIOYBJSDAHSWCBGWOCBNMIDFEWCBDCBFGVMLHOGCT
MVTFTMWGEJGTLGGVXHNMOSVDJPPCBGDJPPGJHGMJCCFDCREFGFHXGYMISWIWYETWGDTTR
ACXPFPLQHDLOCCQTZWOEWQIDGSOMKBJXVLTGYQDKYYIFHPHOCCZRUHSCVFRJPWCQTDCBF
GVWHHTZPOMPDHSCTKDHQWMIUHSKTRJHKTDLWOSQICQRJFHTHZMNRVJDPPCBQJOECFGHGJ
LLVGENAJDCHKTFDUKZOPBOGVWVPALXKBJWCHBKGDVFPCCPOTGSCHCXCRTRVKLRSPTGFBN
SLPVVHHSLLPILIWPQYSUTHZZGDDXRLLFHKTBEFGFHLODRJSWGCFZNSRUUPRVWQVWERQHK
TALLFHKTBZLGCIIVPKYOVHOTBVCKPJPBKSGPBOYHHHGKLPFGLIHFPPSGDIERQPHCCDSEV
WWWYEOMPDHSCTKDHEFGVSVXQVMHWWWSCGPQRBSHYUBRIVPPQKQHVPQCWGLWEFUIFWDPPR
SWJOWANOLBGZLKHDCRTRYOVIVPKQFHJBVGPRLCAJDCHKTFMCEOXHSZRJSULWDCVVHBCYC
AKRJZOFCJHQSPLGBWXFPJAOWBMXMVVHGGOGUDRHOWUKHKDIEYPMUTGEPKQWXCYUJOWTJP
PKHKPGRGXSQBSDSEVDCOMFQFUTBNCQTDCBFGVWHHHSYVWDBGFPGWZDIWBPCWEWYKAGHAT
OMYBWDHSCROBBSYRQTRCSQMTOOAHSCYCUAR

Les séquences fréquentes indiquées en couleur dans ce passage sont illustrées dans le tableau 3.8.

Séquence	Position	Décomposition	Séquence	Position	Décomposition
HK	55	7^2+6	LL	345	7^3+3
	307	$43x7+6$		423	$7x5x3^2x2^2$
	356	$7x5^2x2^2+3$		444	7^2x3^3
	447	7^2x3^3+2		486	$23x7x3^2$
	482	$17x7x2^3+3$	DI	540	$7x7x5x2+1$
	664	$23x7x2^2+3$		736	$7x7x3x5+1$
	734	$13x7x2^4+3$		813	$53x7x2+1$
	762	$13x7x2^3$			
HS	79	$11x7+2$	CB	108	$7x5x3^2$
	107	$7x5x3+2$		115	$7x2^4+3$
	219	$31x7+2$		123	$17x7+4$
	421	$7x5x3x2^2+1$		127	$7x3^2x2+1$
	562	$7x5x2^4$		166	$23x7+5$
	673	$7x3x2^5+1$		274	$13x7x3+1$
	807	$7^2x5x3+2$		334	$47x7+5$
	835	$109x7+2$		793	$113x7+2$
	856	$61x7x2^2$			

Tab 3.8. Facteurs de décomposition des séquences identiques

Les facteurs communs les plus fréquents sont 2, 3, 5, 7 et 13. Selon Kasiski, chaque facteur doit être de la forme $nk+d$ avec k, la taille de la clé. Les valeurs de d sont 1, 2, 3, 5 et 6. On

remarque que seul le facteur commun, supérieur à *d* qui apparaît dans toutes les décompositions est 7. Il doit être probablement la clé de chiffrement.

Effectivement, la clé de chiffrement est le mot « POLYCOD », de taille 7, mais ce n'est pas toujours le cas.

c. Algorithme

Le test de Kasiski prend en entrée un texte chiffré *Cipher* de taille *n* et une estimation de la longueur maximale de la clé utilisée *max_k*.

Function 3.2. KasiskiTest(Cipher, max_k)

```
j, d ← 0
Pour i ← 0 à n faire
      j ← i + 1
     TantQue (j < n) faire
          taille_seq ← 0
          TantQue ( Cipher[ seq + i] = Cipher[seq + j] faire
                Seq[taille_seq] ← cipher[j + taille_seq]
                taille_seq ←  taille_seq + 1
          FinTantQue
          Si taille_seq ≥ max_k
             distance[d] ← j – i
             d← d + 1
          FinSi
             j← j + 1
     FinTantQue
FinPour
max_k ← gcd(distance)
```

Le principe de l'algorithme consiste à déterminer les séquences *seq* identiques de taille *taille_seq* séparées par une distance *j-i*. La table *distance* contient l'ensemble des distances qui séparent les différentes séquences identiques rencontrées. Le *gcd* des éléments de cette table désigne la longueur probable de la clé de chiffrement.

3.4 Techniques de cryptanalyse des chiffrements symétriques

3.4.1 Cryptanalyse linéaire

Fût proposée par (Matsui, 1993) et destinée à casser les chiffrements symétriques, la cryptanalyse linéaire est une attaque à texte clair connu et consiste en l'analyse de l'évolution de la différence basée sur l'approximation linéaire pour divers paires de textes clairs et chiffrés dans le but d'annuler la non-linéarité des Sboxes et par conséquent, déterminer la clé du chiffrement.

Comme la plupart des attaques aux chiffrements par blocs, la cryptanalyse linéaire cible le dernier tour de l'algorithme de chiffrement et tente de déterminer un ensemble de positions de

bits $i_1, i_2, .., i_m$ du texte clair et un autre ensemble de positions $i_1, i_2, .., i_n$ des bits du texte chiffré (avec m pas forcément égal à n) tel que la somme de chaque série de bits prend la même valeur pour la plupart des textes clairs utilisés. Autrement dit,

$$x[i_1] \oplus x[i_2] \oplus ... \oplus x[i_m] \oplus x_r[j_1] \oplus x_r[j_2] \oplus ... \oplus x_r[j_n] = \alpha \quad (3.9)$$

avec x_r le résultat du chiffrement du dernier tour.

Pour un petit nombre de couples (x, x_r), α serait nulle. Pour un chiffrement non linéaire, elle serait de l'ordre de ½. En augmentant le nombre de couples, α progresse vers le 1. Si on note p, la probabilité que l'égalité (3.9) soit satisfaite, le nombre de couples nécessaire à cette condition serait de l'ordre inverse de la relation $(p-½.)^2$. La meilleure approximation connue est satisfaite avec une probabilité de l'ordre de $½+2^{-22}$, nécessitant la connaissance de 2^{43} couples de textes clairs/chiffrés.

3.4.2 Cryptanalyse différentielle

Il s'agit d'une attaque basée sur des données claires choisies. L'idée de la cryptanalyse différentielle revient à (Biham & Shamir, 1991) et repose sur des paires de textes clairs ayant une différence constante α obtenue à l'aide de l'opérateur XOR.

En posant $x_0 = x$ et $x_i = f(x_{i-1}, k_i)$ pour tout i=1 à r, avec r, le nombre de tours du chiffrement utilisé ; L'attaque s'intéresse à l'examen de la différence $x_i + x'_i$ pour toute paire de textes clairs x et x' et celle des chiffrements correspondants en sortie $\beta = x_{r-1} + x'_{r-1}$ et tente d'extraire des motifs qui peuvent renseigner sur les propriétés des S-boxes utilisées.

En notant la probabilité $P_{\alpha,\beta} = P(x_{r-1} + x'_{r-1} = \beta / x_i + x_i' = \alpha)$, l'attaque cherche à trouver une valeur du couple (α, β) permettant de maximiser P. Cette idée permet de déterminer k_r à partir de plusieurs messages x et leurs chiffrements x_r. Pour ceci, on itère pour un grand nombre de paires (x, x_r) et (x', x'_r) le pseudo-code suivant :

Fonction 3.3. Max_P(k, β)

Pour toute valeur de k_r faire
  Calculer $x_{r-1} = f^{-1}(x_r, k_r)$ et $x'_{r-1} = f^{-1}(x'_r, k_r)$
  Si
    $x_{r-1} + x'_{r-1} = \beta$, incrémenter un compteur relatif à k_r.
  FinSi
FinPour

Le DES est résistant à la cryptanalyse différentielle, néanmoins, les versions allégées (de moins de 10 tours) sont vulnérables à ce type d'attaque. Un chiffrement à 8 tours, nécessite seulement 2^{38} textes choisis.

3.4.3 Attaque homme-du-milieu

L'attaque MITM, abréviation de *man-in-the-middle* est une méthode permettant à l'attaquant de se placer entre deux utilisateurs et pouvoir intercepter leurs échanges de données. L'acte devient dangereux si l'attaquant se permet de modifier les informations des utilisateurs en se faisant passer pour l'un d'eux aux yeux de l'autre.

En cryptographie symétrique, le problème réside au niveau de l'échange de clés interceptées par l'attaquant. L'attaque est basée sur l'observation suivante :

$$\text{Si } c = DES_{k_2}(DES_{k_1}(p)) \text{ alors } x = DES_{k_1}(p) = DES_{k_2}^{-1}(c) \tag{3.10}$$

Ainsi et, pour une paire de textes clair/chiffré (p,c), on procède au chiffrement de p avec divers valeurs de k_1 produisant ainsi des textes chiffrés c qui seront stockés dans une table de couples $(k_1, DES_{k1}(p))$. La deuxième partie de l'attaque consiste à déchiffrer le crypté c avec divers valeurs de k_2 afin de déterminer x. L'idée de l'attaque est de chercher si par hasard un des résultats x pourrait se trouver parmi les valeurs stockées dans le tableau généré précédemment. Si c'est le cas, la clé k_2 est dévoilée. Dans ce cas, le résultat x va être encore déchiffré par la même manière dans le but de déterminer k_1. Dans le cas contraire on continue à tenter d'avantage de clés.

L'attaque de l'homme-du-milieu est utilisée pour intercepter et dérouter les clés relevant de la cryptographie asymétrique.

3.5 Autres attaques

3.5.1 Cryptanalyse par les heuristiques

Les techniques d'optimisation étaient toujours considérés comme un outil efficace et intelligent pour la résolution de problèmes combinatoires de taille importante moyennant une consommation raisonnable de ressources (Yean Li, Samsudin, & Belaton, 2005) (Blum & Li, 2008) (Dadhich, Gupta, & Yadav, 2014). Les metaheuristiques, branche des techniques d'optimisation inspirées du comportement de phénomènes naturels présentent un potentiel de traitement des instances complexes offrant ainsi des solutions approximatives acceptables. La cryptanalyse de chiffrements, un problème classé difficile était dans le temps, objet de différentes attaques basées sur diverses metaheuristiques. Néanmoins et, à cause de l'environnement de l'implémentation nécessitant un paramétrage délicat, les résultats étaient modestes et divergentes.

Le détail des travaux importants dans ce domaine seront discutés dans les chapitres suivants.

3.5.2 Recherche exhaustive de la clé :

La technique consiste simplement à essayer toutes les combinaisons de clés possibles jusqu'à trouver la bonne. Théoriquement, la technique fonctionne à tous les coups, néanmoins, elle consomme plus de ressources que toute autre technique similaire[28]. Evidemment et, à cause des restrictions matérielles, la recherche est envisageable quant aux chiffrements ayant un nombre

28. Lors d'une substitution simple à 26 caractères, on a 26 ! dés différentes ; soit un nombre n de possibilités de l'ordre de 403.291.461.126.605.635.584.000.000. Sur une machine qui exécute 10^9 possibilités par seconde, on accompli le test dans un laps de temps qui s'élève à $n/(365\times24\times3600\times10^9)$. Soit 1.2788×10^{10} années.

restreint de possibilités tels les chiffrements classiques. On comprend cependant l'intérêt d'utiliser des algorithmes de chiffrement complexes et les clés d'une certaine taille afin de pouvoir résister à une telle attaque.

Les chiffrements symétriques utilisent des clés de 56 bits minimum ; l'espace de recherche est donc de l'ordre de 2^{63} ce qui leur assure un certain niveau de sécurité. A vu la puissance de calcul actuelle, un espace clés de confiance serait de l'ordre de 2^{80} ; on recommande cependant des clés à 128 bits pour une sécurité à long terme.

L'attaque exhaustive du chiffrement DES[29] a été accomplie en janvier 1998 en 39 jours sur un réseau de 104 machines couplées, puis en 56 heures en juillet 1998 à l'aide d'une machine dédiée.

De telles attaques n'ont jamais été réalisées avec succès sur des chiffrements de 128 bits.

3.6 Mesure de robustesse d'un chiffrement

La sécurité d'un chiffrement est liée directement à la clé qui n'est connue que par les personnes habilitées tandis que l'algorithme de chiffrement est supposé connu. Ce principe est connu sous le nom de *principe de Kerckhoffs* (Kerckhoffs, 1883). Donc, une attaque réussie envers un chiffrement permet de retrouver la clé du chiffrement et, par la suite, dévoiler le texte clair. Un attaquant doit avoir à disposition certains échantillons de textes chiffrés et, dans la mesure du possible une idée quant à l'algorithme de chiffrement et la clé afin de pouvoir générer des textes clairs. La performance d'une attaque est une fonction inverse de la robustesse du système de chiffrement et, évaluée selon les critères suivants :

- *Complexité de donnés* : taille et type de données nécessaires à l'attaque,
- *Complexité en temps* : volume temporaire de traitement, calculé à base d'opérations élémentaire par unité de temps,
- *Complexité en mémoire* : espace de ressources nécessaire à l'attaque,
- *Coût* : qualité du résultat obtenu à la condition d'arrêt fixé.

Un chiffrement est considéré robuste si l'espace de clés est relativement vaste. Il devait produire des cryptogrammes jugés aléatoires dans le sens où ils ne peuvent être modélisés par des procédés mathématiques. Un tel cryptosystème doit résister à toutes les formes attaques ainsi énumérées, ce qui est difficile à prouver dans le sens où une recherche aléatoire utilisant la stratégie de *l'homme-du-milieu* peut aurait par chance, une possibilité de trouver la bonne clé. De même, l'attaque par *mot probable* peut également réussir si on a suffisamment d'informations sur les habitudes du propriétaire des chiffrements. Donc, un cryptosystème jugé incassable est simplement théorique. En pratique, la disponibilité de divers paramètres entourant l'environnement de chiffrement peuvent fausser cette règle.

Comme remède à cette situation, le chiffrement par *masque jetable* apparaît le plus sûr dans tous les temps car chaque texte possède sa propre clé de même taille et le chiffrement n'est autre qu'une somme exclusive entre le texte et la clé. Il est donc facile de prouver mathématiquement qu'il n'existe aucune attaque fructueuse à un tel cryptosystème si les clés utilisées sont distribuées uniformément. Cet acte parait difficilement réalisable en pratique dû à la manière de l'échange de

29. http://www.eff.org/descracker.html

clés entre émetteur et destinataire ; jusqu'à lors, la sécurité prouvée ou, en autres termes, la garantie de l'absence de failles d'un système de chiffrement demeure incertaine.

3.7 Conclusion

La cryptanalyse est un procédé d'exploration basé sur des outils mathématiques, hasardeux en partie, permettant d'obtenir de portions significatives d'un texte chiffré. Vu sa difficulté, un tel traitement est confié dans la plupart des cas, à la puissance de calcul. Le résultat attendu n'est jamais fiable en entier où une intervention manuelle ou semi-automatique doit être opérée afin de pouvoir assimiler le sens du texte chiffré sans le découvrir en entier. Donc, tout processus d'investigation en ce sens n'aboutit jamais à sa fin, il sera interrompu dès que le résultat commence à s'éclaircir.

Malgré son efficacité, la cryptanalyse par analyse de fréquences ne fût mise en œuvre que huit siècles après son apparition, lorsque la cryptographie commençait à être florissante aidée par le développement des outils de calcul. Sa première victime était peut être Marie 1 ere d'Ecosse.[30]

La cryptanalyse des chiffrements symétrique quant à elle, demeure toujours incertaine. Les résultats obtenus sont relativement restreints et limités aux clés de moins de 64 bits malgré l'abondance des techniques d'attaques qui dépassent les méthodes de chiffrements connues.

Ainsi, ce chapitre présente un survol des célèbres techniques de cryptanalyse classiques et modernes, leurs caractéristiques et leurs limites. On a remarqué que la plupart d'entre elles sont dédiées. Cela nous a permis de préciser le contexte de notre contribution qui s'articule autour des attaques par les techniques heuristiques, méthodes jugées plus générales et peuvent être adaptées à n'importe quel chiffrement.

30. Dans le but d'assassiner sa cousine, la reine Elisabeth 1ère , Sir Antony Babington et John Ballard, un prêtre, ont échangé les messages chiffrés avec Marie Stuart, reine d'Ecosse, afin de l'appuyer à l'accession au trône avec l'aval de Pkilippe II, le roi d'Espagne. Sir Francis Walsingham, secrétaire de Elisabeth 1ère a réussi à intercepter les correspondances, Thomas Phelipes, agent d'espionnage a pu les déchiffrer. Marie Stuart a été condamnée en octobre 1586 et exécutée en fevrier 1587. Affaire connue sous le nom de *complot de Babington*. Le chiffrement utilisé est une substitution simple contenant des chiffres et certains caractères spéciaux.

4. Heuristiques d'optimisation combinatoire

4.1 Introduction

La nature, notamment son évolution était toujours un guide visuel pour l'homme. Il s'est inspiré depuis ses ressources en matière d'expérience lui permettant d'améliorer sa vie. Elle lui a permis également de développer cette expérience et de la modéliser en paradigmes utilisés comme outils de résolution de problèmes combinatoires divers.

En informatique, cette idée a permis de développer certaines disciplines telles l'intelligence artificielle, les réseaux de neurones, la théorie des jeux et d'inventer des techniques d'optimisation qui, par leur conception, ayant permis de concevoir des outils d'implémentation de raisonnement logique formel, basés sur l'apprentissage et l'évolution et, indépendants des idées basiques du concepteur.

4.2 Techniques d'optimisation

Les techniques d'optimisation englobent l'ensemble des méthodes permettant d'orienter et de cadrer la recherche d'une solution dans un espace d'exploration défini, relatif à un problème donné et ce, avec ou sans contraintes. Elles occupent une place importante en recherche opérationnelle, programmation linéaire et dans divers domaines informatiques.

Les problèmes qui relèvent de l'optimisation sont appelés problèmes combinatoires dont la majorité sont à caractère économique, industriels ou de nature combinatoire, c'est-à-dire qu'ils englobent un ensemble d'instances, souvent nombreuses et que leurs solutions ne peuvent êtres exactes ou même si c'est le cas, ne peuvent être attestées et ce, en absence de référence confirmant cet effet : c'est le cas d'optimisation *déterministe*.

Un problème d'optimisation est défini souvent par un espace de solutions S et une fonction objectif f. Le but de l'exploration est de localiser $s^* \in S$ possédant la meilleure qualité en regard

de la fonction *f*. Selon la nature du problème cette fonction peut être maximisée ou minimisée. Dans la suite de ce mémoire, on aborde le cas de la minimisation (qui peut être inversée dans le cas de la maximisation) et formulé par l'équation suivante :

$$s^* = \min(f(s) \backslash s \in S) \tag{4.1}$$

Dans d'autres cas, on se demande souvent quel volume de ressources, notamment en temps de calcul peut être alloué à résoudre certaines instances d'un tel problème. Dans divers cas, on se trouve face à des situations où il paraît impossible de l'accomplir dans un délai raisonnable : on parle cependant d'optimisation *difficile*, c'est le cas où le nombre de combinaisons devient exponentiel par rapport à la taille du problème. Dans ce cas, la recherche exhaustive de solutions est donc impensable en raison du temps de calcul induit. Pour un problème de voyageur de commerce, l'espace de recherche croit en (n-1) ! avec n : le nombre de villes à visiter. Si le nombre de villes est 20 par exemple, il faudra explorer 19 ! Chemins, soit 1.2×10^{17}, ce qui est acceptable à être résolu avec une méthode traditionnelle *exacte* telle le *Recuit Simulé*, *Séparation et évaluation* ou la solution optimale est garantie. Dans un problème de cryptanalyse de chiffrement par substitution classique par exemple, le nombre d'instance que peut prendre une clé d'essai s'élève à 26 !, soit approximativement 4×10^{26}. Un tel nombre d'instances ne peut être accompli en une période d'exécution raisonnable en utilisant une méthode exacte[1]. On se contente cependant de solutions *approximatives* sans garantie de la qualité et ce, au dépens du temps de calcul. La dite solution est fournie par une méthode *rapprochée* ou heuristique adaptée au problème considéré.

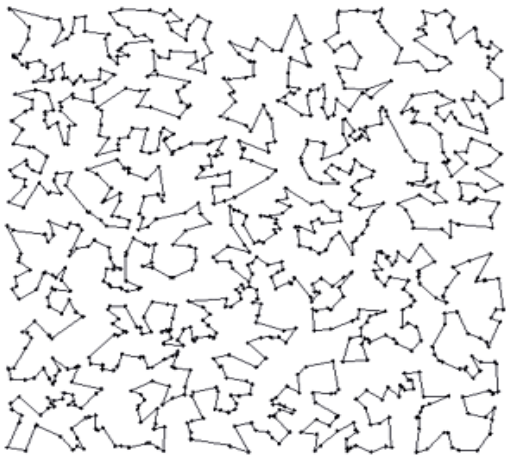

Fig. 4.1 : Représentation du chemin le plus court passant par une distribution aléatoire de 1000 points.

4.3 Heuristiques et metaheuristiques

Une heuristique est une stratégie qui améliore l'efficacité de recherche d'une solution aux dépens éventuellement l'exactitude ou l'optimalité du résultat. Elle emploie un processus d'exploitation aléatoire afin de réduire le temps de calcul engendré par l'utilisation des méthodes

1. A raison de 1 microseconde par instance, l'algorithme prendra plusieurs années sachant qu'une année compte $3{,}16 \times 10^{13}$ microsecondes.

exactes. On se contente de l'usage d'une telle technique pour la résolution des problèmes *NP-Complets* où la recherche d'une solution exacte nécessite un coût exponentiel et, où l'usage de la force brute s'avère sans succès notamment pour les grandes instances.

Le terme heuristique remonte aux années 60. Il a été, selon (Newell, Shaw, & Simon, 1957) "U*n processus heuristique peut résoudre un problème donné, mais n'offre pas la garantie de le faire*". (Feigenbaum & Feldman, 1963) définissent une heuristique comme étant "*une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre sorte de système qui limite drastiquement la recherche des solutions dans l'espace des configurations possibles*".

Le terme metaheuristique est utilisé par opposition aux heuristiques dédiées à un problème spécifique ; il désigne les heuristiques ayant des caractéristiques polyvalentes et peuvent être adaptées à résoudre des problèmes à variables discrètes en premier temps, puis généralisées par la suite pour être adaptées aux problèmes de natures diverses et à variables continues. D'un autre côté, les metaheuristiques n'offrent à leur tour qu'une solution sans garantie d'optimalité, ceci et, du point de vue recherche opérationnelle, n'est pas un désavantage car il n'existe pas d'alternative plus efficace en matière de temps de traitement autre que les méthodes exactes. D'où on préfèrera souvent un optimum global en un temps raisonnable qu'une solution exacte en temps rédhibitoire.

Le terme *métaheuristiques*[2] a fait son apparence dans les années 80, (GLover, 1986) présenta sa définition du terme : metaheuristique "*se réfère à une stratégie maître qui guide et modifie d'autres heuristiques pour produire des solutions au-delà de ceux qui sont normalement générés dans une quête d'optimalité locale*". D'autres définitions plus adaptées ont été également énoncées : "*recherche intelligente*" (Pear, 1996), "*heuristique dans un contexte d'intelligence artificielle* " (Vob, 1993) ou comme "*une stratégie d'apprentissage*" (Osman & Kelly, 1996). Le réseau Metaheuristics[3] a défini ce terme comme étant " *un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliqués à une grande variété de problèmes. On peut voir la metaheuristique comme une boîte à outils algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier* "

Les metaheuristiques sont inspirées en général, par l'évolution des phénomènes naturels biologiques, physiques ou éthologiques. Elles ont un caractère stochastique afin de surmonter l'explosion combinatoire, leur performance dépend de la formulation de la fonction objectif et de la manière de choix des paramètres utilisés. Leur efficacité est déterminée en fonction du temps de calcul, de la puissance des machines et de la qualité de la solution obtenue. D'autre part, les metaheuristiques sont souvent le seul moyen de traitement grandes instances relatifs aux problèmes combinatoires. En plus, elles sont souvent itératives réduisant ainsi la complexité algorithmiques et, ont également la faculté, moyennant quelques modifications, d'éviter les minimax locaux en cas de dégradation de la fonction objectif au cours du traitement.

Dans la littérature, on rencontre de nombreuses techniques d'optimisation déterministes pouvant être classées en deux grandes catégories : méthodes de *recherche locale* et *metaheuristiques à population*. Ces techniques partent toujours d'une solution basique déterminée aléatoirement ou par un échantillonnage de l'espace de recherche puis, explorer le voisinage pour la première catégorie ou engager plusieurs solutions en parallèle pour la deuxième. Concernant leur limite, les

2. Du Grec, *meta* : au-delà, *heuriskein* : recherche ; signifie : un niveau abstrait de la recherche.

3. Metaheuristics Network: www.metaheuristics.org

metaheuristiques partagent les mêmes inconvénients, notamment la difficulté d'ajustement des paramètres de fonctionnement et de la consommation élevée de ressources. Leur efficacité dépend de de la taille de l'échantillonnage ainsi que de la forme de la fonction objectif.

4.4 Historique:

L'introduction du principe de l'évolution dans la littérature remonte à un demi-siècle environs avec la naissance de deux metaheuristiques : la *stratégie d'évolution* (Rechenberg, 1965) et la *programmation évolution*naire (Fogel, Owens, & Walsh, 1966) ; mais ce n'est qu'une dizaine d'années après que ce mécanisme a été mis en pratique pour la résolution de problèmes réels car, c'était la naissance des machines de calcul puissantes. Cet acte a amorcé une évolution en boule de neige dans le domaine des heuristiques par la naissance de nouvelles versions ou d'hybridation entre des techniques existante. La chronologie des principales metaheuristiques a été marquée par l'apparition des *algorithmes génétiques* (Holland J. H., 1975). Une dizaine d'années après, c'était le tour des méthodes classiques : Le *recuit simulé* (Kirkpatrick, Gelatt, & Vecch, 1983) et la *recherche tabou* (GLover, 1986). La littérature a été ensuite, enrichie par une série de nouvelles variantes : *colonies de fourmis* (Dorigo M. , 1992), les *essaims particulaires* (Eberhart R. C., 1995), l'*estimation de distribution* (Mülhenbein & Paaß, 1996), *l'évolution différentielle* (Storn & Price, 1997). Ensuite, la recherche s'est penchée vers l'hybridation et l'extension des techniques existantes notamment les algorithmes génétiques interactifs et les méthodes inspirées du comportement de chasse des animaux vivant en collectivité : *Bat Algorithm* (Yang, 2010), *Penguins Algorithm* (Gheraibia & Moussaoui, 2013), *Wolf Pack Algorithm* (Wu & Zhang, 2014).

4.5 Caractéristiques des metaheuristiques

4.5.1 Voisinage

Soit *S*, l'ensemble de solutions à un problème combinatoire. La structure de *voisinage* d'une solution est une fonction V définie par (Blaum & Roli, 2003) :

$$V : S \rightarrow 2^{s} \text{ qui associe à chaque } s \in S \text{ un ensemble de voisins } V(s) \subset S.$$

V(s) est appelé voisinage de *s*.; désigne l'ensemble de solutions où chacune d'elles est obtenue par une *transformation* élémentaire permettant de changer une solution *s* en une autre solution *s'* à l'aide de modifications de paramètres de l'environnement. Cependant, les solutions ainsi énumérées sont dites *voisines*. La transformation est dite *locale* si la différence entre *s* et *s'* est faible. Si *s* est une chaine de caractères par exemple, *s'* devait être la même chaine avec un seul caractère de changé par rapport à *s*. Une telle transformation doit vérifier deux critères :

- La *réversibilité* : engendrer un ou plusieurs points de retour permettant de retrouver le schéma précèdent de *s* au cas où la solution s'avère insatisfaisante.
- *L'accessibilité* : d'engendrer un effet sur tout l'ensemble V(s) ou du moins, sur un sous-ensemble de V(s) ayant une influence sur la qualité de la solution.

Ce dernier critère nécessite bien entendu, une mémoire d'archive permettant de retracer l'état du parcours sur un nombre d'étapes déterminé.

Pour définir un voisinage, on a recours en général à certaines opérations mathématiques élémentaires selon la nature du problème et le codage de la solution, à savoir :

- *Complémentation* : remplacer un élément de la solution par son complément : 0 par 1 si le codage est binaire par exemple, ou un caractère par un autre faisant partie de l'ensemble des caractères utilisés si le cardinal de cet ensemble est supérieur à 2. Dans ce cas, il faut prendre en compte le nombre d'occurrence de chaque caractère si nécessaire. Dans un problème de cryptanalyse par substitution par exemple, la clé ne doit pas comporter de répétition de caractères où la complémentation se confond avec l'échange.
- *Echange* : la manière d'intervertir deux caractères se trouvant à de positions distinctes données.
- *Insertion-décalage* : consiste à un décalage d'une position d'une sous-chaine de la solution entre deux positions *i* et *j* et, où le caractère *j* remplacera le caractère *i*. Ce type de transformation peut perturber la qualité de la solution dans certains cas et sortir du cadre du local au global car son effet s'étend à une large portion de la chaine représentant la solution.

En pratique, la manière du choix du voisinage s'inspire de la structure du problème considéré et de la qualité du résultat notamment en matière de diversité. Son avantage réside dans la possibilité de contrôler le processus d'orientation vers l'objectif qui doit s'opérer en un temps polynomial de préférence inférieur à ($O(n^3)$) car, si ce n'est pas le cas, on a tendance à utiliser une force brute qui gaspille plus de ressources inutiles. Pour un problème continu, le terme voisinage n'a pas de sens, tandis que si le problème est discret, le choix du voisinage a un effet majeur sur la qualité de la solution.

4.5.2 Exploration

La manière de l'exploration efficace de l'espace doit être renseignée par les diverses contraintes du problème qui permettent, entre autres, d'éviter le blocage en optima locaux et d'avoir un instinct raisonnable quant au choix de la meilleure direction parmi les instances proposées. Cette direction dit être renseignée, entre autres par :

- L'ensemble des données ou informations relatives au problème considéré : on parle cependant de *diversification* qui a pour objectif d'orienter l'exploration vers des zones inexplorées jugées prometteuses. Cette stratégie peut altérer la qualité du résultat mais, favorise d'un autre coté le surpassement des minima locaux et pénalise les mouvements souvent sollicités.
- L'*intensification,* ou la manière d'exploration visant des régions intéressantes au sein de l'espace de recherche et ce, à base de l'information déjà récoltée. Donc, elle se fonde sur l'idée d'apprentissage des propriétés favorables rencontrées au sein de l'exploration

- La *mémoire*, un *support* d'apprentissage aussi important que les données du problème car, il permet à l'algorithme de ne retenir que les zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux et surtout économiser les ressources de traitement.

La manière de choix de quelle stratégie à adopter à chaque situation parait difficile à définir dû à la spécificité de chaque problème traité et notamment en absence de modèles généraux. Cependant l'exploration doit prendre en compte les diverses stratégies disponibles en alternant des phases d'intensification, de diversification et d'apprentissage, ou en mêlant ces notions de façon plus étroite durant tout le processus de traitement.

4.5.3 Classification

Etant donné qu'elles ne peuvent, à priori être spécifiques à tel ou autre problème, les metaheuristiques ont été classées de différentes manières et ce, au dépens de leurs caractéristiques telles la stratégie de l'exploration : si elle est globale ou locale, le type de solution : si elle est exacte ou rapprochée, le type de parcours : direct, explicite ou évolutif, etc. D'une autre manière plus intuitive, la classification consiste à distinguer les metaheuristiques à population, c'est-à-dire, inspirés d'un phénomène naturel qui manipulent plusieurs parcours en parallèle de celles à parcours unique, ou les distinguer par rapport à l'usage d'une fonction objectif stable durant tout le processus du traitement ou d'une fonction objectif dynamique qui change de topologie selon les informations collectées au cours de l'exploration. Une autre alternative consiste à distinguer les metaheuristiques qui fonctionnent sans mémoire de celles ayant la faculté de mémoriser des informations récoltées durant le cycle de l'exploration et qui peuvent cependant retracer des solutions déjà examinées. Pour ce dernier cas, on distingue également la mémoire à court terme contenant l'archive les derniers mouvements effectués de la mémoire globale contenant des paramètres synthétiques plus généraux. En littérature, on rencontre d'autres classifications dites empiriques qui consistent à mesurer les metaheuristiques quant à leur spécificité, notamment leur complexité, la qualité de la solution produite ou par leur variété et leur capacité à résoudre certains types de problèmes.

En général, les metaheuristiques sont assez facilement surjetées à des extensions ou construites à base d'autres metaheuristiques spécifiques, donc ne peuvent être associées à une classification figée. On rencontre dans cette catégorie l'optimisation multimodale (Goldberg & Richardson, 1987) où le résultat englobe un ensemble d'optima locaux, la combinaison hybride (Talbi, 2002) construite à base de différentes metaheuristiques afin d'en bénéficier de leurs avantages respectifs ou les metaheuristiques parallèles (Alba, 2005) pour lesquelles le traitement est réparti sur plusieurs calculateurs.

Malgré les diverses approches considérées, certaines techniques peuvent être intégrées dans plus d'une catégorie d'où la classification demeure incertaine dans la plupart des cas comme mentionné dans la figure 4.2.

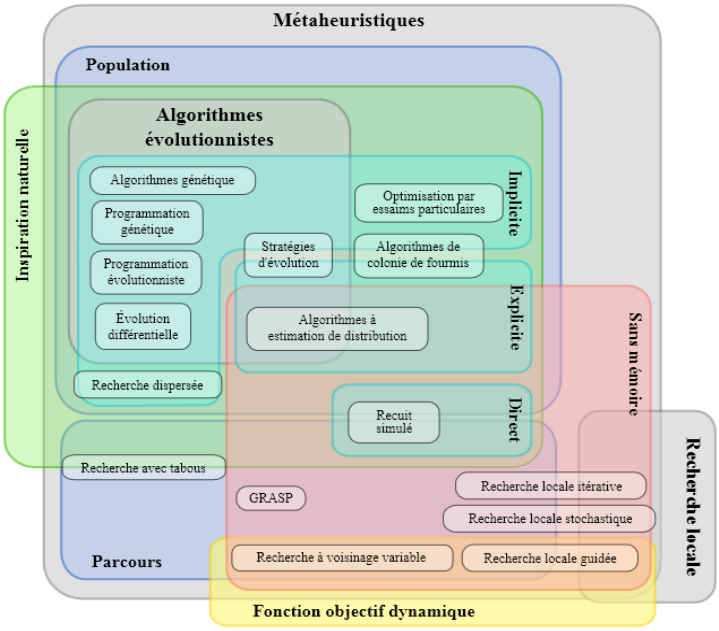

Fig. 4.2 : Classification des metaheuristiques[4]

Cependant, la classification la plus plausible à notre point de vue consiste à distinguer les metaheuristiques à base de leurs stratégies de construction de parcours. On en dénombre dans ce cas :

- Classe des méthodes de *recherche locale* (ou d'améliorations itératives) où la construction du parcours s'opère par des modifications locales tout au long du processus d'exploration. Elle englobe les techniques de descente, Recuit simulé, tabou, .. avec une fonction objectif stable ;
- Classe des méthodes de *construction* utilisant l'instanciation successive des variables selon un ordre statique ou dynamique dans un espace en structure d'arbre. On dénombre dans cette classe les méthodes séparation et branchement, les algorithmes gloutons, .. utilisant une fonction objectif dynamique et une complexité exponentielle dans le pire des cas ;
- Classe des méthodes à *population* (ou évolutives) utilisant l'intelligence collective des individus qui évoluent par des transformations génétiques.

La suite de ce chapitre est dédiée à la présentation de ces méthodes.

4.6 Metaheuristiques de recherche locale

4.6.1 Définition

Appelées également méthodes d'*amélioration itératives* ou méthodes de *descente*, les metaheuristiques de recherche locale consistent à déterminer d'une manière itérative la solution d'un problème d'optimisation en utilisant des outils mathématiques, logiques ou autres

4. source : http://commons.wikimedia.org/wiki/File:Metaheuristics_dassification_fr.svg?uselang=fr

conformément à la nature de la fonction objectif du problème considéré. Leur principe consiste à examiner un ensemble de solutions candidates S et retenir seulement celle considérée temporairement comme optimale et ce dans les limites des ressources considérées comme illustré par le pseudo-code de l'algorithme 4.1

Algorithme 4.1. Recherche locale

Entrée: S, f : resp. espace de recherche et fonction objectif,
s : Solution initiale ($s \in S$)
Sortie: s^*: solution optimale
Initialisation :
$s^* \leftarrow s$
Répéter
$s \leftarrow V(s)$
Si $f(s) < f(s^*)$ **alors** $s^* \leftarrow s$
Jusqu'à (*critère d'arrêt)*

Concrètement et, à partir d'une solution initiale s_0, généralement aléatoire ou résultat d'un autre algorithme, engendrer une suite finie de solutions $s_1, s_2, .., s_n$ obtenues itérativement de proche en proche. Le choix de chaque solution s_i à partir de s_{i-1} se fait de manière à ce que s_i soit meilleure que s_{i-1} en regard de la fonction objectif globale. La construction de chaque solution obéit à une stratégie adéquate quant au choix du bon voisinage.

La condition d'arrêt d'une recherche locale peut être un nombre fixe d'itérations ou une période d'exécution déterminée. Néanmoins, ce type d'algorithmes est dit *sous-optimal* dans la mesure où la meilleure solution peut se trouver loin du voisinage des solutions examinées et que l'arrêt peut être provoqué par l'absence d'amélioration de solution après un certain nombre de pas.

Les algorithmes de recherche locale sont largement utilisés pour la résolution de problèmes difficiles.

4.6.2 Algorithmes

Dans cette catégorie, nous présentons certains algorithmes connus et ayant des stratégies de mouvement diverses :

a. Algorithme de descente

Consiste à progresser au travers de l'espace de recherche en choisissant à chaque étape la meilleure solution voisine selon la fonction objectif considérée. Le procédé prend fin si un critère d'arrêt est atteint ou, au mauvais cas, lors de la rencontre d'un minima local.

La méthode de descente prend son origine des travaux de (Robbins & Monro, 1951) et (Kiefer & Wolfowitz, 1952) relatifs à l'approximation stochastique. Elle compte parmi les

metaheuristiques les plus sollicitées vue sa simplicité d'implémentation, toutefois, elles comportent certains inconvénients qui limitent leur efficacité, notamment :

- la recherche du meilleur voisin s'avère difficile si la taille de l'espace de recherche (voir la taille du voisinage) est importante.
- La méthode ne peut distinguer l'optimum global d'un optimum local et prend fin si l'un d'eux est rencontré. Elle s'avère donc inefficace lors de l'exploration d'un espace contenant plusieurs minima locaux, ce qui est le cas pour la plupart des problèmes combinatoires.

L'algorithme 4.2 illustre la méthode considérée. Il emploie comme entrée un espace de recherche en arbre où chaque nœud décrit les voisins qui lui y sont associés et une solution de départ aléatoire. Le résultat illustre la meilleure solution rencontrée.

Afin de faire face aux inconvénients ci-cités, des améliorations dans la méthode ont été apportées, notamment la possibilité d'accepter certaines solutions moins bonnes au cours du traitement dans le but de surpasser les minima locaux et par conséquent, étendre l'exploration à d'autres zones de l'espace de recherche.

Algorithme 4.2. Méthode de descente

Entrée: S,f : resp. espace de recherche et fonction objectif,
s : solution initiale ($s \in S$)
$V(s)$: structure voisinage <element>
Sortie: s^*: solution optimale
Initialisation :
$s^* \leftarrow s$
repeter
générer V(s)={$s_1,s_2,..,$ *taille_V(s)*}
$s \leftarrow s_1$
Pour $i=2 \rightarrow$ *taille_V(s)*
Si $f(s)<f(s_i)$ alors $s \leftarrow s_i$ **Fsi**
Fpour
Si $f(s) < f(s^*)$ **alors** $s^* \leftarrow s$ **Fsi**
Jusqu à (*critère d'arrêt)*

b. Algorithme BFS

Similaire à la méthode de descente dans la mesure du choix du meilleur voisin, l'algorithme BFS (*Breadth First Search*) permet de construire un chemin optimal qui lie un nombre fini de sommets d'un graphe en utilisant une file de priorité de voisins pour chaque sommet visité.

Son principe consiste et, à partir d'un sommet initial $s \in S$, ordonner les sommets voisins $v(s)$ dans une file d'attente selon leur importance par rapport à la fonction objectif. Le prochain sommet du parcours est choisi parmi les meilleurs sommets de $v(s)$ non encore exploités. La méthode permet de contourner les minima locaux dans la plupart des cas, d'éviter les sommets

visités, néanmoins, elle contourne également dans certaines circonstances, des sommets d'importance non négligeable.

Algorithme 4.3. Méthode BFS

Entrée: *S,f* : resp. espace de recherche de taille n et fonction objectif,
s_0 : Solution initiale (*s* ∈ *S*)
v: structure voisinage<élément>
F' : structure file_priorité (element,estimation,etat)
Sortie: *s**: solution optimale
Initialisation :
*s**←*s_0*
generer *v(s_0)*
creerfile(*F*) //initialement vide
enfiler(*F*,*v(s_0)*) //inserer le voisinage non visité de s_0
TantQue *F*≠*Ø*
TantQue F.element.etat=visité
Defiler(*F.element*)
FTantQue
F.element.etat←visité
s←*F.element*
générer *v(s)*
enfiler*(F,v(s))*
FinTantQue
*s**← *s*

L'algorithme génère une liste d'attente composée des successeurs de tous les sommets visités au cours de l'exploration, triés par qualité de solution. L'insertion dans la liste se fait par la fonction *enfiler*() qui sélectionne le voisinage non visité et l'insère d'une manière ordonnée. Chaque sommet visité est déduit de la liste par la fonction *defiler*().

La restriction de l'algorithme est qu'il gère un nombre *fini* de sommets appartenant à un graphe *orienté* car il n'y a pas de critère d'arrêt relatif au temps de calcul.

c. Algorithmes Glouton

Les algorithmes gloutons (*Greedy algorithms*) sont des stratégies de recherche itérative qui permettent, à chaque itération de modifier certains paramètres du problème, notamment la valeur du seuil à atteindre sans toutefois remettre en cause les choix antérieurs et ce, contrairement aux metaheuristiques classiques où une marche arrière est possible.

Leur principe est de démarrer d'une solution aléatoire qui sera modifiée à chaque étape et, éventuellement complétée d'une manière progressive afin qu'elle soit meilleure selon l'environnement local rencontré. La condition d'arrêt s'impose au cas où l'exploration devient impossible; La solution est construite en deux phases :

- Phase *constructive* permettant à chaque étape d'insérer un nouvel élément solution s à S. Cet élément est choisi parmi une liste de candidats classés dans une liste d'attente F selon leur profit en regard de la fonction objectif considérée. La qualité de la solution produite dépend du cardinal de cette liste : plus qu'il y a d'éléments candidats, plus qu'il y a de solutions produites. Les solutions générées en cette phase ne sont pas optimales en regard du voisinage de chaque élément choisi car le choix des éléments candidats se fait d'une manière globale sans tenir compte de leur caractère local.
- Phase *d'amélioration* permettant d'affiner la solution construite par un procédé de recherche locale. Son principe est de mesurer la performance de chaque élément par rapport à son voisinage $v(s)$ en vue d'une permutation possible.

Algorithme 4.4. Méthode GRASP

Entrée: S,f : resp. espace de recherche de taille n et fonction objectif,
s_0 : Solution partielle initiale ($s_0 \in S$)
F : structure file_priorité (element,estimation,etat)
Sortie: s^*: solution optimale
Initialisation :
$s^* \leftarrow s_0$
creerfile(F) //initialement vide
enfiler(F,s^*)
Répéter
TantQue $F.element.etat = visité$
Defiler($F.element$)
FTantQue
$F.element.etat \leftarrow visité$
générer $v(F.element)$
$s^* \leftarrow s^* \cup opt(F.element, v(F.element))$
Jusqu'à <*critère d'arrêt*>

Les phases ainsi énumérées sont répétées à chaque itération jusqu'à satisfaction du critère d'arrêt. La solution ainsi construite est considérée comme meilleure et renvoyé comme résultat. L'algorithme 6.4 illustre la méthode GRASP.

Afin d'améliorer le temps d'exploration, la liste F est mise à jour à chaque itération par suppression des éléments devenus inutilisables. La fonction $opt(a,b)$ permet de sélectionner l'élément ayant le plus de profit en regard de la fonction objectif considérée.

Ce type d'algorithmes peut conduire à un résultat exact, mais c'est rarement le cas, car il ne possède pas une vision globale sur l'espace de recherche. De même, les problèmes traités sont en général, de type combinatoire où on ignore d'avance leur résultat. En pratique, on fixe un seuil qu'on essaye de l'atteindre d'une manière rapprochée en acceptant cependant une solution si elle est proche de l'optimal fixé. Du côté implémentation, les algorithmes Gloutons peuvent être piégés dans des cycles, donc considérés comme incomplets. Ils s'arrêtent également au premier

chemin trouvé, donc non optimaux. Leur complexité en temps est de l'ordre de $O(b^p)$ où p est la profondeur de l'arbre et b est le facteur de branchement.

4.7 Metaheuristiques de branchement et optimisation

Plutôt d'examiner la totalité de solutions au sein d'un espace de recherche à la conquête de la meilleure d'entre elles, on préfère parfois ignorer une partie de cet espace, jugée non intéressante sous réserve qu'il soit organisé d'une manière prédéfinie. Ce principe relève d'un paradigme classique basé sur la notion : *diviser pour régner*, permettant de subdiviser le problème traité en sous-problèmes de taille réduite et simples à résoudre.

4.7.1 Séparation et évaluation

a. Définition

La méthode de séparation et évaluation, connue sous son appellation anglaise *B&B* est une technique générique de résolution de problèmes combinatoires discrets ayant un nombre fini de solutions. Proposée par (Land, H A & Doig, G A, 1960) , elle consiste, à *séparer* le problème considéré en plusieurs sous-problèmes de taille réduite. Partant d'une solution initiale, l'exploration procède ensuite à une *évaluation* de chaque sous-ensemble de solutions et de les majorer selon leur degré d'optimalité dans le but de retenir uniquement ceux susceptibles de contenir des solutions potentiellement meilleures que la solution de référence.

b. Principe

La structure de l'algorithme est basée sur un arbre ordonné E où la racine e_0 représente l'état initial du problème. Chaque feuille désigne une solution s admissible. Les nœuds internes réfèrent aux instanciations partielles de certaines variables d'état du problème. La séparation décrit la manière de construction de l'arbre E de solutions qui dépend de la nature du problème considéré. Cela revient à découper l'arbre E en sous-arbre $e_0, e_1, ..,e_r \in E$ (pas forcément disjoints) ou chaque e_i est associé à une solution s_r. L'évaluation consiste à déterminer la valeur optimale de chaque sous-arbre de racine e_r. : en partant d'un seuil d'optimalité initial s_0, la recherche consiste à parcourir chaque nœud e_i de l'arbre à la conquête d'une solution s^* meilleure que s_0. L'évaluation d'un nœud interne e_i consiste à déterminer le majorant s_i^* obtenu parmi les solutions s_i contenues dans le sous-arbre de racine e_r. A chaque pas i de descente, l'évaluation inspecte la solution du nœud e_i considéré ; si elle est meilleure que la solution courante s^*, elle la remplacera; dans le cas contraire, le sous-arbre de racine e_i sera éliminé.

L'avantage de la méthode permet donc de *raccourcir* l'espace d'exploration en s'en passant de certaines branches jugées inutiles.

c. Algorithme

Soit s_0 la solution initiale du problème. L'algorithme parcourt d'une manière itérative l'arborescence de solutions en évaluant chaque sous-arbre de racine *nœud_courant* en maintenant la meilleure solution trouvée s^*. Vu que l'arbre est ordonné, L'exploration s'arrête dès qu'une feuille est rencontrée.

Algorithme 4.5. Séparation & évaluation

Entrée: S : resp. espace de recherche de taille n
s_0 : Solution initiale ($s \in S$)
T: structure arbre(noeud,fils[p],estimation)
Sortie: s^*: solution optimale
Initialisation :
$T.nœud \leftarrow racine,$
$s_0 \leftarrow T.nœud.estimation$
$s_g \leftarrow s_0$
Repeter
$nœud\_courant \leftarrow T.noeud$
Pour i←1 à p **faire**
$s_i \leftarrow noeud\_courant.fils[i].estimation$
Si $s_i < s^*$
$s^* \leftarrow s_i$
$noeud\_courant \leftarrow noeud\_courant.fils[i]$
Sinon $noeud\_courant.fils[i] \leftarrow \emptyset$
FinPour
Jusqu'à ($nœud\_courant = \emptyset$)

d. Limites

Comme toute autre metaheuristique, l'algorithme *B&B* est gourmand en ressources notamment lors du traitement de grandes instances (Mezmaz, Melab, & Talbi, 2007). D'un autre côté, la manière du choix du prochain nœud peut avoir un effet sur des performances expérimentales même si l'arbre est ordonné. Cependant, la stratégie de la meilleure solution locale n'est pas toujours fiable étant donné que l'évaluation d'un nœud correspond à l'évaluation de toutes les solutions portées à ses feuilles et qui peuvent être distinctes. Toutefois et, en fonction de la structure de données utilisées, on envisage certaines autres alternatives telles la stratégie en largeur, celle du nœud le plus prioritaire ou une combinaison justifiée de plusieurs d'entre elles. De même, la qualité d'un nœud peut être estimée auprès d'un calcul statistique en tenant compte de la qualité des nœuds qui lui sont successeurs (moyenne des solutions dérivées, évaluation par rapport à un coefficient de pondération, etc.)

4.7.2 Path-Finding A*

L'algorithme *A-etoile* permet à l'aide d'une fonction d'évaluation, d'estimer le meilleur voisinage pour chaque nœud en construisant un chemin acceptable entre un nœud initial et un nœud final au sein d'un espace de recherche représenté par un graphe orienté moyennant une consommation de ressources optimale.

Proposé par (Hart P. , 1968), le A* est similaire à l'algorithme de *Dijkstra*. Son principe consiste et, à partir d'un état initial x affecté d'un coût s_0, nul en général, Ensuite, tous les voisins admissibles de x sont insérés dans une liste dite *Open-List* et seront analysés afin d'en déduire le nœud y ayant le coût optimal s_{xy}. Ce nœud va être transféré vers une autre liste, dite *Close-List* afin de ne pas intervenir désormais dans le processus de recherche. Le nœud x sera archivé comme élément du chemin optimal et remplacé par y dans *Open-List*; il servira de base pour la recherche suivante. Le processus réitère tant que cette liste contient des éléments ou si le nœud final est atteint. Le chemin optimal est l'ensemble des nœuds affectés à *Close-List* dans l'ordre de leur insertion.

Algorithme 4.6. A*

Entrée: S : resp. espace de recherche de taille n,
x, s_x : élément initial et solution associée ($x \in S$)
O: structure Liste(noeud,voisinage[p],f)
F: structure Liste(nœud,f)
Sortie: F: chemin optimal
$O.noeud \leftarrow \{x\}; F \leftarrow \{\emptyset\}; g(x) \leftarrow 0; O.f \leftarrow h(x);$
TantQue ($x \neq$ *nœud final*)
$F \leftarrow F \cup \{x\}$
$F.f(x) \leftarrow g(x)+h(x)$
$O.Voisinage \leftarrow \{y / \ y \in V(x) \wedge y \notin F\}$ pour tout y
Si $O \neq \emptyset$
$y^* \leftarrow y \ / f(y)=optimum(O.voisinage[p].f)$ pour tout p
Fsi
$x \leftarrow y^*$
FinTantQue

La spécificité de cet algorithme est qu'il exige une vision globale de l'espace, c'est-à-dire, connaître l'emplacement du nœud final, et par conséquent, proposer une *estimation heuristique* du chemin avant le traitement. La fonction d'évaluation sera de la forme $f = g+h$. Pour chaque nœud x, $g(x)$ étant la valeur du chemin optimal reliant le sommet de départ à x; $h(x)$ est la valeur estimée de l'autre partie du chemin reliant x au nœud final.

L'algorithme trouvera en général une solution lors du traitement des heuristiques admissibles. Dans ce cas, le dernier nœud doit apparaître dans *Open-List* lors des dernières itérations. Si ce n'est pas le cas, l'arrêt est provoqué par l'épuisement de tous les nœuds de cette liste. S'il se termine avec une solution, c'est qu'il n'y a pas d'autres alternatives meilleures.

D'un autre côté, l'algorithme garde la trace de tous les nœuds visités, il s'avère gourmant en mémoire si le nombre d'instances est grand. De même, il exige une vue d'ensemble sur l'espace de recherche, notamment l'existence d'un chemin entre le nœud de départ et le nœud final et pouvoir l'évaluer. De même, et pour un problème donné, il peut y exister plusieurs chemins de départ possibles avec un nombre de nœuds distincts. Si un mauvais choix de départ se produit, l'algorithme est forcé de traiter d'avantage de nœuds sans toutefois améliorer le résultat. Dans de telles situations, un traitement préliminaire est requis telle l'affectation d'une valeur raisonnable à la fonction *h* ou utiliser une application de la programmation dynamique permettant d'éliminer tous les chemins inutiles.

4.8 Metaheuristiques à population

Classe de metaheuristiques inspirés du comportement social d'animaux vivants en communautés organisées. Elle est basée sur le *mimétisme* où les individus évoluent dans un milieu caractérisé par une compétition amicale qui les oppose lors de la conquête de nourriture. En effet, ces individus sont dispersés au sein de l'espace en gardant un contact observateur de leur voisinage. Dès qu'un d'entre eux localise une piste conduisant à un butin, les autres seront informés dès que possible et tendent de l'imiter dans sa progression à base d'un comportement présent au niveau de chaque individu, à savoir :

- Chaque individu possède une information limitée relative à son environnement local et ses voisins. Ce fait permet de créer une cohérence d'interaction entre les individus de la communauté malgré la manœuvre autonome de chacun d'eux.
- Chaque individu obéit à un ensemble restreint de règles simples permettant de créer une coordination efficace des activités sans supervision.
- Absence de conflits ou inter blocage, plutôt une collaboration implicite et auto-organisée

Ce phénomène, appelé *intelligence en essaim* (Beni & Wang, 1989) est construit à base d'interactions, souvent très simples, permettant à la communié de résoudre des problèmes compliqués.

Les metaheuristiques inspirées de la nature font partie des techniques évolutionnaires élaborés au cours des années 50 (Fraser, 1957). Plus tard et, avec l'apparition des calculateurs de puissance en 70, la modélisation de ces techniques a donné naissance à diverses approches, notamment :

- Les stratégies d'évolution dédiées aux problèmes à variables continues,
- La programmation évolutionnaire qui tente d'évoluer les structures d'automates à états finis,
- Les algorithmes génétiques destinés à résoudre les problèmes à variables continus.

Ces techniques sont dédiées essentiellement à la résolution des problèmes où les algorithmes classiques d'optimisation, d'apprentissage ou de conception automatique ne peuvent fournir des résultats satisfaisants surtout en matière de précision.

4.8.1 Colonies de fourmis

a. Définition

L'optimisation par colonie de fourmis, *ACO* en abrégé, est une classe de metaheuristiques inspirés par le comportement des fourmis au sein d'une colonie et, dédiée à résoudre les problèmes d'optimisation difficiles. L'idée originale a été proposée par (Colorni, Dorigo, & Maniezzo, 1991) (Dorigo M. , 1992). Le *Ant-System* (Dorigo, Maniezzo, & Colorni, 1996) était le premier algorithme inspiré de cette analogie ; il a été destiné à résoudre le problème de voyageur de commerce ; ses résultats n'ont pas été compétitifs. Plus tard, de nombreux autres algorithmes ont vu le jour : le *Ant Colony System* (Gambardella & Dorigo, 1997), *Elitist Ant_system* (Bonabeau, Dorigo, & Theraulaz, 1999), *MAX-MIN Ant-system* (Stütle & Hoos, 2000), *Hypercube Ant-system* (Blum, Roli, & Dorigo, 2001) dont le champ d'application a été étendu à plusieurs types de problèmes, notamment l'ordonnancement, l'assignement quadratique, le data mining, le routage, le transport, etc.

b. Description

Lors de son déplacement dans un espace de recherche pour la conquête de nourriture, chaque fourmi dépose au passage de la *phéromone*, une substance odorante permettant de créer une piste chimique. Au contact de l'air, la phéromone s'*évapore* après un laps de temps. Le mouvement des fourmis se fait d'une manière aléatoire dans un premier temps ; à la détection d'une piste, la fourmi qui, communique avec ses congénères par *stigmergie* (Grassé, 1959) est censée suivre cette piste en déposant de nouveau son propre phéromone. L'intensification du trafic se localise aux alentours du nid et également dans le secteur de la nourriture, ce qui renforce en phéromone les pistes qui relient ces deux zones avec disparition des pistes lointaines. Ce comportement permet, au fur de la progression des fourmis, de retrouver le chemin le plus court (selon la géographie de l'espace) entre le nid et la nourriture qui, désormais, emprunté par l'ensemble de la colonie.

c. Principe et algorithme

L'algorithme ACO a été construit à base des étapes suivantes :

- Dans un espace de recherche, assimilé à un graphe fortement connexe, chaque fourmi *m* est censée se trouver sur un nœud *i*.
- Si *i* n'est pas un nœud terminal, la probabilité que son prochain nœud sera *j* est donnée par la relation suivante :

$$p(i \to j) = \frac{\tau(i,j)^{\propto} d(i,j)^{\beta}}{\sum \tau(i,j)^{\propto} d(i,j)^{\beta}} \quad (4.2)$$

où $\tau(i,j)$ et $d(i,j)$ respectivement : la quantité de phéromone sur l'arc (i,j) et sa longueur; α, β : coefficients de pondération comprises entre 0 et 1.

- En empruntant l'arc (i,j), la fourmi dépose une trainée $\Delta\tau(i,j)$ de phéromone qui s'ajoute à la quantité déjà présente sur cet arc conformément à la relation :

$$\tau(i,j) \leftarrow \tau(i,j) + \Delta\tau(i,j). \quad (4.3)$$

- La phéromone déposée s'évapore d'une manière naturelle et homogène sur l'ensemble les arcs du graphe. La quantité évaporée par unité temporaire est définie par la relation :

$$\tau(i,j) \leftarrow (1-\rho)\tau(i,j) \qquad (4.4)$$

où ϱ, le taux d'évaporation, constante comprise entre 0 et 1.

Algorithme 4.7. ACO

Entrée: $S(\tau,d)^{m,n}$: espace de recherche de taille m×n,
d^p: Colonie de fourmis de taille p
s_i : Liste(nœud) // solution initiale associée à a_i (i=1..p) avec $s \subset S$
Sortie: s^*: chemin optimal
Initialiser $\tau(i,j)$, $d(i,j)$ pour tout $(i,j) \in (m \times n)$, *periode_evap*
Pour chaque fourmi a_i
  $s_i \leftarrow \{i\}$ avec *i : nœud aléatoire de S*.
  Evap←0
FinPour
TantQue $i \neq$ *nœud final*
  Pour chaque fourmi a_i
    Choisir un nœud j cf (eq. 4.2)
    Si j ∉ s_i alors
    $s_i \leftarrow s_i \cup \{j\}$
    FinSi
    Pheromenter (i,j) cf (eq. 4.3)
    i←j
    evap←*evap*+1
    Si *evap=periode_evap*
      Evaporer (i,j) cf(eq. 4.4) pour tout *(i,j)*
      Evap←0
    FinSi
    $s^* \leftarrow Opt(s_i, i=1..p)$
  FinPour
FtantQue

d. Versions

Le succès et la popularité de la technique notamment dans la recherche du plus court chemin par de mécanismes simples a permis d'élaborer plusieurs versions répondant aux divers types de problèmes traités :

- Fourmis virtuelles

Version est dédiée à résoudre des problèmes discrets. Contrairement aux fourmis naturelles (dites réelles) qui se déplacent et déposent leur phéromone d'une manière continue, le déplacement des fourmis virtuelles consiste en des transactions d'états en passant d'un nœud à un autre. Le dépôt de phéromone se fait une fois la fourmi ayant accompli un chemin *hamiltonien* entre le nid et la nourriture, la quantité déposée est proportionnelle à la qualité de la solution. Bien entendu, cette particularité nécessite une mémoire *tabou* contenant l'historique des actions. Le dépôt de la phéromone se fait en chemin de retour, exigeant également un parcours supplémentaire. L'évaporation s'opère d'une manière périodique.

La technique des fourmis virtuelle exige d'avantage de mémoire et de temps. Ce fait est récompensé par certains avantages :

- Seules les fourmis ayant atteint leurs destinations ouvrent droit au dépôt de phéromone, autrement dit, les fourmis bloquées en minima locaux seront simplement éliminées du circuit.
- Lors de son déplacement, la fourmi choisi l'état suivant à base de la performance de son parcours global et non de l'état local. De même, elle peut revenir sur un état parcouru si sa décision est mauvaise.

- Fourmis élitistes

Version améliorée des fourmis virtuelles. Elle consiste à octroyer une quantité additionnelle de phéromones dite *bonus* sur les arcs appartenant aux meilleurs tours trouvés.

Le but de la méthode permet, d'une manière intuitive à garder les meilleurs chemins riches en phéromone et focalise la recherche sur les zones prometteuses en incitant certaines fourmis dites *élitistes* à emprunter les arcs faisant partie des tours déterministes ce qui garantit la convergence et évite les solutions non-optimales.

- Max-Min Ant-System

Version élaborée par (Stütle & Hoos, 2000). Elle consiste à favoriser l'exploration d'avantage d'espace et ce par la limitation de la quantité de phéromone déposée ou évaporée sur les arcs du graphe représentant l'espace considéré. Son principe est de garder une certaine homogénéité de phéromone sur l'ensemble de ces arcs par la limitation des quantités de phéromone déposée ou évaporée par deux bornes supérieure B_{sup} et inférieure B_{inf} définies par l'expérimentation. Egalement, le dépôt de phéromone se fait à chaque itération par seule, la fourmi se trouvant sur le meilleur chemin du graphe.

- Virtuel Max-Min Ant-System

Version hybride élaborée à base de la version des fourmis virtuelles et celle du Max-Min Ant-system. Elle consiste à borner la quantité de la phéromone sur chaque arc par deux limites supérieure et inférieure et que la phéremontation se fait uniquement par les fourmis ayant accompli un chemin hamiltonien.

e. Limite

Comme tout autre metaheuristique, chaque individu au sein d'une colonie ACO est à priori indépendant dans son mouvement, ce qui favorise le mécanisme de l'exploration. Son inconvénient réside dans la possibilité de se bloquer dans des minima locaux ; ceci est dû

essentiellement à une mauvaise répartition initiale des fourmis à travers l'espace d'exploration. Cette idée pose également un autre problème majeur relatif au nombre important de paramètres mis en jeu et qu'il fallait initialiser d'une manière optimale, un défi difficile à réaliser en l'absence de modèles mathématiques permettant de gérer ces paramètres.

4.8.2 Algorithmes génétiques

a. Définition

Faisant partie des algorithmes évolutionnaires, les algorithmes génétiques (*GA* en abrégé) sont des méthodes d'optimisation stochastiques simulant l'évolution naturelle dans le sens lié à la compétition où seuls, les individus les plus aptes sont autorisés à survivre et transmettre leur patrimoine génétique aux générations suivantes. En pratique, ces algorithmes permettent de fournir des solutions rapprochées à un problème d'optimisation dans un temps raisonnable. Le premier modele formel fût introduit en littérature par (Holland, 1975), suivi par plusieurs études de vulgarisation (Goldberg D. , 1989) (Jean-Marc & Thomas, 1993) (tlo@mit.edu, 2003). Actuellement, les GA sont employés dans divers domaines, notamment l'industrie, l'économie, la planification et les outils de prise de décision.

Les GA utilisent une *population* comportant un certain nombre d'individus (*chromosomes*). Le mécanisme de l'évolution consiste à faire évoluer cette population au cours d'un certain nombre de *générations* dans le but d'en améliorer les individus en utilisant des transformations génétiques : *croisement* et *mutation*. Chaque nouvelle génération maintient un nombre fixe de chromosomes *sélectionnés* à base de leur adaptation à l'environnement et qui *remplacent* certains autres moins performants dans le but de garder une taille constante de chaque génération.

b. Description et algorithme

Comme tout algorithme itératif, les *GA* débutent par une population $p(0)$ de n individus ayant des caractéristiques variées et généralement aléatoires. Chaque individu est représenté par une chaine de m caractères (ou bits) représentant une solution éventuelle du problème considéré. Le corps de l'algorithme repose sur une boucle qui enchaine les diverses étapes relatives à l'évolution génétique, à savoir :

- l'évaluation des solutions à base de la fonction objective du problème considéré,
- La sélection d'un sous ensemble de solutions $s' \in S$ avec $s' \leq n/2$ conformément à certaines règles définis par les opérateurs génétiques.
- Soumettre les solutions sélectionnées aux opérateurs génétiques : croisement de sous chaines et mutation de caractères (ou bits) afin de reproduire de nouvelles solutions différentes de leurs antécédents,
- Construire une nouvelle génération de solutions à partir de la population précédente et des solutions nouvellement crées. Afin de garder une population de taille fixe, certaines solutions de la génération seront remplacées.

Le processus itératif prend fin si une solution acceptable a été obtenue ou au cas échéant, en absence d'amélioration entre deux générations ou à la limite d'une période d'exécution fixe.

Algorithme 4.8. AG

Entrée: Sp : espace de recherche de taille p,
$p_{c,m}^{0}$: population initiale de chromosomes de taille c×m gènes
s_c^G : ensemble de solution associées aux chromosomes c durant la génération G (c=1..p ;G=1..n)
Sortie: s*: chemin optimal
Pour chaque chromosome c évaluer s_c^0 **FinPour**
TantQue $G < n$
Sélectionner $p_{k,m}^{G}$ chromosomes (k<c/2)
Croiser(c_1, c_2,t) avec c_1, $c_2 \in p_{k,m}^{G}$ et t :nbre de segments
Pour chaque chromosome c
Muter(c_i) avec i : aléatoire compris entre 1 et m
FinPour
$s^* \leftarrow Opt(s_c^G, i=1..p)$
FtantQue

c. Opérateurs génétiques

Des transformations « aveugles » s'opérant stochastiquement sur les individus indépendamment de la nature du problème traité. Ils sont au nombre de trois, à savoir :

- *Sélection/Remplacement* : permet de déterminer *n* individus qui vont subir les opérateurs génétiques parmi les *p* individus de la population considérée. La sélection peut être réalisée de différentes manières :
  - *rang* : les individus ayant les meilleurs profils d'adaptation,
  - *roulette* : sélection aléatoire par probabilité de i/p (avec p=∑i) où i désigne le poids d'adaptation de chaque individu.
  - *tournoi* : sélection par élimination des individus pris deux à deux,
  - *uniforme* : sélection aléatoire selon une probabilité de 1/p.
- *Croisement* : a pour objectif de créer de nouveaux individus par recombinaison d'une certaine façon d'un ou deux individus parents procréateurs. La manière la plus utilisée dans la recombinaison consiste à découper la structure d'un individu en deux, voire trois segments, les échanger avec des segments de taille similaire provenant d'un autre individu. Le but du croisement est de créer une variété de genres à chaque génération et d'éviter ainsi la domination d'un type particulier.
- *Mutation* : conçu dans le but d'apporter une certaine diversité dans la population et permettant de conserver toutes les caractéristiques disponibles des individus au sein de chaque génération. Afin d'appliquer ce principe d'une manière souple, la mutation s'opère avec une faible probabilité de l'ordre de 4 à 5%.

d. Limite

L'efficacité des GA est conditionnée par le choix de meilleures valeurs des divers paramètres utilisés. Ces paramètres sont nombreux : la taille de population, nombre de points de croisement,

manière de sélection et de remplacement auxquels on ajoute les caractéristiques de l'implémentation à savoir fonction objective, le nombre de générations, etc. Ces paramètres sont difficiles à ajuster en l'absence de modèle mathématique adéquat. Il faut cependant user de l'expérience et de l'intuition, chose non souvent évidente.

e. Mise en œuvre

Contrairement aux stratégies de descente ou de recherche locale où la progression est liée étroitement à la valeur de la solution, l'évolution au sein des GA s'intéresse particulièrement à la structure de la solution. Il est souhaitable au-delà de cette propriété, d'imaginer une conception d'individus qui permet aux opérateurs génétiques de concevoir une descendance d'individus associés à certaines solutions acceptables. Cette idée pourra être imaginée sous l'une des deux propositions suivantes :

- Repérer et conserver un bloc réduit de solution valide à travers une majorité des individus sélectionnés à chaque génération,
- Repérer et éviter les variations dans la structure des individus qui entrainent des perturbations majeures dans le mécanisme de la recherche et influent bien entendu sur la convergence de la solution.

4.8.3 Essaims particulaires

a. Définition

Son origine provient des observations de (Reynolds, 1987) et développée et publiée par (Kennedy & Eberhart, 1995) l'optimisation par essaims particulaires, (*OEP*, en abrégé) ou PSO (en anglais), est une metaheuristiques bio-inspirée des communautés vivant en collectivités organisées notamment les groupes d'oiseaux, essaims d'insectes ou bancs de poissons. Elle exploite une population d'individus pour la recherche de régions prometteuses au sein d'un espace de solutions à *n* dimensions. Ce type d'algorithme a prouvé son efficacité dans divers domaines notamment la construction mécanique, conception de systèmes, traitement d'images, etc.

b. Description

La méthode repose sur une population d'individus disposés aléatoirement à travers un espace de recherche. Chaque individu (appelé *particule*), constitue une solution potentielle du problème considéré, se déplace d'une position à une autre avec une *vitesse* adéquate en ajustant la direction de sa trajectoire selon trois paramètres: sa meilleure position, la meilleure position au sein de ses proches voisins ainsi que la meilleure position dans le groupe. A base des optimums locaux et empiriques, l'ensemble des particules ont tendance à converger vers les positions permettant d'aboutir à une solution optimale du problème traité.

c. Formalisation et algorithme

Un essaim de particules est caractérisé par un nombre donné de particules fixé d'une manière expérimentale à base de la taille de l'espace de recherche et la puissance de calcul utilisée. Chaque particule est dotée de :

- une vitesse limitée entre deux bornes v_{max} et v_{min} et ce, afin d'éviter l'explosion de l'espace et passer à côté de l'optimum ou bien se regrouper et réduire l'espace.
- un voisinage en guise de réseau social restreint sélectionné par divers géométries selon la nature de l'environnement.
- des coefficients de confiance : un compromis entre la tendance à la conservation dans le but de prendre une décision instinctive ou la confiance à l'expérience du voisinage afin de l'imiter dans ses décisions.

Lors de son mouvement dans un espace $\mathbb{R}^n$, au moment t, chaque particule i a une position x_i^t. Elle se déplace avec une vitesse $v_i^t \in \mathbb{R}^n$ selon sa perception de l'environnement telle la cohésion, l'alignement et la direction des voisines.

La vitesse et la position de la particule i, au moment t, sont mises à jour conformément aux règles suivantes:

$$v_i^{t+1} = c_i v_i^t + c_p(p_i^t - x_i^t) + c_g(g^t - x_i^t) \tag{4.5}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{4.6}$$

avec p_i et g respectivement la position de la voisine de i la mieux placée et la position de la meilleure particule du groupe. c_i, c_p and c_g: des nombres aléatoires (coefficients de *confiance*) distribués uniformément dans [0,1].

Algorithme 4.9. OEP

Entrée: S^n : espace de recherche de taille n,
s : ensemble de solution associées aux particules i (i=1..n)
Sortie: s^*: solution optimale
Initialisation :
Générer x_i, v_i, c_i (i=1.. n), c_p (p=1.. n_p) et c_g
Evaluer s_i, (i=1.. n)
Générer p_i = *voisinage(i)* pour tout i
Initialiser $g \leftarrow \{p_k / s_k = opt(s_p, p=1..n_p\}$
Répéter
Tirer nombres aléatoires: c_i, c_p, $c_g \in [0,1]$
Pour chaque particule i faire
$v_i \leftarrow c_i v_i + c_p(p_i - x_i) + c_g(g - x_i)$ avec $v_i \in \{v_{min}, v_{max}\}$
$x_i \leftarrow x_i + v_i$
Si $(s_i < s_{pi})$ $p_i \leftarrow x_i$ **FinSi**
Si $(s_{pi} < s_g)$ $g \leftarrow p_i$ **FinSi**
FinPour
Jusqu à <*critère d'arrêt*>

Les données initiales du problème comportent entre autres, la taille du problème, les paramètres des particules : positions, vitesses, voisinage et coefficients de confiance (aléatoires). Ces paramètres permettent de calculer les solutions correspondantes. Etant donné que la solution optimale n'est guère imprévisible, il est préférable d'utiliser plusieurs critères d'arrêt, à savoir, une solution acceptable et, au cas échéant une limite en temps ou un nombre d'itérations prédéfini.

d. Limite

La variable *v* relative au déplacement de la particule est en fait, abusive ; elle réfère plutôt à une direction qu'à une vitesse car le vecteur $\overrightarrow{v_i}$ n'est pas homogène à la propriété de la vitesse. Par conséquent, une forte valeur de *v* assignée à une particule peut la déporter hors des limites de l'espace et attirer d'autres particules si sa performance est acceptable. Ce problème a été objet d'étude par (Eberhart, Simpson, & Dobbins, 1996) qui a introduit la notion de limitation de la distance parcourue par une particule au cours d'une itération. En plus, les coefficients de confiance associés à la vitesse permettent de stabiliser l'équilibre entre l'intensification et la diversification. Les études de (Clerc & Kennedy, 2002) et (Kennedy, Eberhart, & Shi, 2001) ont montré que la convergence peut être obtenue si ces coefficients sont dépendants, chose difficile à réaliser en pratique car elle est spécifique à chaque type de problème traité.

De même, la topologie et le volume du voisinage ont une influence considérable sur la performance de l'algorithme. Ainsi, la *version globale* où l'essaim est associé à un voisinage unique altère en partie l'exploration, ce qui conduit à la stagnation dans un optimum local et donc, à une convergence prématurée. Les autres variantes de l'algorithme dites *versions locales* où l'essaim est réparti en plusieurs sous-ensembles ayant différents voisinages, ont été proposées afin d'améliore la convergence. Néanmoins, la taille de chaque voisinage demeure un compromis entre la convergence et la consommation de ressources qui doit être ajustée selon l'environnement d'application.

Afin de répondre à ces imperfections, plusieurs approches ont été élaborées telle l'insertion d'un coefficient d'inertie (Shi & Eberhart, 1999) (Chatterjee & Siarry, 2006) permettant de créer un équilibre entre la recherche locale et globale, la topologie FIPS (Mendes, Kennedy, & Neves, 2004) qui utilise l'information disponible au sein de plusieurs voisines au lieu de la meilleure et la version TRIBES, un algorithme PSO sans paramètres (Clerc, 2003) ou l'utilisation d'une topologie dynamique de voisinage *Dcluster* (Abbas, 2012).

4.9 Performance des metaheuristiques

Pour résoudre un problème donné, on cherche toujours un algorithme spécifique ou dédié à ce problème, s'il en existe. On gagne cependant en complexité et consommation de ressources ; Dans le cas contraire, une metaheuristique peut être adaptée à cet effet. L'évaluation de sa performance dépend de la qualité des réponses aux questions suivantes :

- Peut-on avoir une solution lorsqu'elle existe ?
- Si c'est le cas, quelle metaheuristique permettant d'avoir une bonne solution ?
- S'il existe une bonne solution, peut-on confirmer si elle est optimale ?
- Le compromis ressources/solution-optimale est-il raisonnable ?

A l'exception de la première question, il n'est malheureusement pas possible de répondre de manière directe et précise aux autres questions posées: en effet, si diverses metaheuristiques pouvant être utilisées pour la résolution d'un problème donné, il est difficile d'en favoriser une par rapport aux autres car, certaines similitudes sont évidentes entre ces différentes approches (voisinages), elles diffèrent sur des points relativement délicats (fonction objectif, liste des mouvements interdits, opérateurs et paramètres, …). La comparaison s'avère délicate pour deux raisons : d'une part, la manière d'affiner les différents paramètres intervenant dans ces méthodes avec la même rigueur et, d'autre part, la quantité de ressources permettant d'obtenir une solution acceptable.

Les metaheuristiques *évolutionnaires* seront probablement plus gourmandes en ressources de calculs, mais on peut supposer aussi qu'elles se prêteront bien au parallélisme, donc parviennent à de bonnes solutions pour de problèmes complexes. Leur performance dépend du paramétrage utilisé et de l'équilibre entre la diversification et une exploration locale.

Les metaheuristiques à *parcours* présentent l'avantage d'être simples à mettre en œuvre et peuvent offrir des solutions de qualité acceptable pour des instances de taille moyenne, néanmoins, elles présentent certaines faiblesses tel la possibilité du blocage en minima locaux ainsi qu'un traitement de taille quant aux méthodes dotées de mémoire.

Les méthodes *exactes* offrent, dans la plupart des cas des solutions, mais au prix de temps de calcul prohibitif pour nombres d'applications complexes sans toutefois garantir qu'elles sont acceptables.

Certains algorithmes peuvent se ranger dans plus d'une catégorie à la fois, telle la méthode GRASP, qui construit un ensemble de solutions, qu'elle améliore ensuite avec une recherche locale. La plupart des méthodes développées ces dernières années sont d'ailleurs souvent à cheval sur ces deux approches ; donc, elles permettant tirer profit de leurs avantages cumulés. La plupart des approches récentes ont été élaborées à base d'un algorithme à population, couronné par une phase de recherche locale.

En conclusion, les metaheuristiques ne peuvent, a priori, être spécifiques à la résolution de tel ou tel type de problème, leur classification reste assez arbitraire. Quelle que soit l'approche utilisée, il faut constater qu'il est possible de résoudre n'importe quel problème d'optimisation combinatoire pourvu qu'il soit solvable ; La qualité de la solution obtenue et le volume de ressources consommées dépend en grande partie de la manière de modélisation et éventuellement d'une petite chance supplémentaire.

4.10 Conclusion

Les metaheuristiques engendrent une famille de méthodes approchées adaptables à un grand nombre de problèmes d'optimisation combinatoire. Leur efficacité dépend de divers critères, la manière de modélisation, le type de paramétrage, la stratégie d'exploration. Ainsi, aucune méthode particulière ne peut garantir qu'une metaheuristique sera plus efficace qu'une autre sur problème donné.

La performance des metaheuristiques peut différer d'une instance à l'autre et il n'est souvent pas possible de définir une, qui s'avère la plus adaptée sur l'ensemble des instances (cf. *No Free Lunch Theorem*). Il s'avère donc nécessaire d'opter pour certaines modifications visant à améliorer

la qualité de la solution, notamment le principe d'oscillation stratégique (Glover & Laguna, 1997) visant à modifier périodiquement les paramètres d'une recherche itérative, la structure de la solution (tel le croisement de segments de solutions) afin de créer la diversification ou utiliser une recherche guidée en forçant l'utilisation de certains déplacements non sollicités afin de favoriser certaines portions de l'espace.

5. Les Metaheuristiques en Cryptanalyse Classique

5.1 Introduction

En littérature, la cryptanalyse des chiffrements ou plus précisément, le problème de recherche de clé peut être modélisé comme tout autre problème combinatoire similaire. L'espace d'exploration est assimilé à une structure arborescente où chaque chemin correspond à une solution du dit problème. Le modèle de l'arbre est représenté implicitement par un état de départ et des transitions. La complexité dépend du nombre de voisinage de chaque nœud et de la profondeur de l'arbre ainsi que la longueur des divers parcours envisagés durant l'exploration. Le temps est mesuré en fonction du nombre de nœuds examinés et l'espace est mesuré par le nombre de nœuds conservés en mémoire.

Dans ce contexte, selon l'étude de (Uddin & Youssef, 2006b), (Uddin & Youssef, 2006a), la cryptanalyse des chiffrements classiques peut être réduite à un problème de *PVC* qui peut être formulé de la manière suivante : dans un graphe connexe G_n de n nœuds (correspondants à n caractères), une clé k_t de t caractères ($t \leq n$) est représentée par un chemin reliant t sommets du sous-graphe fortement connexe G_t. D'un autre côté, les études menées (Clark, 1998) ont montré que les metaheuristiques d'optimisation sont idéalement adaptées aux implémentations des attaques envers les chiffrements classiques. Il ajoute que, contrairement à certains problèmes où l'approche heuristique donnera un résultat optimal, la clé de coût minimal en cryptanalyse ne traduit pas forcément à un texte clair optimal en nature de l'approximation de la fonction objectif dont la valeur est liée au référentiel de comparaison utilisé (corpus). Cette idée a été endossée par (Giddy & Savafi, 1994) qui affirment que :

"Most failures to decrypt correctly were the result of the global minimum not being the correct decipherment [. .] The simulated annealing algorithm finds the global minimum, but this minimum does not necessarily represent the correct decipherment."

Il est important de noter que les techniques fondées sur l'optimisation ne constituent pas une "solution miracle" pour briser les systèmes cryptographiques, et encore moins pour résoudre des problèmes difficiles en général. Une illustration concrète de ce fait concerne l'application des algorithmes génétiques pour la résolution du problème *Knapsack Cipher*[1] effectuées par (Spillman, 1993) et qui a reçu beaucoup d'attention d'auprès divers chercheurs dans la littérature dont (Clark, Dawson, & bergen, 1996) qui ont démontré que les modèles proposées ne conviennent que pour la cryptanalyse des chiffrements de taille réduite car, en pratique , il est douteux qu'ils soient capable à affronter des problèmes réels.

La méthode la plus simple de l'exploration de l'espace de clés est la recherche exhaustive qui nécessite une moyenne de n !/2 évaluations avant d'identifier la bonne solution ce qui est hors norme pour un espace de 26 nœuds, toutefois si l'espace de recherche peut être réduit d'une manière intelligente en ignorant les chemins inutiles, l'exploration devient raisonnable moyennant certaines spécificités liées à l'objectif du problème (Bauer, 1997).

Le modèle de référence d'un problème de cryptanalyse est la *permutation* ou *combinaison probabiliste* avec ou sans répétition où, d'après la théorie de probabilité; le résultat de tirage aléatoire de certaines objets d'une catégorie donnée est quasi non nul car une modification mineure dans la fonction de calcul telle la permutation de deux caractères d'une clé de déchiffrement aura une percussion mineure sur le résultat. Selon la théorie de Shannon (Shannon, 1949), il existe une relation entre la clé et le texte chiffré et, à base de ce constat, il s'ensuit qu'une clé correcte (ou presque) produit un résultat lisible (Clark J. A., 2003).

5.2 Formulation du problème

5.2.1 Structure de clé

La *clé* est un ensemble de caractères de l'alphabet dont la taille dépend du chiffrement utilisé : s'il est monoalphabétique, la clé est une table à deux dimensions comportant les 26 lettres de l'alphabet en ordre sur une ligne et en désordre sur l'autre ligne. Pour les autres chiffrements, sa taille est *t* et, est déduite à partir de l'indice de coïncidence du texte chiffré. Pour le chiffrement par transposition, la clé est convertie en chiffres indiquant l'ordre des lettres qui la composent. Au départ, une clé est générée aléatoirement. Au cours du traitement et, pour les chiffrements par substitution monoalphabétique et transposition, la génération de clés s'opère à l'aide des permutations sans répétition de caractères de clés précédentes, tandis que la clé des autres chiffrements est une combinaison d'un certain nombre des caractères de l'alphabet utilisé avec possibilité de répétition.

5.2.2 Structure de textes

La base de textes utilisée doit être aussi grande que possible afin de refléter les statistiques standards du langage utilisé. En littérature, les textes d'essai utilisés sont généralement de taille spécifiée (Garg, 2005) (Toemeh & Arumugam, 2007) (Morelli, 2004) ou dédiés à un certain environnement ou dans la plupart des cas, non spécifié (Hamdani, Shafiq, & Khan, 2010). De même, certains travaux ont préféré, et sans raison apparente, des textes d'un certain style

1. Système de chiffrement à dé publique inventé par Ralph Merkle et Martin Hellman en 1978.

particulier dont les résultats peuvent ne pas satisfaire d'autres types de textes (Forsyth & Savafi, 1993) ; d'autres (Jackobsen, 1995) (Spillman, Janssen, Nelson, & Kepner, 1993) ayant utilisé des tes simples de source inconnue.

De même, la base d'essai doit comporter un ou plusieurs corpus ou plus précisément des tables de fréquence de caractères extraites de corpus standards afin que le référentiel des tests (fonction de coût) doit être le plus proche possible du réel (Mekhaznia & Menai, 2014). Ainsi, une fonction objectif bien construite doit pouvoir connaître le langage du texte chiffré et permet, dans la mesure du possible d'avoir une idée sur le style du chiffrement considéré.

5.2.3 Solution initiale

L'application d'une technique heuristique nécessite une solution initiale de départ, en général aléatoire. Afin de raccourcir l'espace d'exploration et limiter le temps de recherche, cette solution doit être élaborée manuellement à base de certaines données jugées optimales, ou à base d'un prétraitement ou résultat préliminaire d'un autre algorithme notamment le GRASP ou simplement par intuition. La solution initiale est une clé si la metaheuristique utilisée est une recherche locale et un ensemble de clés si la metaheuristique est à population.

La fonction 5.1 comporte un pseudo-code qui détermine la taille de la clé *t* à partir du texte chiffré en utilisant les fonctions appropriées définies au chapitre 3. Cette clé est définie par une structure de table initialisée par des lettres aléatoires de l'alphabet. Le paramètre *TestDuplicate* est une fonction qui ignore tout caractère précédemment choisi afin d'éviter les répétitions des caractères au sein de la clé. Elle est utilisée uniquement pour la substitution monoalphabétique car les autres types de chiffrements autorisent la répétition de caractères.

Fonction 5.1. SolutionInitiale(Cipher, Alpha, t_max)

```
IndiceCoincidence ←Friedmantest(Cipher)
Si IndiceCoincidence ≈ 0.065          // Chiffrement mono
    Pour i ← 1 à 26 faire
        K[i]← random(Alpha,26,TestDuplicate)
    FinPour
FinSi
Si IndiceCoincidence ≈ 0.038          // Chiffrement poly/transpo
    t ← kazizkiTest(Cipher,t_max)
    Pour i← 1 à t faire
        K[i]← random(Alpha,26)
    FinPour
FinSi
```

5.2.4 Recherche locale

Le voisinage d'une solution *V(k)* est l'ensemble de clés *k'* que l'on peut obtenir en appliquant à *k* des permutations ou décalages selon plusieurs manières à savoir :

- *Remplacement* : le caractère courant sera remplacé par un autre choisi parmi un sous ensemble de l'alphabet disponible,
- *Echange* : intervertir le caractère courant avec son successeur si existe,
- *Insertion* avec décalage : consiste à choisir une position *i*, et insérer en position courante le caractère situé en position *i* (avec 0 < i < taille de la clé) puis à décaler tous les caractères anciennement situés entre la position courante (incluse) et *i* (exclus) d'un cran à droite si i > position courante et d'un cran à gauche sinon.
- *Permutation* : consiste à permuter le caractère courant avec un autre situé en position i (avec 0<i<taille de la clé).
- *Inversion* : consiste à inverser l'ordre de l'écriture des caractères situés entre la position courante et une position i (avec 0 < i < taille de la clé).

La transformation doit prendre en compte les restrictions suivantes :

- Omettre tout échange déjà opérée et ce, à l'aide d'une liste Tabou d'une taille donnée,
- Omettre une utilisation double d'un même caractère dans le cas de substitution monoalphabétique.

Définir un voisinage s'avère embarrassant et nécessite d'avantage d'idées, voire d'intuition afin de rediriger l'exploration vers la zone adéquate de l'espace de recherche, de limiter le temps de recherche et surtout d'éviter les optima locaux.

En cryptanalyse et, à chaque itération, un caractère de la clé fera objet de modification ou de remplacement par un autre appartenant à son voisinage. Selon le type de chiffrement, la modification peut atteindre plusieurs caractères simultanément. La fonction permettant par exemple d'inverser le caractère courant *i* appartenant à une clé *k* de taille *t*, par un autre caractère choisi parmi l'alphabet disponible est donnée par l'algorithme suivant :

Function 5.2. Voisinage(k, i, n)

```
car← random(Alpha,26)
Si n =26
   x ← k[i]
   j ← 1
   TantQue k[j] ≠ x  faire
         j ← j + 1
   FTantQue
FinSi
   k[j]← car
```

A noter que la modification opérée au sein de la clé est retenue seulement si la nouvelle clé engendre un coût relativement meilleur à son précèdent.

5.2.5 Fonction objectif

a. Etat de l'art

La fonction objectif est une mesure de l'optimalité d'une solution relative à un problème donné. Elle désigne le coût ou la qualité ; Dans notre cas, on utilise le coût, il doit donc être minimisé afin d'avoir un résultat satisfaisant[2]. La fonction de coût est étroitement liée à l'objectif du problème par une relation ou un facteur d'approximation permettant de refléter l'état de la solution. Si l'objectif est inconnu ou ne peut être atteint lors du traitement, la fonction de coût est évaluée par approximation.

Dans le contexte de la cryptanalyse, la fonction de coût évalue la pertinence d'une clé candidate pour un texte chiffré donné. Une clé dont les caractères sont corrects en majorité, produira un texte déchiffré proche du réel et aura donc, un faible coût proche du minimum global de l'espace de recherche. La manière d'évaluation d'une clé consiste à une analyse du texte déchiffré, extraire son profit et le comparer à un profit référentiel d'une source de données, de préférence un corpus ; autant que le profit est proche du texte analysé, le coût sera minimal. Cette méthode n'est efficace que si le référentiel comporte des propriétés similaires au texte clair obtenu (notamment la linguistique, le style, la syntaxe, etc.) et que ce dernier est de taille importante afin que l'analyse statistique sera signifiante. (Jackobsen, 1995) déclare que la fonction du coût ne permet de donner un résultat acceptable qu'en présence d'échantillons volumineux de textes chiffrés.

En littérature, la fonction de coût d'une clé de déchiffrement k a été formulée de différentes manières. Le premier modèle de cette fonction a été élaboré par (Spillman, Janssen, Nelson, & Kepner, 1993) dans une implémentation des Algorithmes Génétiques pour la cryptanalyse de certains chiffrements classique. L'équation correspondante a la forme suivante :

$$fitness = (1 - \sum_{i=1}^{26}\{|SF[i] - DF[i]| + \sum_{j=1}^{26}|SDF[i,j] - DDF[i,j]|\}/4)^8 \quad (5.1)$$

avec *SF*, *DF* : respectivement le texte clair observé et le référentiel de comparaison associé aux unigrams, bigrams et trigrams. La fonction a été élaborée de façon qu'elle soit sensible pour les petites différences (puissance 8) et moins sensible pour les grandes différences (division par 4). Les résultats de la cryptanalyse n'étaient pas avantageux vus les moyens de l'époque mais l'étude a permis de mettre en œuvre des concepts liés à la modélisation, notamment les paramètres de sélection et mutation.

Une année après, (Clark, 1994) a proposé une fonction générale, moins complexe, pour la cryptanalyse des chiffrements par substitution monoalphabétique de la forme :

$$cost(k) = \alpha \sum_{i \in A}|K^u_{(i)} - D^u_{(i)}| + \beta \sum_{i,j \in A}|K^b_{(i,j)} - D^b_{(i,j)}| + \gamma \sum_{i,j,k \in A}|K^t_{(i,j,k)} - D^t_{(i,j,k)}| \quad (5.2)$$

2. Chaque problème combinatoire possède un objectif associé à une fonction à minimiser ou maximiser. Habituellement, le terme minimiser est associé à une fonction de *coût*. Dans le cas des problèmes économiques où l'objectif est un profit à maximiser, on parle de fonction de pertinence (*fitness*).

où *K*, *D* : désignent respectivement le texte clair observé et le référentiel de comparaison associés aux unigrams, bigrams et trigrams ; α, β, γ : des coefficients de pondération déterminées par l'expérimentation. Dans sa contribution, il souligne la difficulté de son application vu sa complexité ($O(n)^3$), ainsi qu'aux complications liées aux trigrammes (dont les tables statistiques correspondantes sont rares et variés). De même, dans ces tables on ne trouve pas la totalité des caractères de l'alphabet, ce qui retarde la convergence vers l'objectif. Il réalisa donc des tests afin de déterminer les meilleures valeurs des coefficients α, β, γ. Il aboutit en fin de compte à la relation α+β+γ=1 et ce, à l'issue de tests utilisant 100 messages variés. Lors de ses expérimentations, Il utilisa, pour l'attaque des chiffrements monoalphabétique, les bigrams seulement, soit (α,β,γ)=(0,1,0) en se basant sur les travaux de (Forsyth & Savafi, 1993). Il démontra la supériorité de la recherche Tabou par rapport aux Algorithmes Génétiques et le Recuit Simulé.

La même fonction a été utilisée par (Dimovski & Gligoroski, 2003) pour la cryptanalyse du chiffrement par transposition avec un le jeu de coefficients (α,β,γ)=(0,1,0). Son argumentation quant à ce choix est que la fréquence des unigrams ne change pas au cours du processus du chiffrement, donc ignorée. Les trigrams ont été également ignorés par manque de tables statistiques efficaces. De même, leur usage accroit la complexité de la fonction du coût vers $O(n^3)$ au lieu de $O(n^2)$ si on utilise seulement les bigrams et ce, au dépens de leur performance. Les tests ont été opérés sur des textes de 200 à 1000 caractères, chiffrés par transposition. Les résultats ont révélé la performance des Algorithmes Génétiques par rapport aux metaheuristiques classiques Recuit simulé et Tabou Search.

Cette même fonction a été encore reprise plus tard par (Toemeh & Arumugam, 2008) dans une nouvelle approche basée sur les Algorithmes Génétiques pour la recherche de clés dans la cryptanalyse de chiffrements polyalphabétiques. Les tests ont permis de révéler la quasi-totalité des caractères de la clé pour des textes de 1000 caractères en soulignant une performance relative en temps d'exécution.

Dans leurs travaux relatifs à la cryptanalyse des chiffrements polyalphabétiques, (Clark & Dawson, 1997) ont proposé une nouvelle fonction de coût définie par l'équation suivante :

$$F(k) = w1 \sum_{i=1}^{N} (K_1[i] - D_1[i])^2 + w2 \sum_{i,j=1}^{N} (K_2[i,j] - D_2[i,j])^2 + w3 \sum_{i,j,k=1}^{N} (K_3[i,j,k] - D_3[i,j,k])^2 \qquad (5.3)$$

où *K*, *D* : désignent respectivement le texte clair observé et le référentiel de comparaison associés aux unigrams, bigrams et trigrams ; w1, w2 et w3 : des coefficients de pondération déterminées par l'expérimentation. Les tests préliminaires utilisant les Algorithmes Génétiques ont permis de fixer les valeurs optimales des coefficients de pondération : deux alternatives ont été définies (w1,w2,w3)=(0.4,0.6,0) et (0.2,0.3,0.5), soit en général w1+w2+w3=1. Ces valeurs ont été injectées dans le processus de la cryptanalyse qui a permis de découvrir 90% des chiffrements pour des textes dépassant les 600 caractères.

Dans leurs travaux de cryptanalyse du chiffrement par substitution (Uddin & Youssef, 2006b), ont utilisé une fonction moins compliquée, comportant les unigrams et bigrams seulement, définie de la façon suivante :

$$\text{cost(k)} = \lambda 1 \sum_{c \in \{A,B,..,Z\}} \left| R^{u}_{(c)} - DK^{u}_{(i)} \right| + \lambda 2 \sum_{c1,c2 \in \{A,B,..,Z\}} \left| R^{b}_{(c1,c2)} - DK^{b}_{(c1,c2)} \right| \qquad (5.4)$$

où λ1 et λ2 des coefficients de poids ayant les valeurs (1,0) et (0,1) et *R*, *DK*, respectivement le texte clair observé et le référentiel de comparaison associés aux unigrams et bigrams. Les tests réalisés à l'aide de la technique de Colonie de Fourmis ont permis d'obtenir 50% et 90% resp. de texte clair en utilisant les unigrams et bigrams séparément et ce, avec des textes dépassant les 600 caractères.

(Giddy & Savafi, 1994) démontrent que la normalisation des n-grams améliore considérablement la qualité des chiffrements. Ils proposent donc, la fonction de coût suivante :

$$cost(k) = \propto \frac{\sum_{i,j \in A} \left| K_{(i)}^{uni} - D_{(i)}^{uni} \right|}{K_{(i)}^{uni}} + \beta \frac{\sum_{i,j \in A} \left| K_{(i,j)}^{bi} - D_{(i,j)}^{bi} \right|}{K_{(i,j)}^{bi} + \varepsilon} + \gamma \frac{\sum_{i,j,k \in A} \left| K_{(i,j,k)}^{tri} - D_{(i,j,k)}^{tri} \right|}{K_{(i,j,k)}^{tri} + \varepsilon} \quad (5.5)$$

où ε désigne une quantité insignifiante évitant la division par zéro.

Dans le même contexte et, afin de contourner les difficultés relatives aux n-grams, (Russel, Clark, & Stepny, 2003) ont proposé une fonction de coût basée sur un dictionnaire qui détecte la présence des sous-chaines (mots ou syllabes) au sein du texte déchiffré. Le raisonnement est que si un mot valide est détecté, il y a de fortes chances pour que ses caractères correspondent à ceux de la clé cherchée. L'idée est inspirée de la technique du *mot probable*. La fonction ainsi définie est illustrée par l'équation suivante :

$$dict(k) = \frac{1}{L} \sum_d d^2 N_d \quad (5.6)$$

où L : la taille du texte chiffré, N_d, le nombre de mots de longueur d existants dans le dictionnaire et présents dans le texte déchiffré par la clé k. Cette fonction permet d'éliminer les mots corrects générés par coïncidence au cours du déchiffrement et qui n'appartiennent pas au dictionnaire.

(Carter & Magoc, 2007) ont utilisé une fonction similaire à celle définie en (5.2) pour la cryptanalyse des chiffrements par substitution. Diverses heuristiques classiques et modernes ont été testées pour le but de souligner les avantages et les caractéristiques de chaque heuristique pour la résolution de ce type de problèmes.

b. Choix

Il est difficile de comparer la performance relative aux alternatives ainsi décrites dans la littérature. Ainsi, (Giddy & Savafi, 1994) critiquent l'absence de normalisation de la fonction proposée par (Forsyth & Savafi, 1993) et qu'elle apparait limitée à l'environnement de test. Le modèle de (Spillman, Janssen, Nelson, & Kepner, 1993) présente des arguments injustifiés et apparaît compliqué en matière de calcul, tandis que le modèle proposé par Clark nécessite un paramétrage critique pour donner de résultats satisfaisants.

Dans notre cas, on se contente d'utiliser cette alternative (éq. 5.2) car elle a été recommandée par la plupart des recherches mentionnées et moyennant quelques modifications des paramètres de la manière suivante :

- Pour les cas de la substitution monoalphabétique et le chiffrement par blocs, le coefficient γ est fixé à zéro car les statistiques des trigrams sont limitées à certains caractères de

l'alphabet seulement et ne peuvent fournir des résultats raisonnables quant aux caractéristiques du langage utilisé,

- Pour les cas de la substitution polyalphabétique et la permutation, le coefficient α est fixé à zéro également, car ces types de chiffrement sont insensibles à l'analyse des fréquences des unigrams.

Les valeurs des paramètres retenus sont inspirées de la littérature et justifiées lors des expérimentations.

5.2.6 Déchiffrement & évaluation

En partant d'un texte chiffré quelconque et d'une clé dont la taille a été préalablement définie, l'opération de cryptanalyse n'est autre qu'un déchiffrement ordinaire du texte selon la technique de chiffrement considérée. Le procédé du déchiffrement est présenté par le pseudo-code illustré dans la fonction 5.3.

Fonction 5.3. Decrypt(Cipher, Plain, k)

```
Si <Substitution mono>
   Pour i ← 1 à n faire
       j ← 1
       TantQue Cipher[i]≠ cle[1,j]
           j← j+1
       FintantQue
       Plain[i] ← cle[2,j]
   FinPour
FinSi
Si <Substitution poly/ substitution par blocs>
   Pour i ← 1 à n faire
       Plain[i] ← (cipher[j] - cipher[cle[j%ot]] +26)%26
   FinPour
FinSi
Si <Transpo>
    Pour i ← 1 à n-t pas t faire
        Pour j ← i à i+t faire
            Plain[j] ← cipher[k[j]]
        FinPour
    FinPour
FinSi
```

L'évaluation consiste à quantifier le texte déchiffré afin de déduire son coût. Le coût est calculé à partir de la fonction définie au §5.2.5/b en utilisant un corpus et des tables de statistiques n-grams. La valeur du coût renseigne sur l'état de la clé utilisée pour le déchiffrement : autant que le coût est proche de 0, la clé est proche de l'optimal. Afin de garder une homogénéité dans les

structures, la taille des tables unigrams, bigrams et trigrams a été fixée à 26 éléments et les statistiques relatives au corpus sont insérées dans le champ *Cfreq*.

La fonction 5.4 illustre le pseudo-code qui décrit le processus de calcul du coût du texte. La première étape de ce processus consiste à calculer la fréquence d'apparition des caractères, bigrams et trigrams du texte dont les valeurs seront insérées dans les champs *Tstat* respectives des tables n-grams. La deuxième étape consiste à calculer la différence absolue des valeurs *Cstat* et *Tstat* qui sera affectée des coefficients de pondération.

Fonction 5.4. Cout(Plain, α,β,γ)

```
Structure :Plain[n], Ngram:{Alpha[3],Cstat,Tstat}
          Ugram, Bgram, Tgram: Ngram[26]
Pour i ← 1 à n-2 faire
    j← 1
    TantQue j < 26
        Si Plain[i]= Ugram[j].Alpha[1] alors Ugram[i]. Tstat ++ FinSi
        Si Plain [i]= Bgram[j].Alpha[1] & Cipher[i+1]= Bgram[j].Alpha[2] alors
            Bgram[i]. Tstat ++ FinSi
        Si Plain [i]= Tgram[j].Alpha[1] & Plain [i+1]= Bgram[j].Alpha[2] & Plain
            [i+2]= Bgram[j].Alpha[3] alors Bgram[i]. Tstat ++ FinSi
        j← j+1
    FintantQue
FinPour
Pour i ← 1 à 26 faire
    UCost ← UCost+ |Ugram[i].Cstat- Ugram[i].Tstat / n|
    Bcost ← Bcost+ |Ugram[i]. Cstat - Bgram[i]. Tstat / n|
    Tcost ← Tcost+ |Ugram[i]. Cstat - Tgram[i]. Tstat / n|
FinPour
Cost ← α*UCost+ β*Bcost+γ*Tcost
```

5.2.7 Critère d'arrêt & minima local

Le test d'arrêt est un repère de maitrise du processus de traitement. A chaque itération, il intervient pour juger la qualité des solutions produites et décide de l'arrêt ou non de l'exploration de l'espace de recherche. Les critères d'arrêt utilisés dans l'implémentation sont au nombre de trois. L'arrêt est engagé si l'un d'eux a été atteint :

- Un nombre fixé *d'itérations* (ou générations lors de l'usage des heuristiques à population).
- Une durée fixe de *temps* de calcul imposée.
- Absence *d'amélioration* de la solution après un nombre déterminé d'itérations.

5.3 Complexité

La complexité engendre une estimation du nombre d'opérations élémentaires à accomplir pour résoudre les instances du problème. Les travaux dans ce domaine ont permis de classer les problèmes en fonction de leur complexité (Papadimitriou C., 1994). Il en existe cependant un très grand nombre de classes[3] dont les plus connues sont celles comme définies sur la machine de Turing. Les problèmes combinatoires dont la cryptanalyse des chiffrements faisait partie sont classés comme NP ou NP-difficiles; donc, pour les problèmes de taille importante, il est souvent irraisonnable d'employer des algorithmes exacts ; il fallait recourir aux metaheuristiques afin d'obtenir des solutions acceptables (Clark & Dawson, 1997).

Avec un alphabet *A* de cardinal *n* (avec n=26 en général) et *K*, l'ensemble de clés *k* de taille *t* chacune ; on a :

- Pour le chiffrement monoalphabétique : soit *card(k)=t=n*, donc le nombre de clés maximal à envisager lors d'une attaque est une permutation sans répétition dans n, soit n !
- Pour un chiffrement polyalphabétique ou par transposition: soit *car(k)=t < n*, le nombre maximal de clé à envisager est une combinaison de *t* caractères parmi *n* sans ordre prédéfini et sans répétitions $C_n^t = \binom{n+t-1}{t} = \frac{(n+t-1)!}{t!(n-1)!}$ et avec des répétitions possibles, soit $C_n^t = \binom{n}{n-t} = \frac{(n)\,!}{(n-t)\,!}$
- Pour un chiffrement par blocs : soit *card(k)=t<n* avec un nombre *b* de clés possibles. Le nombre maximal de clés envisagé est un ensemble *b* de *t* combinaisons parmi *n* possibilités ; soit $b\,\frac{(n+t-1)\,!}{t\,!(n-1)\,!}$ possibilités avec répétitions et $b\,\frac{(n)\,!}{(n-t)\,!}$ possibilités sans répétitions.

En général, la complexité d'une fonction de coût à *g* termes est de l'ordre de $O(N^g)$ avec *N* la taille de l'alphabet et *g*, les facteurs n-grams. On peut cependant aller jusqu'au 3-grams si ces derniers possèdent des statistiques fiables, ce qui représente un calcul énorme lors de l'usage des méthodes exactes.

5.4 Implémentation

5.4.1 Algorithme de descente

L'algorithme consiste à chaque itération, à échanger un caractère de la clé avec un autre choisi parmi l'alphabet disponible et ce, d'une manière permettant d'améliorer la fonction du cout. L'Algorithme 5.5 illustre les étapes principales de ce processus. Le traitement débute par la détermination de la longueur de la clé à l'aide de *l'indice* de *coïncidence* du texte chiffré *Cipher*, puis lui affecter un coût maximal. A chaque itération, un caractère de la clé est substitué à un autre parmi ses voisins, qualifié de meilleur après déchiffrement et calcul du coût du texte résultat *Plain* en utilisant *Tstat* et *Cstat* resp. les statistiques de fréquence de caractères du texte et du corpus utilisé. Le résultat consiste en une clé optimale *k** et le texte clair correspondant $Plain_{k^*}[n]$.

3. http ://www.complexityzoo.com/

Algorithme 5.5. Cryptanalyse_classique_recherche_Locale

Entrée: *Cipher[n], Plain[n], Alpha[26], CoutMax*
Sortie: k^*, $Plain_{k^*}[n]$
IndiceCoincidence← FriedmanTest(*Cipher,k,Alpha*)
$k^*.Cout \leftarrow CoutMax$
Repeter
  $Decrypt(Cipher,Plain_k,k)$
  $k.Cout \leftarrow cout(plain_k)$
  Si $k.Cout < k^*.Cout$
    $k^* \leftarrow k$
    $k^*.Cout \leftarrow k.Cout$
  FinSi
  $k \leftarrow V(k,random(p),p)$
Jusqu'à <*critère d'arrêt*>

L'inconvénient de la descente réside dans les minima locaux qui peuvent apparaître dans les circonstances suivantes :

- Un voisinage ayant donné un résultat meilleur doit être gardé dans sa position au niveau de la clé. Cependant, il est susceptible de donner un résultat encore mieux s'il a été utilisé dans un autre endroit de la clé.
- Le processus peut d'arrêter prématurément si aucun voisinage ne peut fournir un résultat meilleur. Cette règle n'est pas toujours vérifiée en cryptanalyse étant donné qu'une clé ayant un coût optimal ne peut fournir forcément un texte de meilleure qualité.

5.4.2 Algorithmes BFS

L'algorithme BFS (Algorithme 5.6) utilise une liste F contenant les caractères de l'alphabet en attente d'utilisation, initialement vide. A chaque itération, un caractère i de la clé k est examiné, déterminer son voisinage $V(i)$ à l'aide de la fonction *Generer_Voisinage(V,i)*, le fusionner dans F d'une manière ordonnée selon le coût de chaque caractère. Ensuite, parcourir cette liste l'aide de la fonction *Defiler*(*F.element*) afin de déterminer un remplaçant de i parmi les caractères de F n'ayant pas été utilisés en réduisant le coût total de la clé k.

Selon le principe de l'algorithme, le voisinage sélectionné de i correspond au meilleur élément parmi l'ensemble de ses voisins. En cryptanalyse et, contrairement au parcours dans un graphe, la notion de voisin est imprévisible car tous les caractères de l'alphabet utilisé peuvent être considérés comme des voisins. La recherche de voisinage peut être décrite de la manière illustrée par le pseudo-code de la fonction 5.7.

Algorithme 5.6. Cryptanalyse_classique_BFS

Entrée: *Cipher[n], Tstat, Lstat, Alpha[26], V[26],*
F[26,3] : File de priorité <element,etat>
CoutMax
Sortie: *k**: clé optimale
$Plain_{k*}[n]$: Texte clair de taille n caractères relatif à la clé k*
IndiceCoincidence← FriedmanTest($Cipher^n$,*Alpha(),k*)
Decrypt(Cipher,$Plain_k$*,k)*
k.Cout* ← *CoutMax*
i ← *1*
Generer_Voisinage(V,i)
enfiler(F,V)
k_essai←*k**
TantQue *i* ≤ *t*
  TantQue *F.element.etat*=.F.
    Defiler(F.element)
  FTantQue
  F.element.etat ← .T.
  k_essai[i]←*F.element*
  Decrypt(Cipher,$Plain_{k\_essai}$*,k_essai)*
  Si *k_essai.Cout* < *k*.Cout*
    *k** ← *k_essai*
  FinSi
  i ← *i* + *1*
  Generer_Voisinage(V,i)
  *enfiler(F,*V*)*
FinTantQue
*k**← *k*

Fonction 5.7. Generer_Voisinage(V,i)

Pour *j* ← 1 **à** *26* **faire**
  Si *Alpha[j]* ≠ *i*
    V[i]← *Alpha[j]*
  FinSi
FPour

La fonction *enfiler(F,V)* fusionne les éléments de la liste V avec ceux de la liste F en évitant l'écrasement en ignorant les éléments identiques. La fonction 5.8 illustre ce principe.

Fonction 5.8 Enfiler_element(F,V)

```
i ← 1 ; j ← 1
TantQue j ≤26 et V[i] ≠ Ø faire
    Si F[j] = Ø
       F[j] ← V[i]
       j ← 1
       i ← i+1
    FinSi
    Si F[j] = V[i]
       i ← i+1
       j← 1
    FinSi
    Si F[j] ≠ V[i] alors j ← j+1 FinSi
FTantQue
```

5.4.3 Algorithmes de Séparation et évaluation

L'algorithme de séparation et évaluation (algorithme 5.9) nécessite un espace de recherche arborescent. Chaque niveau de l'arbre doit contenir l'ensemble des caractères de l'alphabet. A chaque itération, qui correspond à l'insertion d'un nouveau caractère dans le corps de la clé, ce caractère est choisi parmi l'ensemble des caractères en vue. Le nombre de niveaux de l'arbre doit être égal à la taille de la clé. La technique présente l'avantage d'ignorer les branches de faible qualité, néanmoins, le coût doit être évalué au préalable, ce qui représente un travail énorme en matière de ressources.

Algorithme 5.9. Cryptanalyse_classique_Séparation_évaluation

```
Entrée: Cipher[n], Tstat, Cstat, Alpha[26], V[t,25]
        CoutMax, k[t_max]
Sortie: k*, Plain_k*[n]
IndiceCoincidence← FriedmanTest( Cipher,Alpha,k)
k*.Cout ← CoutMax
Pour i ← 1 à 25 faire
  Pour j ← 1 à t faire
      k_essai[j] ← V[j,i]
      Decrypt(Cipher,Plain_k_essai,k_essai)
      k_essai.Cout ← cout(k_essai,plain_k)
      Si k_essai.Cout < k*.Cout alors k ← k_essai FinSi
  FinPour
FinPour
k*← k
```

Dans cet algorithme, la solution initiale est la valeur d'une clé aléatoire dont chaque caractère représente l'entête d'une liste comportant les autres caractères de l'alphabet. Pour toute itération,

chaque caractère est substitué à un autre de sa liste privée puis, la clé correspondante est évaluée. Si sa valeur est inférieure à la valeur de la clé avant modification, la nouvelle clé est retenue, sinon la substitution est annulée. Chaque clé *k[t]* est une chaine de t caractères possédant chacun un voisinage comportant les 25 autres caractères de l'alphabet. A chaque itération, la permutation des caractères s'opère au sein d'une clé d'essai qui sera retenue seulement si elle présente une amélioration dans le coût.

L'algorithme permet de tester toutes les possibilités de permutations, ce qui engendre une consommation assez importante de ressources.

5.4.4 Algorithmes Génétiques

Similaire à algorithmes des colonies de fourmis, les AG emploient une génération de solutions qui évoluent en même temps par interaction vers l'optimum.

Algorithme 5.10. Cryptanalyse_classique_AG

Entrée: *Cipher[n], Alpha[26]*
Cle_struc<car[t],Cout> : Structure de clé candidate
k[TaillePop] : Cle_struc
MaxGen, Nb_Seg, Taux_mut
Sortie: *k*, $Plain_{k*}[n]$*
IndiceCoincidence← FriedmanTest(*Cipher*,*Alpha*,*k*)
Pour *i* ← *1 à TaillePop* **faire**
Decrypt(Cipher,$Plain_{k[i]}$,k[i])
k[i].Cout ← *cout(k[i],$plaink_{[i]}$)*
FinPour
i ← *1*
TantQue *i* < *MaxGen*
Pour *j* ← *1 à TaillePop* **pas** *2* **faire**
k[j] ← *cross(k[j], k[j+1],Nb_seg,1)*
k[j+1] ← *cross(k[j], k[j+1],Nb_seg,2)*
FinPour
Pour *j* ← *1 à TaillePop* **faire** *k[j]* ← *mut(k[j], Taux_mut)* **FinPour**
Pour *j* ← *1 à TaillePop* **faire**
Decrypt(Cipher,$Plain_{k[j]}$,k[j])
k[j].Cout ← *cout(k[j],$plain_{Cle[k]}$)*
FinPour
i ← *i+1*
Ordonner(k,cout)
FtantQue
*k** ← *k[1]*

L'Algorithme 5.10 décrit les principales étapes d'un GA pour la cryptanalyse de chiffrements. Chaque individu de la population est assimilé à une clé candidate de *t* caractères choisis parmi un

ensemble de 26 représentant les sommets de l'espace de recherche. Les opérateurs génétiques correspondent aux échanges, décalage ou permutation de caractères. Le processus commence par une population initiale de clés générées aléatoirement. A chaque itération, la fonction *Ordonner* trie la liste des clés en fonction de leurs coûts afin de faciliter la sélection.

5.4.5 Algorithme de Colonie de Fourmis

A la différence des algorithmes précédents, l'optimisation par colonies de fourmis offre un processus de recherche basé sur plusieurs solutions qui évoluent en même temps.

Algorithme 5.11. Cryptanalyse_classique_VMMAS

Entrée: **Cipher***[n], Alpha[26]* : Alphabet (A..Z)
Cle_struc<*car[t],ph,cout*> : Structure de clé candidate
k[maxAnt], k, Cle_essai* : *Cle_struc ; MinPh, MaxPh, periode_evap, a, β, ϱ*
Sortie: *k**: clé optimale
$Plain_{k*}[n]$: Texte clair de taille n caractères relatif à la clé k*
IndiceCoincidence← FriedmanTest(*Cipher,Alpha,k*)
Pour *i* ← *1 à maxAnt* **faire**
  Pour *j* ← *1 à t* **faire**
    k[i].car ← aleatoire(Alpha)
    k[i].ph ← MaxPh
  FinPour
  $k.Cout \leftarrow Cout(k,plain_k)$
FinPour
TantQ <*critère_arrêt_non_satisfait*>
Pour *i* ← *1 à maxAnt* **faire**
  HamiltonianPath ← *1*
  j ← *1*
  TantQue *(j < t)* **&** *(HamiltonianPath=1)* **faire**
    Pour *c* ← *1 à 25* **faire**
      Cle_essai.car[j] ← *Alpha(c)*
      $Decrypt(Cipher,Plain_{k\_essai},k\_essai)$
      $Cle\_essai.Cout \leftarrow cout(Cle\_essai,plain_{Cle\_essai})$
      Si *Cle_essai.Cout < k[i].Cout* alors *k[i]* ← *Cle_essai*
      Sinon *HamiltonianPath* ← *0* **FinSi**
    FinPour
    Si *HamiltonianPath* alors *MiseAjourPh(k[i],Minph,Maxph))* **FinSi**
    Si *period_evap* alors *EvapPh(k[i])* **FinSi**
  FinTantQue
  Si *k[i].Cout < k*.Cout* alors *k*.Cout* ← *k[i].Cout* **FinSi**
FinPour
FTanQ

L'algorithme 5.11 présente la version VMMAS qui utilise une colonie de *maxAnts* fourmis ; chacune d'elles représente une clé candidate *k*. L'espace de recherche *Alpha* comporte 26 sommets qui désignent les caractères que peut prendre une clé *k* de taille *t*. Le déplacement d'une fourmi *i* d'un sommet *a* au sommet *b* correspond à la substitution du caractère *a* de la clé k_i par un autre *b*. Ce dernier est choisi à base de la quantité de phéromone *ph* présent sur l'arc (a,b) qui se traduit par le coût de la clé correspondante. La progression de chaque fourmi évolue tant que la variable booléenne *HamiltonianPath* est vraie, ce qui correspond au dépôt d'une quantité de *ph* sur l'arc traversé par la fourmi à l'aide de la fonction *MiseAjourPh*. La fonction *Evap* régresse le phéromone périodiquement sur l'ensemble des arcs du graphe. Vu le nombre important de paramètres, l'algorithme consomme d'avantages de ressources, néanmoins, il progresse vers la bonne direction car toute fourmi ayant été bloqué en un minima local est simplement éliminée.

5.4.6 Algorithme d'Essaims particulaires

Comme toute heuristique à population, l'OEP utilise une population d'essaims assimilée à une liste de clés candidates de déchiffrement. La position x_i de chaque individu *i* de l'essaim représente le coût $Cout_{k[i]}$ de la clé associée *k[i]*. L'algotithme 5.12 illustre la cryptanalyse par OEP.

Algorithme 5.12. Cryptanalyse_classique_OEP

Entrée: *Cipher[n], Alpha[26]* : Alphabet (A..Z)
Cle_struc<*car[t],cout*> : Structure de clé candidate
k[TailleSwarm] : Ensemble de clés candidates
CoutMax
Sortie: *k*, Plain*$_{k*}$*[n]*
IndiceCoincidence← FriedmanTest(*Cipher,Alpha,k*)
Générer x_i, v_i *(i=1.. TailleSwarm)* et c_g
k.Cout* ← *CoutMax*
Init_OEP()
Répéter
  Pour *j* ← *1 à TailleCle* **faire**
    g ← *0*
    Pour *i* ← *1 à TailleSwarm* **faire**
      k[i].car[j] ← *aleatoire(Alpha)*
      Decrypt(Cipher,Plain$_{k[i]}$*,k[i])*
      k[i].Cout ← *Cout(k[i],plain*$_{k[i]}$*)*
      Si *k[i].Cout* < *k*.Cout*
        k.Cout* ← *k[i].Cout*
        g ← *1*
      FinSi
    FinPour
    Si *(g = 1)* **Pour** *i* ← *1 à TailleSwarm* **faire** *k[i]* ← *k** **FinPour** **FinSi**
  FinPour
Jusqu'à <*critère d'arrêt*>

Le principe du voisinage est limité au meilleur individu k^* de l'essaim. La vitesse de chaque individu $v_{k[i]}$ correspond à la différence du coût $Cout_{k[i]}$ de la clé correspondante et du coût $Cout_{k^*}$ de la meilleur clé obtenue. A chaque itération, la différence ($Cout_{k[i]}$ - $Cout_{k^*}$) est calculée pour chaque $k[i]$. Cette clé sera déplacée à la position x^* si $v_{k[i]}$ est supérieure à une constante donnée c_g. Le processus prend fin après un nombre fini d'itérations ou en absence d'amélioration durant une période donnée.

La fonction *init_OEP* décrite par l'algorithme 5.13 permet d'initialiser l'essaim de particules. Les caractères de chaque clé sont choisis aléatoirement parmi les éléments de A Chaque clé est évaluée ; celle ayant le coût optimal est classé comme meilleur voisinage.

Algorithme 5.13. Init_OEP()

Pour *i* ← *1 à TailleSwarm* **faire**
  Pour *j* ← *1 à t* **faire**
    k[i].car[j] ← *aleatoire(Alpha)*
    Decrypt(Cipher,Plain$_{k[i]}$*,k[i])*
    k[i].Cout ← *Cout(k[i],plain*$_{k[i]}$*)*
    Si *(k[i].Cout - k*.Cout)* > c_g alors *k*.Cout* ← *k[i].Cout* **FinSi**
  FinPour
FinPour

5.5 Conclusion

L'étude ainsi faite en ce chapitre a permis de modéliser les diverses metaheuristiques d'une manière de pouvoir procéder aux attaques des chiffrements considérés. La base d'essai, les clés de déchiffrements et leurs voisinages, l'alphabet du langage utilisé ainsi que les textes à déchiffrer ont été modélisés à base de tableaux de taille fixe afin de pouvoir gérer le traitement d'une manière statique ce qui permet d'éviter les désagréments liés au débordement en mémoire lors de l'usage des listes et de tableaux dynamiques.

De même, chaque algorithme est doté de plus d'un critère d'arrêt, ce qui permet d'éviter les boucles infinies si une condition d'arrêt ne peut être satisfaite.

6. Les Metaheuristiques en Cryptanalyse symétrique

6.1 Introduction

La plupart des attaques aux chiffrements symétriques sont dédiées et non transposables à d'autres chiffrements. Elles nécessitent également une puissance de calcul importante vu la non-linéarité dont sont dotés les Sboxes rendant ainsi la relation entre les données d'entrée et de sortie confuse. D'autre part, la cryptanalyse de ces chiffrements peut être formulée tel un problème combinatoire ou l'espace de recherche est un ensemble de clés k de taille n dont les valeurs sont réduites à l'ensemble $\{0,1\}$. Les metaheuristiques peuvent être modélisées afin de résoudre ce problème de la même manière que la cryptanalyse classique moyennant quelques différences liées au type de données utilisées dans chaque chiffrement et à la fonction objectif utilisée pour quantifier les résultats d'attaques.

Comme les chiffrements symétriques utilisent des blocs de données binaires, l'analyse statistique de caractères ne peut être envisagée comme manière d'évaluation du résultat d'une clé, Il est donc indispensable de s'en passer de la fonction objetif utilisée en cryptographie classique et d'en aménager une autre plus adaptable à ce type de problèmes. La recherche dans ce contexte propose certaines altératives jugées acceptables ; la plupart d'entre elles sont basées sur la qualité des résultats de l'ensemble des clés utilisées dans l'attaque ; ce fait a pour conséquence de devoir utiliser un ensemble de clés en même temps, ce qui favorise l'usage des metaheuristiques à population et écarte les techniques à solution unique.

Après une évaluation des travaux relevant au problème, on présente dans le ce chapitre notre contribution qui consiste en une proposition d'une fonction objectif, inspirée de la littérature et modifiée d'une manière permettant d'en profiter le mieux de la qualité du résultat obtenu par

chaque clé candidate afin maintenir une population de clés performante durant tout le processus d'attaque.

De même, ce chapitre présente également une modélisation de trois metaheuristiques, à savoir l'Optimisation par colonies de fourmis, les algorithmes génétiques et l'optimisation par essaims particulaires quant à l'attaque des chiffrements symétriques ainsi que le codage des données et paramètres et les algorithmes associés. Les metaheuristiques classiques à solution unique n'ont pas été traités dans ce chapitre vu qu'elles ne peuvent être adaptées à ce genre de problèmes.

6.2 Attaques aux chiffrements symétriques

6.2.1 Etat de l'art

La recherche actuelle a montré que la majorité des chiffrements classiques peuvent être brisés à l'aide des techniques heuristiques ; ce qui n'est pas le cas pour les chiffrements modernes. Ceci est dû essentiellement à la structure non linéaire dont les chiffrements symétriques sont dotés. Au cours des deux dernières décennies, une progression considérable dans les travaux liés à la cryptanalyse des chiffrements symétriques a été signalée. En littérature, les attaques relevant à ce chiffrement remontent aux années 90 où (Biham & Shamir, 1991) ont proposé leur idée relative à la cryptanalyse différentielle ; elle a été testée sur le DES (Biham & Shamir, 1993b) (Biham & Shamir, 1993a) ; les expérimentations ont montré que 2^{47} textes choisis suffisent à révéler la clé de chiffrement. Ce résultat a été amélioré par (Matsui, 1994) qui proposa la cryptanalyse linéaire à l'algorithme DES et a montré que 2^{43} textes connus sont suffisants à révéler cette clé.

En pratique, l'attaquant aux chiffrements, dispose uniquement des textes chiffrés. Dans certains cas, il peut avoir accès à d'autres détails tels la langue avec laquelle les textes ont été écrits, la taille des textes, l'algorithme de chiffrement qui permet de faciliter le processus de recherche de clé. Sans ces détails, la cryptanalyse aux textes chiffrés seulement est un challenge (Biryukov & Kush, 1998) ; néanmoins l'usage des metaheuristiques dans ce contexte a été longtemps sollicitée en dernière décennie où (Bafghi & Sadeghiyan, 2004) ont utilisé dans leurs travaux les metaheuristiques basées sur les colonies de fourmis qui s'avèrent également intéressants moyennant un paramétrage correct ; D'un autre côté, (Clark, Jacob, & Stepney, 2004) ont proposé une approche basée sur les fonctions booléennes en vue d'attaque des structures non linéaires des Sboxes ; suivi de (Laskari, Meletiou, Stamatio, & Vrahatis, 2005) qui ont utilisé l'OEP pour l'attaque du SDES ; les résultats ont prouvé que l'approche est apte à briser les Sboxes. S'un autre côté, (Song J. , Zhang, Meng, & Wang, 2007) (Song J. , Zhang, Meng, & Wang, 2007) (Hernández, 2002) montrent que les algorithmes génétiques constituent un outil d'attaque efficace et prometteur pour les chiffrements de Feistel. Les travaux de (Nalini & Rao, 2007) englobent des expérimentations d'attaque aux chiffrements symétriques par diverses metaheuristiques ; Les résultats ont prouvé l'efficacité de telles techniques pour la cryptanalyse du DES et 4DES. En 2009, (Husein, Bayoumi, Holail, Hasan, & Abd El-Mageed, 2009) ont combiné les algorithmes génétiques avec la cryptanalyse différentielle pour élaborer un algorithme rapide pour l'attaque du DES ; les tests ont été opérées sur le DES avec 8 tours. (Shahzad, Siddiqui, & Khan, 2009) ont montré que le OEP est plus compétitif que les algorithmes génétiques quant à l'attaque du 4DES. Egalement, la technique OEP a été reprise par (Abde--Elmoniem, Ghali, & Hassanien, 2011) et par (Mekhaznia, Menai, & Zidani, 2013) pour l'attaque de certains chiffrements symétriques, notamment le DES ; les résultats ont permis de déduire la

majorité des bits de la clé. Les travaux de (Vimalathithan & valarmathi, 2011a) (Vimalathithan R., Valarmathi M. L, 2011b) (Vimalathithan & Valarmathi, 2012) (Jadon, Sharma, Kumar, & Bansal, 2011) (Pandey & Mishra, 2012) ont également prouvé l'efficacité d'OEP pour la réduction de l'espace de recherche de clé de divers chiffrements symétriques.

6.2.2 Stratégies d'attaques

Malgré la non linéarité que caractérise les chiffrements symétriques, les metaheuristiques ont été largement utilisées comme outils d'attaque; néanmoins, les résultats paraissent restreints et propres à chaque environnement d'expérimentation. Egalement, la manière d'attaque aux chiffrements classiques ne peut être envisagée sur les chiffrements symétriques en l'absence de relation concrète entre les textes clairs et chiffrés (Coppersmith, 1994). De même, la génération de clés ne peut être optimisée par une opération génétique car la modification d'un quelconque bit de la clé peut causer des changements imprévisibles au sein du texte résultat. Donc, l'élaboration d'une fonction objectif constitue la majeure difficulté du problème et nécessite une attention particulière.

A base de ces constatations, les stratégies adoptées pour l'attaque de chiffrements symétriques sont variées et s'articulent autour les axes suivants :

c. *Attaque par déduction de bits corrects à base d'une analyse statistique* : L'idée proposée par (Hamdani, Shafiq, & Khan, 2010) et consiste à analyser la fréquence d'apparition des bits au sein des clés ayant permis d'obtenir un meilleur déchiffrement et ce, après plusieurs exécutions. A titre d'exemple, et, suite à 3 exécutions, les clés[1] obtenues sont k_1 = 1001011, k_2 = 1110010 et k_3 = 1101001. L'addition arithmétique des bits '1' et '0' de même position ainsi que les moyennes correspondantes sont décrites dans la table 6.1.

k_1	1	0	0	1	0	1	1
k_2	1	1	1	0	0	1	0
k_3	1	1	0	1	0	0	1
Bits 1	3	2	1	2	0	2	2
Bits 0	0	1	2	1	3	1	1
Moyenne 0	1.0	0.67	0.33	0.67	0.0	0.67	0.67
Moyenne 1	0.0	0.33	0.67	0.33	1.0	0.33	0.33
1 correct	x						
0 correct					x		

Tab. 6.1. Total et moyenne de la fréquence de bits

La valeur de chaque élément de la série dépassant un seuil donné α est considéré comme un bit correct. Dans l'exemple précédent et avec α supposé égal à 0.8, les valeurs dépassant ce seuil se trouvent respectivement aux positions 1 et 5 de chaque série, mentionnées par 'x' dans le

1. On se limite à une taille réduite de la dé pour la darté de l'exemple.

tableau précédent. La conclusion est que le premier bit de la clé et le cinquième seraient probablement 1 et 0.

La même idée a été utilisée auparavant par (Shahzad, Siddiqui, & Khan, 2009) et modélisée pour l'attaque du DES à 4 tours à l'aide de la technique OEP où chaque particule représente une clé candidate. Un processus d'exécution permet de dégager certaines clés ayant une valeur de fonction objectif supérieure à un seuil fixe γ. Un comptage de bits tel défini plus haut est opéré pour chaque clé retenue, ce qui permet de fixer certains bits. Les clés retenues seront insérées comme solution initiale pour un nouveau processus d'exécution.

La fonction objectif utilisée est définie par l'équation

$$f = S/64 \tag{6.1}$$

où S désigne le nombre de bits identiques de même position entre un texte chiffré avec la bonne clé et le texte obtenu avec une clé générée au cours du test.
Les expérimentations réalisées à base d'un seuil γ variable de 0.7 à 0.8, les résultats ont permis de révéler 19 sur les 56 bits composant la clé de chiffrement 4DES.

d. *Attaque de DES par OEP binaire* : Idée de (Jadon, Sharma, Kumar, & Bansal, 2011) proposée en tant qu'amélioration de l'attaque de (Biham & Shamir, 1991) relative à la cryptanalyse différentielle. Comme cette dernière permet de révéler seulement 42 sur les 56 bits composant la clé DES et que le reste, soit 14 bits doit être reproduit par une autre technique, probablement la force brute ; L'idée de Jadon et al. a pour objectif de reproduire ces 14 bits restants à l'aide de l'OEP ce qui permet de réduire l'espace de recherche et contourne ainsi une recherche exhaustive.

   La formulation proposée consiste en un ensemble de solutions X_i où chacune représente une clé de 56 bits dont 42 premiers bits supposés connus. Tous les paires *np* des textes chiffrés utilisés en cryptanalyse différentielle pour générer les 42 premiers bits de la clé seront réintroduits dans le processus de génération des 14 bits restants. Les paires np_{X_i} de ces textes qui satisfont les textes claires connus seront retenus ; soit cnp_{X_i} leur nombre. La fonction objectif utilisée est basée ces statistiques et, est définie par la relation (6.2) :

$$f(X_i) = np - cnp_{X_i} \tag{6.2}$$

   La solution optimale X_i sera obtenue à l'aide d'une valeur de f proche de zéro.

   Les expérimentations ont été opérées à base d'un paramétrage standard et un nombre réduit de d'exécution sans indication sur la nature des textes utilisés. Des résultats comparatifs OEP binaire/OEP comportant une dizaine de paires clairs/chiffrés ont été présentés. En l'absence d'un référentiel de mesure de performance, la conclusion s'est limitée à montrer que l'idée proposée peut être considérée comme un outil efficace d'attaque aux chiffrements par blocs.

e. *Attaque par ACO binaire* ; idée proposée par (Khan, Shahzad, & Khan, Cryptanalysis of Four-Rounded DES Using Ant Colony Optimization, 2010) en utilisant une version binaire de colonie de fourmis baptisée *BACO*. Pour la cryptanalyse du DES à 4 tours. Chaque fourmi représente un bloc de texte de 64 bits ; chaque bit réfère à une position dans l'espace de recherche. Cet espace est composé de deux lignes de 64 sommets chacune ; les sommets de

la première sont tous identiques et égaux à '0', l'autre ligne contient uniquement des '1'. A chaque pas parmi 64, la fourmi choisira un sommet sur l'une ou l'autre des deux lignes, ce qui correspond à un bit du bloc de texte chiffré ; le choix dépend de la quantité de phéromone sur le sommet en question. La solution de départ n'est pas aléatoire, chaque bit i du bloc candidat C'_i est défini par la relation $C'_i = M'_i \oplus C^S_i$ où M et C^s désignent respectivement un texte clair et son correspondant chiffré. La construction de la population initiale est générée à base de plusieurs paires de textes clairs/chiffrés. La fonction objectif considérée est définie par la relation :

$$Fitness = \varrho/64 \tag{6.3}$$

avec ϱ, désigne le nombre des bits égaux de la même position entre C' et C^S

Le processus de traitement porte sur plusieurs échantillons de blocs de textes clairs/chiffrés. Les meilleurs résultats de chaque test vont être examinés pour une éventuelle fixation de certains bits selon l'idée proposée en (a) et seront injectés comme solution initiale pour une nouvelle exécution. Le processus se déroule jusqu'à épuisement de tous les bits du bloc.

Deux séries de tests ont été opérés ; le premier relatif au DES à 4 tours avec un paramétrage similaire à celui présenté en (a) et des résultats similaires également. La deuxième série, consiste en l'attaque de chaque tour séparément. Avec une colonie de 10.000 fourmis, les résultats ont permis de révéler un maximum de 19 bits, néanmoins, la conclusion porte sur les problèmes liées au paramétrage de l'algorithme afin d'améliorer ses performances sans explication quant au choix des valeurs de paramètres utilisées.

f. Attaque à texte chiffré connu par OEP ; idée proposée par (Wafaa, Ghali, Hassanien, & Abraham, 2011) et qui consiste à déterminer le maximum de bits de la clé de 4DES et DES. Le principe de l'approche est le suivant :

Pour le 4DES, l'approche est simple : chaque particule représente un bloc de texte ; l'attaque se fait de la même manière que la cryptanalyse classique, chaque bloc de texte décrypté est évalué à base de la fonction objectif définie par l'équation suivante :

$$f(k) = \alpha \sum_{i \in A} \left| K^u_{(i)} - D^u_{(i)} \right| + \beta \sum_{i,j \in A} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| \tag{6.4}$$

où A, désigne l'alphabet ordinaire ; K et D, les statistiques du langage utilisé et celles du texte décrypté et α, β des coefficients de pondération, avec $\alpha+\beta=1$.

Les expérimentations réalisées en utilisant un standard de paramètres OEP avec une population de 24 particule et un nombre de 50 d'itérations ont permis de révéler un maximum de 36 bits pour un chiffrement d'environ 1000 bits.

Pour le DES, l'attaque par OEP a été limitée à texte clair connu et consiste en l'examen des différences entre les entrées des Sboxes du premier tour et à la sortie des Sboxes du dernier tour et se déroule selon les étapes suivantes :

- générer un ensemble de paires de chiffré-connu/clair-généré à l'entrée de la Sboxe du premier tour et à la sortie du dernier tour.

- évaluer le nombre de bits identiques entre les paires ainsi définis en utilisant la fonction objectif suivante :

$$F_2 = S/48 \tag{6.5}$$

où S désigne le nombre de bits identiques entre les expressions $E(R_0) \oplus k_1$ et $E(R_{15}) \oplus k_{16}$

- En utilisant les paires ayant une évaluation optimale, déterminer les 8 meilleures différences spécifiques à chaque Sboxe.
- Finalement et, compte tenu des résultats obtenus à l'étape précédentes et de certaines spécificités relatives au décalage des bits au sein des sous-clés de DES, déduire la plupart des bits correctes de la clé principale.

Les expérimentations réalisées à base d'une population de 500 particules et un traitement de 200 itérations ont permis de découvrir un maximum de 39 parmi les 56 bits de la clé de chiffrement. Les résultats ainsi obtenus ont été considérés comme un succès et que l'approche proposée est en mesure de révéler d'avantage de bits.

g. Une autre alternative d'attaque généralisée aux chiffrements symétriques, utilisant diverses metaheuristiques où la fonction objectif est basée sur l'analyse de fréquence de caractères de la même manière que la cryptanalyse classique. Cette attaque été objet de recherche dans plusieurs travaux à savoir :

- Les travaux de (Nalini & Rao, 2007) relatives aux heuristiques classiques englobent une synthèse d'attaque aux chiffrements symétriques. Ils ont montré que pour un texte de moins de 4000 caractères chiffré avec le SDES, moins de 30 secondes suffisent à la recherche tabou et le recuit simulé pour révéler complètement une clé de 10 bits. Sous les mêmes conditions, l'attaque au chiffrement DES a permis de révéler 16 bits seulement et ce, en moins d'une minute.
- Une approche similaire a été proposé par (Vimalathithan R., Valarmathi M. L, 2011b) pour l'attaque du DES. Les expérimentations réalisées à base d'une version hybride construite à base de l'OEP et les algorithmes génétiques, baptisée BPSO a permis de révéler 26 bits et ce pour traitement de 1000 itérations.
- Les algorithmes génétiques et l'OEP binaire ont été également utilisés par (Sharma, Pathak, & Sharma, Breaking of Simplified Data Encryption Standard using Binary Particle Swarm Optimization, 2012b) pour l'attaque du SDES. Les tests ont montré que la clé peut être révélée complètement à base d'un texte de 1200 caractères.
- Egalement, une approche d'attaque similaire à 4DES par usage de l'OEP (Dadhich & Yadav, Swarm Intelligence and Evolutionary Computation based Cryptography and Cryptanalysis of 4-round DES algorithm, 2014) a été proposée ; Les expérimentations à base d'un traitement de 1000 itérations, une population de 40 particules et divers autres valeurs de paramètres sans justification ont permis de recouvrir 35 à 40 bits.
- Une autre approche d'attaque à SDES par les algorithmes génétiques proposée par (Teytaud & Fonlupt, 2014) ; Les expérimentations réalisées à base d'une population de 10 chromosomes et un traitement de 100 itérations ont permis de révéler la quasi totalités des bits de clé d'un chiffrement de 100.000 caractères.

– Dans le même contexte, une approche similaire a été présentée par (Gard, Varshney, & Bhardwaj, 2015) ; les expérimentations réalisés sur des textes de 1200 caractères en utilisant une population de 100 chromosomes ont permis de découvrir 9 bits (parmi 10) et ce, en 2 minutes.

6.2.3 Analyse et critiques

L'analyse de près des différents approches mentionnées montre que la plupart des attaques ciblent les versions réduites de l'algorithme DES, à savoir le DES à nombre limité de tours et DES simplifié dont la clé comporte seulement 10 bits; ce qui montre que le DES à 16 tours demeure robuste et résistant aux attaques. Le peu de travaux ayant abordé le DES (Vimalathithan R., Valarmathi M. L, 2011b) (Wafaa, Ghali, Hassanien, & Abraham, 2011) dont les expérimentations ont permis de découvrir 26 et 39 bits respectivement, ne livrent aucune indication sur la nature des chiffrements utilisés qui doivent normalement être extraits d'un corpus connu afin d'approuver la qualité des résultats.

Le deuxième point critique qu'il faut signaler concerne l'ambiguïté relative au paramétrage de l'environnement de tests utilisé. Chaque metaheuristique comporte au moyenne, une dizaine de paramètres qu'il fallait évaluer avant d'entamer une attaque. La plupart des travaux se contentent d'une évaluation à l'intuition tandis que d'autres ne les mentionnent même pas. A ce fait, on ajoute l'absence du type de matériel utilisé afin de justifier le temps d'exécution et par conséquent, la performance de l'approche considérée.

Certaines données posent également un problème ; la taille des chiffrements utilisés est parfois omise malgré son importance dans l'évaluation de la qualité des résultats. Un nombre de blocs de chiffrements important permet une meilleure approximation. De même, certains résultats ne concordent pas avec la nature des données (Teytaud & Fonlupt, 2014) ou 10 bits de la clé ont été révélés à base d'un texte de 100 caractères, par contre, 7 bits seulement ont été découverts à base d'un texte de 102400 caractères. Les Travaux de (Nalini & Rao, 2007) mentionnent une taille de texte 3856 octets, or en chiffrement symétrique, la mesure de la quantité de données se fait par bloc car chaque est traité séparément. Le travail de (Dadhich & Yadav, 2014) montre qu'avec un chiffrement de 4 octets, on peut déterminer 35 bits de la clé, ce qui est irrationnel.

Ces constats favorisent la divergence des résultats et perd de leur efficacité et compétitivité notamment en l'absence de benchmarks standard permettant de mesurer la performance des divers tests présentés ou de les reproduire en cas de besoin.

6.3 Approche proposée

6.3.1 Définition

Les divers travaux mentionnées se concordent sur l'efficacité des metaheuristiques quant à l'attaques des chiffrements et la plupart d'entre eux se concordent également sur le fait que l'analyse de fréquence ne peut être adaptée comme fonction objectif à cause de la non-linéarité et de la confusion dont les Sboxes sont dotés et que son usage a été limité à certains chiffrements réduits telle le SDES. Donc et, comme mentionné au § 6.2.2, la majorité des recherches utilisent la *déduction de bits corrects* comme meilleure manière d'évaluation d'un résultat d'attaque utilisant une metaheuristique.

Notre approche consiste donc à utiliser cette alternative comme fonction objectif moyennant quelques modifications, à savoir :

- On calcule la somme, la moyenne des bits '0' et '1' et la fonction objectif f_k des clés utilisées (telle présentée dans la table 8.1). Les résultats seront triés selon la valeur de la fonction objectif,
- On retient les n-1 premières clés comme candidates pour la prochaine itération,
- La n-ème clé est construite de manière à ce que l'ensemble de ses bits ait une moyenne supérieure à 0.5.

											f_k
k_3	0	**0**	**0**	**1**	1	**1**	**1**	**1**	**1**	1	2
k_1	**1**	**0**	**0**	**1**	**0**	**1**	**1**	0	0	**0**	3
k_4	**1**	**0**	**0**	**1**	1	0	0	**1**	**1**	1	5
k_2	**1**	1	1	0	**0**	**1**	0	**1**	**1**	**0**	5
k_5	**1**	1	**0**	**1**	**0**	0	**1**	**1**	**1**	**0**	6
ΣBits 1	4	2	1	4	2	3	3	4	4	2	
ΣBits 0	1	3	4	1	3	2	2	1	1	3	
Moy. 1	0.8	0.4	0.2	0.8	0.4	0.6	0.6	0.8	0.8	0.4	
Moy. 0	0.2	0.6	0.8	0.2	0.6	0.4	0.4	0.2	0.2	0.6	
k^*_5	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	6

Tab. 6.2. Exemple de Fréquencœ d'apparition de bits proposée

L'exemple de la table 6.2 illustre l'approche proposée. Il contient 5 clés à 10 bits, candidats pour attaque au chiffrement SDES. La moyenne des bits '0' et '1' est calculée de la même manière que celle définie dans la table 6.1. La sélection exclue la clé k_5 qui se trouve en queue du classement ; elle sera remplacée par une nouvelle clé k'_5. Notre motivation quant à cette idée est que chaque clé est évaluée au dépens des autres ; et qu'à base de cette évaluation, on peut déduire la structure théorique de la clé parfaite k^* (avec f_{k^*} proche de 10); il donc utile de l'insérer manuellement dans la liste des candidats au lieu d'attendre qu'elle soit construite par le processus de recherche. Bien entendu, cette clé peut ne pas être la meilleure, car elle est évaluée d'une manière approximative et l'idée proposée n'est qu'une manière de raccourcir le processus de recherche et de l'orienter vers l'objectif.

La fonction objectif retenue relative à la clé k mesure la différence entre les bits de même position au sein d'un bloc du texte clair et celle du bloc équivalent du texte déchiffré avec la clé k générée au cours du traitement. Elle est définie par la relation suivante :

$$f_k = 1 - S/n \tag{6.6}$$

où S désigne le nombre de bits identiques de même position et n, la taille de la clé du chiffrement utilisée. Une clé est dite relativement performante si sa fonction objectif tend vers zéro.

Il est à noter que n prendra la valeur 10 si le chiffrement est réalisé avec l'algorithme SDES et 64 si l'algorithme est DES.

6.3.2 Déchiffrement & évaluation

De la même manière que les chiffrements classiques, l'attaque part d'un bloc de texte chiffré *c*, dont la source est un bloc de taille équivalente de texte clair connu *p*, chiffré avec une clé *k* connue également. Après génération d'un ensemble de clés candidates, l'opération de cryptanalyse consiste, et pour chaque itération, à appliquer l'algorithme de chiffrement considéré (étant donné qu'il est réversible) au texte *c* en utilisant l'ensemble des clés candidates constituant la population de la metaheuristique utilisée. Le procédé du déchiffrement est présenté par le pseudo-code illustré dans la fonction 6.1 qui reçoit en entrée le bloc chiffré *cipher* de taille variable (8 bits s'il s'agit de l'algorithme SDES et 64 pour le DES à n tours). Ce bloc subira la transformation initiale IP puis divisé en deux sous-blocs *L* et *R*.. A chaque tour, il subira les transformations liées à la fonction *f* relative au Tour DES (fonction 2.5) à savoir l'expansion *E*, la sélection *S* et la permutation *P*. Finalement, les deux sous-blocs seront concaténés avant application d'une dernière transformation IP^{-1}.

Fonction 6.1. Decrypt_DES(Cipher, Plain, n)

Structure :*L[n/2], LX[n/2], R[n/2],T[n]*

Si <SDES>
  nbTours ← 2 ; *Sboxe* ← $\text{Sboxe}_{\text{SDES}}$;; *IP* ← IP_{SDES}
  IP^{-1}← $\text{IP}^{-1}{}_{\text{SDES}}$; *E*← E_{SDES}; *S*← S_{SDES}; *BlockSize* ← *8*
Sinon <*nDES*>
  nbTours ← *n* ; *Sboxe* ← $\text{Sboxe}_{\text{DES}}$;; *IP* ← IP_{DES}
  IP^{-1}← $\text{IP}^{-1}{}_{\text{DES}}$; *E*← E_{DES}; *S*← S_{SDES}; ; *BlockSize* ← *64*
FinSi
IP*(Cipher)*
Pour *i* ← *1* à *blockSize* faire
  L[i] ← *Cipher[i]*, *R[i]* ← *Cipher[BlockSize /2+i]*
Fpour
Pour *i* ← *1* à *nbTours* faire
  T ← E*(R)*
  T← T⊕k_i
  LX ← S*(T,Sboxe)*
  LX← P*(LX)*
  L ← *R*
  R ← L⊕LX
FinPour
Pour *i* ← *1* à *BlockSize /2* faire
  Plain[i] ← *L[i]*
  Plain[n/2+i] ← *R[i]*
Fpour
IP^{-1}*(Plain)*

Le bloc de sortie *Plain* sera évalué à base de la fonction objectif définie par l'équation 6.6. et illustrée par le pseudo-code de la fonction 6.2

Fonction 6.2. Cout_Texte(Plain,k,Cle_ref,n)

```
Cout ← 0
Pour i ← 1 à n faire
    Si k[i] =Cle_ref[i] alors Cout ← +1 FinSi
FinPour
```

L'évaluation permet de sélectionner un certain nombre de clés à base de leurs *couts*. Elles subiront des opérations génétiques avant d'être insérées dans la liste des candidats de la prochaine itération.

A chaque itération, la clé la moins performante sera éliminée et remplacée par une nouvelle clé *Cle_modele* générée à base des clés les mieux performants de la liste des candidats. Le processus de construction de cette clé est illustrée par le pseudo-code de la fonction 6.3. Un exemple d'une telle clé est donné par la table 6.2

Fonction 6.3. Generation_Cle_modele(Cle)

```
Structure :moy[n,2], Cle<bit[n],cout>
 Pour j ← 1 à n faire
    moy[j,1] ← Somme(k[i].bit[j]; i=1..maxAny et bit[j]='1')
    moy[j,2] ← Somme(k[i].bit[j]; i=1..maxAny et bit[j]='0')
FinPour
Pour j ← 1 à n faire
    Cle.bit[j] ← Max(moy[j,1],moy[j,2]
FinPour
```

6.4 Implémentation

L'évaluation des clés de déchiffrement se fait d'une manière relative selon un classement des autres clés, l'usage des metaheuristiques classiques utilisant une solution unique ne peut donc être envisagé. L'implémentation est cependant limitée aux metaheuristiques à population, à savoir l'ACO, l'OEP et les algorithmes génétiques.

6.4.1 Algorithme de Colonie de Fourmis

L'algorithme gère une colonie de *maxAnts* fourmis ; chacune d'elles représente une clé candidate *k* de taille *n* bits, ce qui correspond à un graphe de n sommets. La structure *Cle_struc* associe à chaque clé *k*, son *cout*, ce qui correspond à la valeur de la fonction objectif. A chaque itération *j*, la fourmi *i* examine le bit *j* de la clé *i* ; elle calcule le coût de cette clé et, est comparée avec le coût après inversion du bit *j*. L'inversion est maintenue si elle produit un coût meilleur. La mise à jour du phéromone se traduit par le calcul de moyenne des bits '0' et '1' comme présenté dans l'exemple de la table 6.2. Les valeurs de bits ayant une moyenne supérieure à 0.5 serviront à construire la clé modèle qui sera injectée dans la population. Le processus continue tant que le

critère d'arrêt n'est pas satisfait. Le résultat consiste en la clé ayant un cout minimal avec le texte déchiffré correspondant.

Algorithme 6.4. Cryptanalyse_Symetriques_ACO

Entrée: *Cipher[n],moy[n,2]*
Cle_struc<bit[n],cout> : Structure de clé candidate
k[maxAnt], k, Cle_essai, Cle_modele* : *Cle_struc*
Sortie: *k**: clé optimale
$Plain_{k*}[n]$: bloc clair de taille n bits déchiffré par la clé k*
Pour *i* ← *1 à maxAnt* **faire**
  Pour *j* ← *1 à n* **faire**
    k[i].bit[j] ← aleatoire(0,1)
  FinPour
  k.Cout ← *Cout(k,$plain_k$)*
FinPour
TanQ <*critère_arrêt_non_satisfait*>
  Pour *i* ← *1 à maxAnt* **faire**
  Cle_essai ← *k[i]*
  Pour *j* ← *1 à n* **faire**
    Cle_essai.bit[j] ← Inv*(Cle_essai.bit[j])*
    Cle_essai.Cout ← *cout(Cle_essai,$plain_{Cle\_essai}$)*
    Si *Cle_essai.Cout* < *k[i].Cout*
      k[i].bit[j] ← *Cle_essai.bit[j]*
    FinSi
  FinPour
  Si *k[i].Cout* < *k*.Cout*
    k.Cout* ← *k[i].Cout*
  FinSi
  FinPour
  Ordonner*(k,cout)*
  CalculMoy**(***moy***)**
  Generation_Cle_modele*(Cle_modele)*
  Remplacer*(k[maxAnt])* avec *Cle_modele*
FtanQ

La structure *Cle_modele* est une clé fabriquée à chaque itération telle la clé *k'*$_5$ illustrée dans l'exemple de la table 6.2. Elle remplacera la clé la moins bonne du processus, ce qui correspond à l'élimination de la fourmi bloquée en minima local. Elle favorise donc la progression vers l'optimum.

6.4.2 Algorithmes Génétiques

Les algorithmes génétiques modifient les valeurs des bits des clés candidates tel représenté par l'algorithme 6.5 où chaque individu de la population représente une clé candidate de *n* bits. La fonction *cross()* divise la clé en deux segments en les échangeant ; la mutation *inv()* n'est autre qu'une inversion d'un bit donné selon le taux proposé. Les clés ainsi transformées seront ensuite utilisées pour décrypter des blocs de chiffrement et calculer éventuellement leurs couts. A chaque itération, la fonction *Ordonner* trie la liste des clés en fonction de leurs coûts ; la clé en queue de liste sera remplacée par la *cle_modele* générée par la fonction 6.3. Le processus commence par une population initiale de clés générées aléatoirement de taille *TaillePop*; Il pred fin après *MaxGen* itérations. La première clé de la liste représente la solution optimale du problème.

Algorithme 6.5. Cryptanalyse_symétriques_AG

Entrée: *Cipher[n]*
Cle_struc<bit[n],Cout> : Structure de clé candidate
k[TaillePop] : Cle_struc
MaxGen, Nb_Seg, Taux_mut

Sortie: *k*, Plain$_{k*}$[n]*
Pour *i ← 1 à TaillePop* **faire**
k[i].Cout ← cout(k[i],plaink$_{[i]}$)
FinPour
i ← 1
TantQue *i < MaxGen*
cross(k,j) avec *j* < TaillePop
k[j] ← inv(k[j], Taux_mut)
Pour *j ← 1 à TaillePop* **faire**
k[j].Cout ← cout(k[j],plain$_{Cle[k]}$)
FinPour
i ← i+1
Ordonner*(k,cout)*
CalculMoy**(***moy***)**
Generation_Cle_modele*(Cle_modele)*
Remplacer*(k[TaillePop])* avec *Cle_modele*
FtantQue
k ← k[1]*

6.4.3 Algorithme d'Essaims particulaires

L'essaim de particules de taille *TailleSwarm* représente l'ensemble de clés de déchiffrement; la position x_i de chaque particule *i* de l'essaim représente le coût $Cout_{k[i]}$ de la clé associée *k[i]*. La structure *k** désigne le meilleur voisinage et représente la clé provisoirement optimale. A chaque itération *i*, le *i*-ème bit de chaque clé candidate prendra provisoirement la valeur du bit de la même position de *k**. Si son cout s'améliore avec un taux c_g par rapport au cout de *k**, il garde cette valeur et deviendra clé optimale, sinon, il reprendra sa valeur initiale.

A l'issue de chaque itération, la liste des clés sera triée selon le cout ; la clé la moins performante sera éliminée et remplacée par la *clé_modele* définie par la fonction 6.3

Algorithme 6.6. Cryptanalyse_symetrique_ OEP

Entrée: *Cipher[n]*
Cle_struc<bit[n],cout>
k[TailleSwarm],Cle_essai : *Cle_struc*
CoutMax
Sortie: *k*, $Plain_{k*}[n]$*
Générer x_i, v_i *(i=1.. TailleSwarm)* et c_g
k.Cout* ← *CoutMax*
Pour *i* ← *1 à TailleSwarm* **faire**
Pour *j* ← *1 à n* **faire**
k[i].bit[j] ← aleatoire*(Alpha)*
k[i].Cout ← *Cout(k[i],$plain_{k[i]}$)*
Si *(k[i].Cout - k*.Cout)* > c_g
*k** ← *k[i]*
FinSi
FinPour
FinPour
Répéter
Pour *j* ← *1 à n* **faire**
Pour *i* ← *1 à TailleSwarm* **faire**
Cle_essai ← *k[i]*
Cle_essai.bit[j] ← *k*.bit[j]*
Cle_essai.Cout ← *Cout(Cle_essai,$plain_{k[i]}$)*
Si *Cle_essai.Cout < k*.Cout*
k[i] ← *Cle_essai*
*k** ← *Cle_essai*
FinSi
FinPour
FinPour
Ordonner*(k,cout)*
CalculMoy**(***moy***)**
Generation_Cle_modele*(Cle_modele)*
Remplacer*(k[TaillePop])* avec *Cle_modele*
Jusqu'à <*critère d'arrêt*>

6.5 Conclusion

Ce présent chapitre présente un aperçu quant à la littérature liée aux attaques aux chiffrements symétriques et utilisant des techniques heuristiques. Les études ainsi faites montrent clairement que le majeur problème réside au niveau de la fonction objectif qui justifie la qualité des résultats d'attaque, étant donné que les diverses fonctions utilisées en cryptanalyse classiques ne peuvent être utilisées en cryptanalyse des chiffrements symétriques à cause de la non-linéarité dont dotées les Sboxes. Egalement, les recherches dans ce contexte se concordent sur le fait que l'évaluation de la qualité d'une attaque dépend de la qualité de chaque clé utilisée et autant que le nombre de clés utilisées est grand, le résultat serait mieux. Ce constat exclut éventuellement de la course toutes les metaheuristiques à solution unique.

L'étude et comparaison des différents travaux nous a permis d'élaborer une stratégie d'attaque permettant à chaque itération, d'évaluer la qualité du résultat de chaque clé candidate et de construire une nouvelle clé à base de la performance des autres clés; elle sera injectée dans la population en remplacement de la clé la moins performante. Cette contribution permet de garder en permanence une population de clés de qualité et de converger la recherche rapidement.

Les metaheuristiques utilisées comme outils d'attaque ont été modélisées à base de structures fixes permettant ainsi de gérer convenablement l'espace mémoire utilisé et d'éviter les problèmes liés aux structures dynamiques, ce qui nous permet d'augmenter la population ou le nombre d'itérations avec confiance.

7. Expérimentation & Résultats

7.1 Introduction

Ce chapitre est consacré à l'évaluation. Son but est de procéder aux attaques d'un matériel varié de chiffrement en utilisant les diverses metaheuristiques traitées. Le procédé des attaques comporte un ensemble d'outils : algorithmes, données, paramètres et stratégies de tests et de comparaison. A l'issu des résultats obtenus, une synthèse des divers metaheuristiques a été élaborée afin de mesurer leur efficacité et performance.

7.2 Données du problème

Les données de l'expérimentation englobent une base de 50 textes extraits de *BrownCorpus*[1] et chiffrés par les divers algorithmes présentés, à savoir : la substitution monoalphabétique (clés de 26 caractères), la substitution par transposition (clés de 5 à 9 caractères), la substitution polyalphabétique (cas du chiffrement Vigénere avec des clés de 5 à 11 caractères), un chiffrement par blocs (clé de 10 caractères); le chiffrement DES et 4DES (clé de 56 bits) et enfin le chiffrement SDES (clé de 10 bits).

Chaque chiffrement a été utilisé pour la génération de 50 textes de taille variant de 100 à 1500 caractères), soit au total, une base test de 300 textes chiffrés.

1. http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml

7.3 Environnement de tests

Dans cette section, un certain nombre de résultats expérimentaux relatifs à la performance de chacun des algorithmes décrits a été présenté. Chaque résultat provient de plusieurs tests avec de valeurs différentes de paramètres : textes chiffrés et tables statistiques, clés de déchiffrement de départ, limites de phéromone, période d'évaporation et nombre de fourmis (cas de l'algorithme VMMAS), taille de chromosomes ; segments de croisement et taux de mutation (cas des algorithmes génétiques), coefficients de voisinage, nombre de particules (cas de l'algorithme d'essaims particulaires). Les résultats ont été présentés par des histogrammes ; Chaque barre représente la moyenne de trois tests indépendants sur le même chiffrement. Ceci est dû à la nature aléatoire des algorithmes utilisés où il est nécessaire de considérer une moyenne de résultats différents afin de donner une représentation raisonnable de la performance de chaque algorithme.

7.4 Corpus & tables statistiques

Comme présenté au §3.3.3, une table de fréquence illustre le pourcentage de la fréquence de chaque caractère d'un texte donné. Dans le contexte de la cryptanalyse classique, on se limite aux 26 caractères de l'alphabet seulement sans prendre en considérations les chiffres, la ponctuation ou l'espace, car ces derniers ne peuvent renseigner sur le sens du texte déchiffré. Vu la diversité des tables disponibles en littérature, les travaux en cryptanalyse ne mentionnent jamais la nature ou la source des tables utilisés, ce qui crée la divergence des résultats présentés et empêche la possibilité d'une comparaison correcte ou reproduction de ces résultats. Afin de limiter cette convergence, les tables relatives aux bigrams et trigrams retenus sont présentées par les tables (tab.7.1.) et (tab.7.2).

Bigram	%	Bigram	%	Bigram	%
OF	0,85	TI	1,07	AN	1,32
NG	0,85	OR	1,07	HE	1,37
EA	0,88	ED	1,08	ON	1,47
TE	0,90	NT	1,08	HA	1,54
IT	0,91	CO	1,09	RE	1,56
OU	0,94	TO	1,10	ER	1,58
AT	0,95	ND	1,11	IN	1,81
IS	0,97	ES	1,18	TH	2,74
AS	0,98	EN	1,31		

Tab. 7.1. Fréquence d'apparition bi-grams

Les chiffres indiqués dans ces tables en guise de pourcentage d'apparition de bigrams et trigrams dans un texte de 100 caractères reflètent un ordre plutôt qu'un pourcentage c'est-à-dire, que le bigram OF par exemple, apparaît moins souvent que le bigram EA, TE et les autres tandis que le bigram TH est le moins visible parmi tous les autres. De même, La somme des valeurs mentionnées dans le tableau ne dépassent pas les 20%, ce qui veut dire que le nombre des n–grams (n>1) est indéfini et que ce type de statistiques ne doit pas être utilisé comme paramètre essentiel dans les

expérimentations. Les statistiques liées aux trigrams et plus généralement aux n-grams (n>3) sont moins utilisées car, plus qu'un mot contient de lettres plus qu'il est moins fréquent au sein d'un texte et qu'il serait plus intéressant d'utiliser un dictionnaire au lieu d'une table statistique (Hart G. , 1994) (Morelli, 2004).

Trigram	%	Trigram	%	Trigram	%
EAR	0,22	TER	0,25	RHI	0,3
ERS	0,22	REA	0,25	FOR	0,3
OFT	0,23	VER	0,26	NTH	0,34
NOT	0,23	DTH	0,26	ENT	0,34
OTH	0,24	TTH	0,27	ERE	0,34
ATI	0,24	YOU	0,27	HAT	0,39
OUR	0,25	ION	0,29	ETH	0,39
EST	0,25	TIO	0,29	HER	0,41
ALL	0,25	INT	0,29		

Tab. 7.2. Fréquence d'apparition tri-grams

Les statistiques liées aux unigrams sont considérées comme un élément incontournable pour l'analyse des propriétés d'un texte, et comme il existe une variété de tables statistiques disponibles, la table de fréquence choisie (Tab.7.3) a été construite à base d'un certain nombre de tables présentes sur le net.

Car	Fréq.	Car	Fréq	Car	Fréq
A	8,27	J	0,66	S	6,48
B	1,45	K	2,19	T	9,05
C	2,81	L	3,89	U	2,86
D	4,28	M	2,63	V	1,04
E	13,02	N	6,79	W	2,13
F	2,10	O	7,73	X	0,20
G	1,96	P	2,66	Y	2,01
H	6,02	Q	0,27	Z	0,11
I	6,76	R	6,21		6,67

Tab. 7.3. Fréquence d'apparition unigrams

7.4.1 Fonction objectif

En cryptanalyse classique, Le coût d'un texte correspond à la somme des valeurs de ses caractères. La valeur de chaque caractère est extraite de la table de fréquence utilisée. La valeur de la fonction objectif est la différence absolue entre le coût du texte déchiffré et le standard du langage représenté par la table de fréquence utilisée. Autant que cette différence est proche de

zéro, autant que le texte en question est plus proche du standard du langage. L'exemple suivant illustre la manière d'une telle évaluation.

Soit le texte chiffré représenté par l'expression : « XHXI HST UN HXHMPLH ». Ce texte comporte 16 caractères (espaces non compris). L'occurence des caractères est donnée par la colonne 3 du tableau (tab.7.4) dont la deuxième colonne comporte les fréquences standards présentées au tableau (tab.7.3). La colonne 4 détermine la valeur de chaque caractère du texte, soit la *Fréq*Occ*. La colonne 5 présente le pourcentage de la valeur calculée, soit *Valeur*16/100*. (Cette conversion est nécessaire car la table standard de fréquences a été élaborée à base de d'un pourcentage). Enfin, la dernière colonne illustre la relation *|Freq-%|* qui désigne la différence entre le standard du langage et celui du texte déchiffré. Si cette différence est nulle par exemple, ceci veut dire que le texte a les mêmes caractéristiques que le langage (type de caractère et fréquence), il est donc, le texte cherché. La valeur de la fonction du coût est donnée par la somme des différences des 26 caractères. Le principe du déchiffrement tend à minimiser cette fonction le plus possible vers zéro.

Car	Fréq	Occ	Valeur	%	Différence
A	8,27	0	0	0	8,2652
B	1,45	0	0	0	1,4508
C	2,81	0	0	0	2,8058
D	4,28	0	0	0	4,2786
E	13,02	0	0	0	13,018
F	2,10	0	0	0	2,1
G	1,96	0	0	0	1,9636
H	6,02	5	30,119	4,819	1,2048
I	6,76	1	6,7582	1,0813	5,6769
J	0,66	0	0	0	0,655
K	2,19	0	0	0	2,1882
L	3,89	1	3,8946	0,6231	3,2715
M	2,63	1	2,6264	0,4202	2,2062
N	6,79	1	6,7894	1,0863	5,7031
O	7,73	0	0	0	7,728
P	2,66	1	2,656	0,425	2,231
Q	0,27	0	0	0	0,2664
R	6,21	0	0	0	6,214
S	6,48	1	6,4772	1,0364	5,4408
T	9,05	1	9,051	1,4482	7,6028
U	2,86	1	2,8642	0,4583	2,4059
V	1,04	0	0	0	1,0428
W	2,13	0	0	0	2,1288
X	0,20	2	0,3976	0,0636	0,1352
Y	2,01	0	0	0	2,0102
Z	0,11	0	0	0	0,108
			Valeur de la fonction du coût		92,102

Tab. 7.4. Exemple de calcul de coût de texte

La valeur de la fonction du coût ainsi déterminée est assez élevée car la colonne 4 du précédent tableau comporte un nombre important de zéros. Pour des textes de taille importante et contenant divers caractères différents, le coût serait inférieur à 1.

Concernant les chiffrements symétriques, la fonction objectif est simple et ne nécessite aucun calcul car elle est basée sur un texte clair connu, c'est à dire, une clé de chiffrement connue. Son principe consiste à comparer le contenu de la clé générée (en nombre de bits et position) et la clé de référence tel mentionné au §6.3.1.

7.4.2 Processus d'exécution & critères d'arrêt

Le corps de chaque algorithme consiste en un processus d'exécution sur un nombre fini d'itérations. Au sein de chacune d'elles, les étapes suivantes seront accomplies dans l'ordre : *génération* d'une clé (selon la metaheuristique utilisée), *déchiffrement* du texte et *évaluation* de la fonction objectif. Le processus peut être arrêté si un critère d'arrêt est vérifié (limite de temps ou de nombre d'itérations). Le test peut être écourté également si la fonction objectif demeure stable durant 10 itérations successives ou régresse durant 5 itérations successives.

7.5 Paramètres des algorithmes à population

7.5.1 Algorithmes de Colonies de Fourmis

Comme il a été décrit en §4.8.1, le *AntSystem* débute par une solution au coût maximal qui doit être minimisé au cours de l'exploration au sein d'un graphe unidirectionnel. A chaque itération, le choix du sens de déplacement est un facteur important dont dépend l'optimalité de la solution. Ce choix est géré par divers paramètres relatifs à l'état du parcours : α et β : variables de choix de direction (eq.4.2); $\Delta\tau$: quantité de phéromone déposée par la fourmi (eq.4.3); ϱ: quantité de phéromone évaporée auxquels on ajoute d'autres paramètres nécessaires à l'implémentation : nombre de fourmis; limites supérieure et inférieure de phéromone sur les arcs; période d'évaporation; etc.

Les paramètres ainsi énumérés doivent être définis d'une manière expérimentale en l'absence de modèle mathématique permettant de les déterminer d'une manière optimale. En littérature, peu de travaux font référence à la manière de définition de ces paramètres ; (Uddin & Youssef, 2006b) dans ses travaux de cryptanalyse des chiffrements par substitution, énonce que ces paramètres constituent un handicap pour les heuristiques car leur ajustement est un problème délicat et nécessite un modèle d'optimisation. D'autres travaux dans ce contexte (Khan, Ali, & Durrani, 2013) (Russel, Clark, & Stepny, 2003) (Khan, Shahzad, & Khan, 2010) ont utilisé les algorithmes de colonies de fourmis pour la cryptanalyse de divers chiffrements sans mention des valeurs de paramètres ou de la manière quant à leurs choix.

De ce fait et, afin de fixer des valeurs optimales de certains paramètres, des tests préliminaires ont été réalisés en utilisant une base 100 passages de *BrownCorpus* de taille variable (100 à 500 caractères) chiffrés par la technique de substitution monoalphabétique. Le processus d'exécution a été fixé à 200 itérations. Les coefficients α et β ont été fixés respectivement 0.7 et 0.8. La figure (7.1) montre l'impact de la variation du nombre de fourmis sur la performance de l'ACS (utilisé comme algorithme de référence) en termes du coût de la solution représentée par la clé de

déchiffrement k définie par l'équation (5.2). On remarque que la fonction du coût décroit si le nombre de fourmis augmente. Elle se stabilise sensiblement si ce nombre dépasse 15 fourmis et ce, pour des textes de différentes tailles.

Fig. 7.1. Variation du coût de solution via le nombre de fourmis

Pour la suite des tests, le nombre de fourmis sera fixé à 20. Un nombre supérieur de fourmis produira des résultats plus performants mais sera gourmand en temps de calcul.

La figure 7.2 fournit des informations sur le pourcentage du nombre de caractères corrects obtenus au sein des clés de déchiffrement par les algorithmes de colonies de fourmis, à savoir : ACS, VAS, EAS et VMMAS et ce, en utilisant des textes chiffrés par les techniques de chiffrements utilisées.

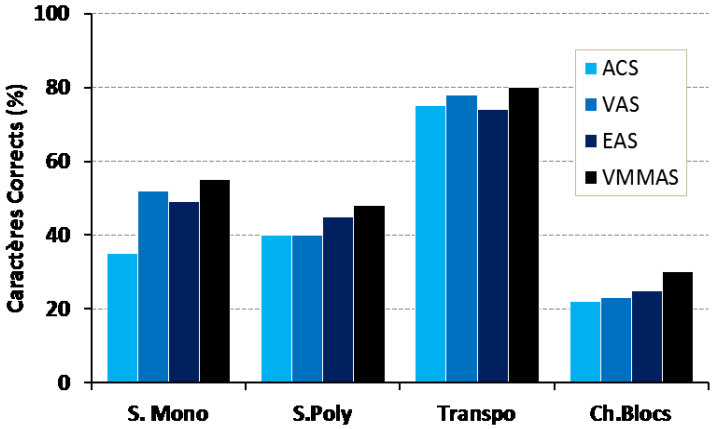

Fig. 7.2. . Performance des algorithmes de Colonie de Fourmis

Il est évident de constater que les algorithmes VAS et VMMAS performent mieux pour la plupart des chiffrements considérés. D'un autre côté, l'algorithme VMMAS surpasse les autres algorithmes pour tous les chiffrements, notamment la transposition. Il apparaît donc que les deux algorithmes EAS et VMMAS produisent des performances équivalentes et dépassant celles des autres algorithmes testés. Du coté chiffrements, il s'avère que la transposition est la plus facile à attaquer vu la taille réduite des clés utilisées pour ce type de chiffrement.

Dans ce qui suit, on retient l'algorithme VMMAS comme référentiel aux algorithmes ACO pour d'éventuels tests de comparaison avec d'autres heuristiques.

7.5.2 Algorithmes génétiques

Similaire aux ACO, les algorithmes génétiques sont dotés également d'un nombre important de paramètres qui doivent être fixés avec soin afin d'aboutir à une solution optimale. Comme mentionné dans l'algorithme 5.10 et, en plus des opérateurs génétiques, les AG utilisent certains paramètres relatives à l'implémentation, à savoir : la taille de la population, le nombre de générations, les diverses méthodes de sélection et de remplacement, le taux de la mutation, le nombre de points de croisement. Ces paramètres doivent être définis au préalable avant d'aborder les expérimentations de performance.

En l'absence de modèles permettant de fixer des valeurs optimales aux paramètres indiqués, en littérature, divers propositions ont été énumérées; (Heydari, Shabgani, & Heydari, 2013) propose deux points aléatoires de croisement (Sharma, Pathak, & Sharma, 2012a), proposent certaines valeurs (population : 100, sélection avec tournoi, un taux de 0.02 pour la mutation et un traitement de 50 générations), de même, (Morelli, 2004) (Garg, 2005) énoncent certaines idées quant au croisement et mutation, (Toemeh & Arumugam, 2007) (Toemeh & Arumugam, 2008) (Verma, Dave, & Joshi, 2007) (Nalini & Rao, 2007) présentent des résultats de cryptanalyse avec des metaheuristiques sans mention des valeurs de paramètres.

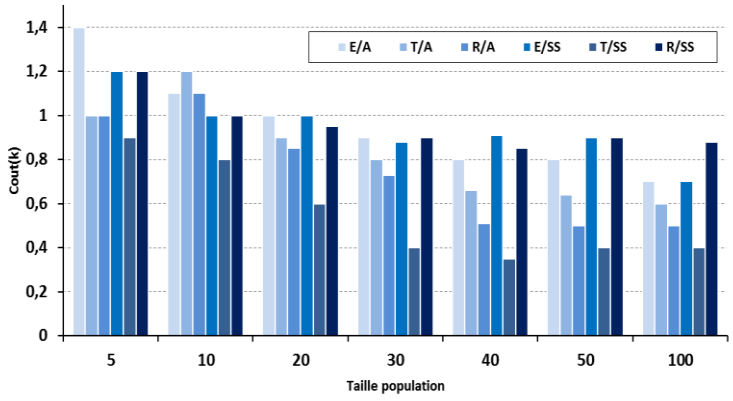

Fig. 7.3. . Variation du coût de solution via la taille de la population

En utilisant une population d'individus variable et deux points de croisements, des tests similaires à ceux présentés au §7.5.1 ont été opérés où la moyenne des résultats obtenus est présentée par le graphique de la figure 7.3 qui montre la variation du coût de clés de déchiffrement donné par l'AG. Diverses alternatives de sélection et de remplacement ont été testées, à savoir les méthodes : élitiste (E), par tournoi (T) et par roulette (R) pour la sélection et les méthodes : stationnaire (SS) et aléatoire (A) pour le remplacement. Le taux de mutation a été fixé à 0,04.

Il est apparait que la méthode élitiste produit une performance supérieure à la moyenne avec usage d'une population de plus de 20 individus. La deuxième meilleure performance est obtenue avec une sélection en tournoi et un remplacement stationnaire.

7.5.3 Algorithme d'optimisation par essaims particulaires

Au sein des algorithmes d'essaims particulaires, les coefficients d'inertie et la nature du voisinage constituent des facteurs essentiels dont dépend la convergence de la solution ; ils permettent en outre de fournir un équilibre entre l'exploration et l'exploitation de l'espace de recherche. La manière de choix des valeurs de ces paramètres a été objet de diverses recherches; (Eberhart & Shi, 2001) ont proposé un concept d'inertie aléatoire et ont constaté une amélioration de la convergence dès les premières itérations ; le meilleur résultat a été obtenu avec des valeurs comprises entre 0.4 et 0.9. (Ganapathi & Purusothaman, 2011) ont proposé les valeurs 0.9 et 2.0 pour les coefficients d'inertie pour la cryptanalyse de chiffrement polyalphabétique, les résultats étaient de l'ordre de 85% pour des textes de taille variable entre 100 et 200 caractères (Sharma, Pathak, & Sharma, 2012b); ont proposé, pour la cryptanalyse du chiffrement SDES, une version OEP binaire en utilisant des valeurs de coefficient d'inertie inférieures à 1 sur des chiffrements de 1000 caractères et ce, en utilisant une population de 100 particules. Des valeurs similaires ont été utilisées par (Vimalathithan & valarmathi, 2011a) et (Vimalathithan & Valarmathi, 2012) respectivement pour la cryptanalyse du DES et l'S-AES avec des textes dépassant les 1000 caractères. Les travaux menés par (Wafaa, Ghali, Hassanien, & Abraham, 2011) relatives à la cryptanalyse du DES avec l'algorithme OEP ont utilisé la valeur 2 aux coefficients d'inertie et la valeur 4 à la vitesse initiale pour une population de 500 particules.

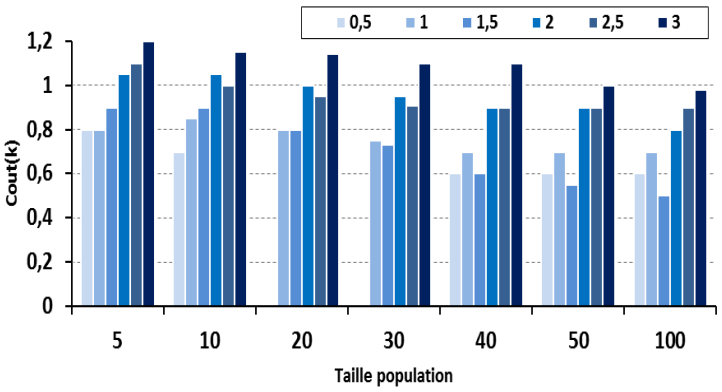

Fig. 7.4. . Variation du coût de solution via la taille de la population

La synthèse des divers travaux cités montre que des valeurs des paramètres proposées sont variées et distinctes car elles ont été fixées dans la plupart des cas à base expérimentale ou intuitive.

Ce fait pose des problèmes d'efficacité en pratique vu la spécificité des résultats quant à l'environnement de test et la nature des données utilisées.

Afin de remédier à cet effet, des tests similaires ont été opérés dans le but de fixer les valeurs optimales de certains paramètres en se basant sur la nature du problème considéré. Le graphique représenté par la figure 7.4 illustre la variation de la moyenne du coût de clés de déchiffrement en fonction de la taille de la population d'essaims et ce, pour diverses valeurs de vitesse de déplacement de particules conformément aux équations 4.5 et 4.6.

On remarque que la performance des clés de déchiffrement (représentée par la position des individus dans l'espace) est inversement proportionnelle à la taille de la population. Ceci est évident, étant donné qu'un nombre élevé de clés permet une exploration plus vaste de l'espace, donc, plus de solutions variées pouvant être optimales en partie et ce, contrairement à la vitesse de déplacement des particules qui apparaît proportionnelle au coût de la solution : autant que la vitesse est élevée, la solution correspondante est moins bonne ; ceci s'explique par le fait que les meilleurs solutions sont toujours proches de la solution optimale, c'est-à-dire que les meilleures clés de déchiffrement comportent plus de caractères similaires aux mêmes positions.

Dans les expérimentations suivantes, on se contente d'une valeur de 0.5 pour la vitesse et, afin de limiter le temps d'exécution, on fixe le nombre de la population d'essaims à 20. Un nombre supérieur à cette valeur permet d'améliorer la solution mais consommera plus de ressources.

7.6 Performance des attaques aux chiffrements classiques

La performance d'un algorithme en matière de cryptanalyse peut être mesurée à base d'un ou plusieurs des critères suivants :

- produire une valeur de la fonction du cout la plus proche possible de 0,
- produire un texte clair dont la géographie de ses caractères (type et position) est la plus proche possible de la géographie des caractères du texte clair source du chiffrement considéré,
- générer une clé de déchiffrement dont la géographie de caractères (type et position) est la plus proche possible de la clé de déchiffrement réelle.

Le premier critère est généralement approximatif comme mentionné au §5.2.5. Une valeur proche de 0 n'implique pas toujours un résultat satisfaisant vu la diversité des tables statistiques utilisées. Les deux autres paramètres sont équivalents, néanmoins, l'usage du troisième paramètre parait plus bénéfique car il permet de synchroniser la permutation des caractères au sein des chiffrements notamment la substitution monoalphabétique et transposition.

Egalement, les résultats ont été évalués en pourcentage de caractères corrects et non en nombre; ceci est dû à ce que la taille des clés diffère d'un chiffrement à un autre : elle est de 26 pour la substitution monoalphabétique, de 5 à 9 pour les autres chiffrements.

7.6.1 Algorithme de descente

Les algorithmes de recherche locale permettent de donner des résultats exacts; Ils ne sont donc pas adaptés à résoudre des problèmes combinatoires de taille importante vu leur consommation successive de ressources. En littérature, en rencontre peu de travaux utilisant ces techniques dans le domaine de la cryptanalyse. (Forsyth & Savafi, 1993) étaient les premiers à utiliser le *Recuit Simulé* pour la cryptanalyse des chiffrements simples ; cette technique et d'autres ont été également reproduites plus tard par (Clark, 1998), (Dimovski & Gligoroski, 2003) , (Garg, 2005) et (Uddin & Youssef, 2006b) pour la cryptanalyse de certains chiffrements classiques dans un cadre comparatif avec d'autres techniques, notamment la *Recherche Tabou*, les algorithmes génétiques et les algorithmes de colonies de fourmis. La majorité des résultats obtenus étaient acceptables pour des chiffrements volumineux dépassant les 1000 caractères.

Dans ce contexte, les tests ont été opérés sur divers chiffrements tels mentionnés au §7.2. Le voisinage de chaque de caractère englobe l'ensemble des autres caractères de l'alphabet excepté le caractère lui-même. La figure 7.5 illustre la variation du pourcentage de caractères corrects selon chaque chiffrement et ce pour des tests de 200 itérations.

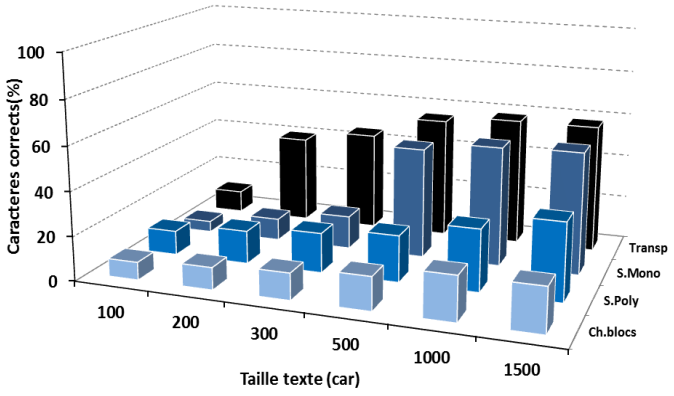

Fig. 7.5. . Performance de l'algorithme de descente (200 itérations)

Les résultats ainsi obtenus paraissent insatisfaisants, ceci est dû essentiellement au temps consommé lors de la recherche du meilleur voisinage auquel on y ajoute la possibilité de blocage au minima local. La figure 7.6 reprend les mêmes tests précédemment indiquées où le nombre d'itérations a été étendu à 1000.

Ainsi, le graphique de la figure 7.6 montre qu'une recherche locale poussée permet d'améliorer les résultats notamment la cryptanalyse des chiffrements par substitution monoalphabétique et par transposition. Ceci est évident, car chaque caractère de la clé de déchiffrement sera remplacé par un autre seulement si ce dernier permet une dégression du coût de la fonction objectif. Dans le cas contraire, le caractère en question sera maintenu et le processus de traitement passe au caractère suivant.

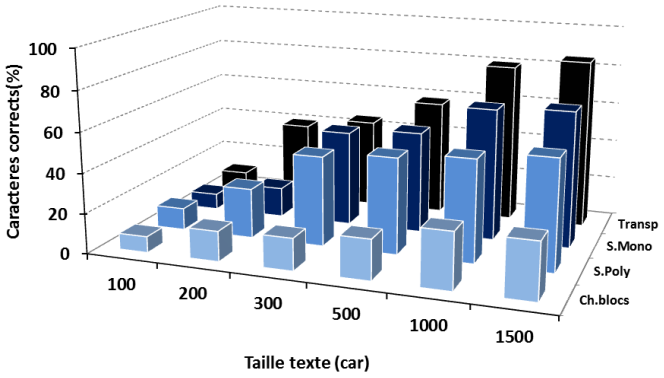

Fig. 7.6. . Performance de l'algorithme de descente (1000 itérations)

L'inconvénient de cette technique réside dans la manière de choix des caractères de remplacement, car les premiers caractères analysés de la clé seront substitués par les meilleurs remplaçants disponibles tandis que les derniers caractères seront maintenus dans la plupart des cas par manque d'alternatives meilleures. Cette stratégie peut en outre provoquer la stagnation de performance après un nombre réduit d'itérations.

7.6.2 Algorithme BFS

Cet algorithme est une alternative de recherche locale permettant d'examiner les sommets d'un graphe par niveau. En littérature, les techniques de recherche locale, notamment le GRASP (*Greedy Randomized Adaptive Search Procedure*), VNS (*Variable Neighbourdood Search*) ou GLS (*Guided Local search*) ont été souvent utilisées séparées ou combinées comme stratégies de recherche de bas niveau ; c'est-à-dire, utilisées pour générer des solutions acceptables de départ ou associées avec d'autre heuristiques tels les algorithmes génétiques pour l'exploration locale dans un espace de recherche réduit (Clark, 2001). En cryptanalyse classique, le BFS consiste à permuter, dans la mesure du possible, tous les caractères de la clé par d'autres afin d'améliorer le coût du texte déchiffré. L'opération est répétée en utilisant la nouvelle clé produite autant de fois que le critère d'arrêt le permet. Le voisinage de chaque caractère comporte l'ensemble des autres caractères de l'alphabet évalués et triés dans la liste F.

La figure 7.7 illustre la variation du pourcentage de caractères corrects révélés au sein des clés de déchiffrement générés par l'algorithme BFS en utilisant les données mentionnées au §7.2 et ce, pour un traitement de 500 itérations.

A première vue, l'exploration BFS présente une performance relativement satisfaisante ; ceci est dû à ce qu'un caractère donné de la clé de déchiffrement peut être permuté à un autre plus d'une fois si nécessaire ce qui évite que les « bons » caractères du voisinage soient monopolisés par les premiers caractères analysés de la clé. Ainsi, même les derniers caractères de la clé peuvent bénéficier d'une permutation intéressante. Ce fait a également l'avantage d'éviter les minima locaux : l'inconvénient majeur de l'exploration locale.

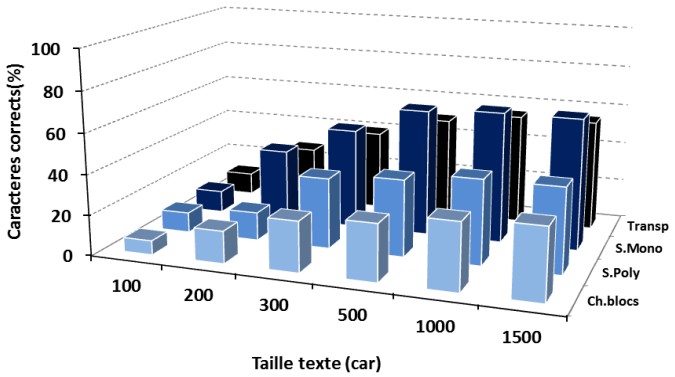

Fig. 7.7. Performance de l'algorithme BFS

Du coté chiffrements, la substitution monoalphabétique et la transposition apparaissent comme faibles vis-à-vis de l'algorithme BFS, tandis que le chiffrement par blocs demeure plus difficile à casser en regard de l'ensemble des chiffrements utilisés. Ceci est bien entendu vrai pour de textes de taille réduite (500 caractères max). La performance de l'algorithme demeure constante pour des textes de taille élevée.

7.6.3 Algorithme de Séparation et Evaluation

La stratégie de séparation et évaluation peut être appliquée facilement sous son format standard à la cryptanalyse de nombreux chiffrements. Toutefois, son efficacité et sa complexité en temps varient en fonction de la taille de la clé de déchiffrement utilisée. Si la clé est relativement longue, l'opération de retour devient embarrassante et fait perdre la qualité de «séparation » associée à cette stratégie (Biryukov, 2005). En littérature, l'algorithme a été peu utilisé pour la cryptanalyse de divers chiffrements complexes (Collard, Standaert, & Quisquater, 2008) et rarement utilisé pour les instances larges (Matsui, 1993).

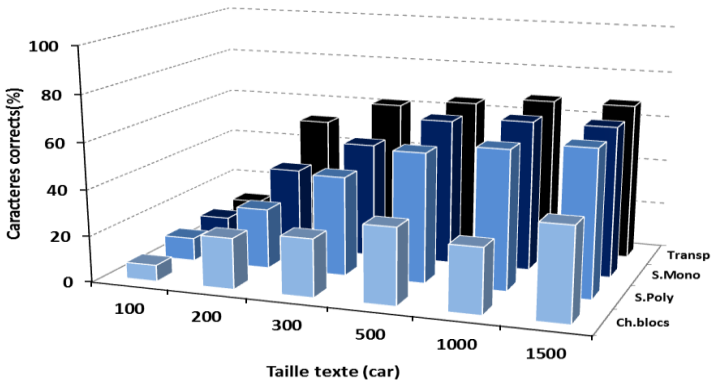

Fig. 7.8. . Performance de l'algorithme de Séparation et Evaluation

La figure 7.8 illustre la variation du pourcentage de caractères corrects au sein des clés de déchiffrement et ce au dépend de taille des chiffrements pour un traitement de 500 itérations. La performance de l'algorithme apparaît lors de la cryptanalyse des chiffrements dont la clé est de taille réduite ; ceci est évident car l'arbre exploré possède un nombre de niveau équivalent à la taille de la clé. De même, la performance varie sensiblement avec la taille des chiffrements ; ceci est également évident car les textes de taille importante reflètent des statistiques proches du standard littéraire utilisé.

7.6.4 Algorithme VMMAS

En utilisant une colonie de 20 fourmis, valeur retenu comme meilleure lors des tests préliminaires (§7.5.1), la performance de l'algorithme VMMAS est illustrée par la figure 7.9. Elle concerne le taux de reconnaissance de caractères corrects au sein des clés utilisées pour le déchiffrement de textes chiffrés par les techniques utilisées, à savoir : la substitution monoalphabétique et polyalphabétique, le chiffrement par transposition et par blocs et ce, pour un traitement de 500 itérations. On remarque qu'à l'exception de ce dernier chiffrement, l'algorithme en question présente un taux de reconnaissance supérieur à la moyenne notamment pour des textes de taille dépassant les 200 caractères. L'algorithme VMMAS, bien que meilleur que les autres algorithmes de colonies de fourmis, apparaît moins performant quant au chiffrement par blocs et, quelle que soit la taille du texte utilisé. Ceci est évident, car ce type de chiffrement est complexe et parait difficile à déchiffrer en général.

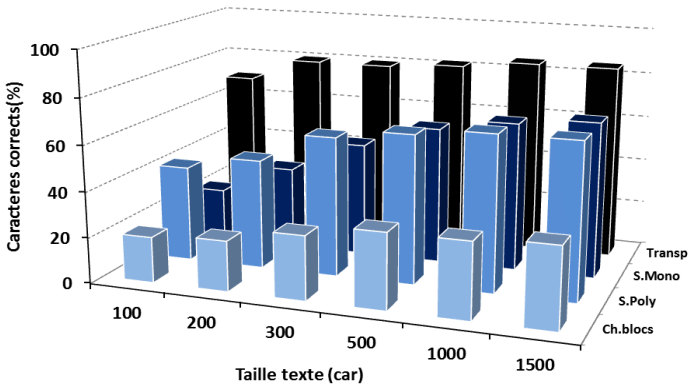

Fig. 7.9. . Performance de l'algorithme VMMAS pour l'attaque aux chiffrements dassiques

7.6.5 Algorithmes génétiques

En utilisant une population de 20 individus, valeur retenu comme meilleure lors des tests préliminaires (§7.5.2), deux points aléatoires de croisement, un taux de mutation égal à 4%, les stratégies de tournoi et stationnaire respectivement pour la sélection des individus et leur remplacement, des tests de 500 itérations ont été réalisées sur la base de textes mentionnée au

§7.2. La moyenne du pourcentage des caractères révélés au sein des clés de déchiffrement a été illustrée par le graphique représenté par la figure 7.10. Dans cette figure, on remarque que la meilleure performance a été obtenue en utilisant des textes de taille supérieure à 300 caractères en général et pour les chiffrements par transposition. Cependant, les algorithmes génétiques performent moins pour les autres types de chiffrement et spécialement pour le chiffrement par blocs.

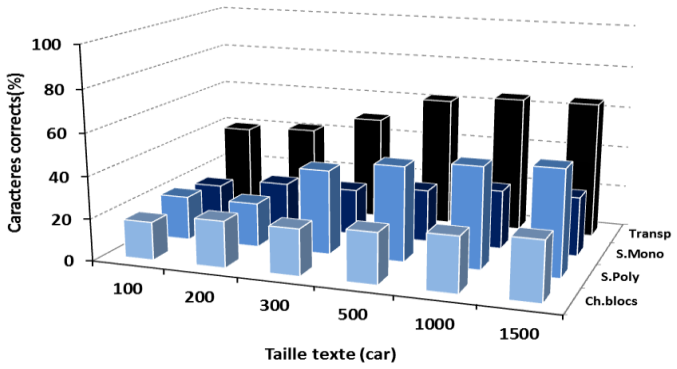

Fig. 7.10.. Performance des algorithmes génétiques pour l'attaque aux chiffrements classiques

L'analyse de ces résultats parait évidente : en général les textes de tailles importantes offrent des résultats meilleurs que les textes courts car ces derniers ne reflètent pas les statistiques standards des langages étant donné qu'ils ne comportent pas parfois l'ensemble des caractères de l'alphabet utilisé. En ce qui concerne la performance des AG, elle parait efficace lors de l'attaque aux chiffrements ayant des clés de courte taille (transposition et substitution polyalphabétique) étant donné que l'opérateur de croisement ne permet pas la stabilisation de chaines de caractères de grande taille (cas de la substitution monoalphabétique par exemple) et qui empêche par conséquence la construction de la bonne clé de déchiffrement.

7.6.6 Algorithme OEP

Pour un essaim de 20 particules distribuées aléatoirement sur l'espace de recherche et, ayant une vitesse initiale égale à 0.4, des tests (cf. §7.3) ont été réalisées sur la base de textes mentionnée au §7.2. La moyenne du pourcentage des caractères révélés au sein des clés de déchiffrement a été illustrée par le graphique représenté par la figure 7.11 et ce, pour un traitement de 500 itérations.

L'optimisation par essaims particulaires produit des résultats acceptables lors de la cryptanalyse des chiffrements de taille réduite. Ces résultats apparaissent performants pour des textes ayant plus de 300 caractères où la moyenne des caractères corrects au sein de la clé dépasse 60% pour la plupart des chiffrements utilisés. Elle est presque intégrale quant aux chiffrements par transposition et par substitution monoalphabétique. La technique OEP et, comme toute autre heuristique, performe moins pour les chiffrements par blocs ; ces derniers sont complexes et

nécessitent, pour leur cryptanalyse, plus de ressources notamment un traitement de milliers d'itérations.

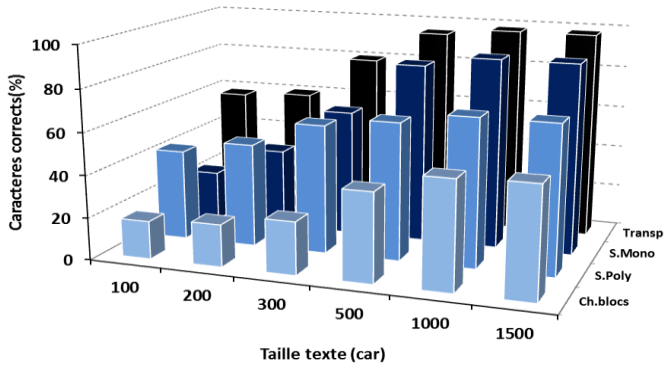

Fig. 7.11. Performance de l'algorithme OEP pour l'attaque aux chiffrements classiques

7.6.7 Comparaison & analyse

Afin de mesurer la performance des metaheuristiques évoquées, un nombre de tests comparatif a été élaboré regroupant les metaheuristiques ci-mentionnées et qui consiste à déchiffrer un ensemble de 50 textes de 500 caractères. La figure 7.12 illustre la moyenne du pourcentage du nombre de caractères révélés au sein des clés de déchiffrement et ce pour un traitement de 500 itérations.

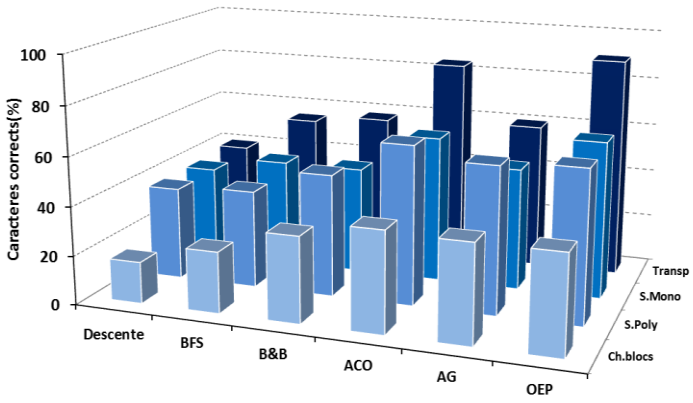

Fig. 7.12. Performance des metaheuristiques vs divers chiffrements classiques

Cette figure montre que les metaheuristiques à population performent mieux que les méthodes d'exploration locale; ce s'explique par le fait que l'usage d'une clé évolutive unique (cas de la

recherche locale) ne peut avoir une vue d'ensemble de l'espace de recherche et perd par conséquent, certaines opportunités liées aux zones non couvertes par l'exploration. Cet acte peut être évité par l'usage simultané d'un ensemble de clés différentes qui évoluent en même temps ; leur interaction permet d'orienter la recherche vers l'optimum sans toutefois se bloquer en minima locaux. On remarque également qu'une meilleure performance peut être obtenue quant à une cryptanalyse des chiffrements par transposition et ce, pour la plupart des techniques de recherche.

La figure 7.13 illustre la performance des diverses metaheuristiques utilisées en fonction du nombre d'itérations et ce, pour la cryptanalyse du chiffrement par transposition (chiffrement ayant la meilleure performance) en utilisant des textes de 500 caractères. Les résultats ainsi décrits montrent que la meilleure performance peut être obtenue à l'aide d'un nombre d'itérations supérieur ou égale à 300. Ceci s'explique par le fait qu'un nombre réduit d'itérations (100 par exemple) ne peut couvrir un nombre acceptable de possibilités de permutations liées aux 26 caractères utilisés. Une clé de transposition de 8 caractères par exemple nécessite une moyenne de 8x13 possibilités afin de couvrir l'ensemble de ses caractères par permutation d'une moitié de l'alphabet utilisée. Ce processus est nécessaire pour avoir une clé de coût raisonnable.

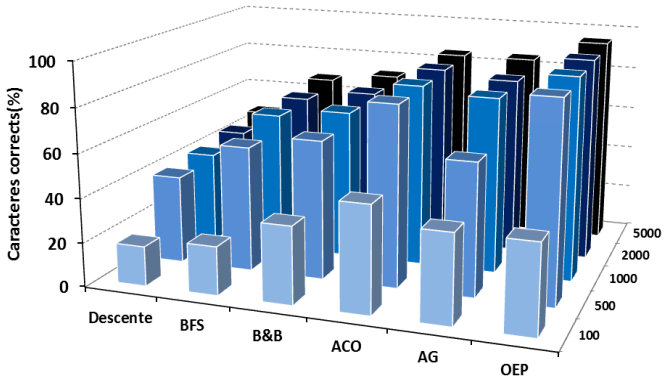

Fig. 7.13. .Performance des metaheuristiques vs le nombre d'itérations

En matière de temps d'exécution, la figure 7.14 montre l'évolution de la consommation en temps des diverses meaheuristiques utilisées et ce, pour un traitement de 500 itérations en utilisant des textes de 500 caractères. Il apparaît clair que les heuristiques à population sont gourmandes en temps de calcul ; ce temps est proportionnel à la taille de la population, néanmoins, les résultats sont plus performantes également. Le meilleur profit peut être remarqué au niveau des algorithmes génétiques car le traitement ne nécessite pas un calcul arithmétique important (telle la vitesse et la position du voisinage de l'OEP ou plutôt la gestion de phéremontation des ACO), ce qui réduit sensiblement le temps consommé par rapport aux autres metaheuristiques. Du côté des metaheuristiques de recherche locale, la meilleure performance est constatée au niveau de la technique de descente ; le surplus du temps consommé par ses concurrents BFS et B&B est dû à la gestion des listes de voisinage de chaque élément de la clé.

En matière de qualité, les meilleurs résultats sont obtenus par l'OEP où le compromis temps-coût parait le plus prometteur.

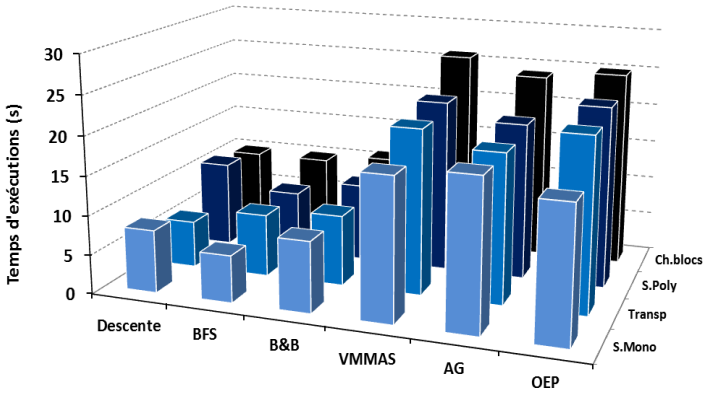

Fig. 7.14 . Performance des metaheuristiques en matière du temps de traitement

7.7 Performance des attaques aux chiffrements symétriques

La performance des attaques envers les chiffrements symétriques est liée directement à la fonction objectif définie par l'équation 6.6. Les résultats ont été évalués en pourcentage de bits corrects par rapport à la clé de chiffrement éventuellement connue. Cette clé est de 56 bits pour les chiffrements de 4DES et DES et de 10 bits pour le SDES.

7.7.1 Algorithme VMMAS

En utilisant des paramètres similaires à celles mentionnés au (§7.5.1), à savoir une colonie de 20 fourmis, des échantillons de textes de 800 à 4000 bits et un nombre de 2000 itérations, la performance de l'algorithme VMMAS est illustrée par la figure 7.15.

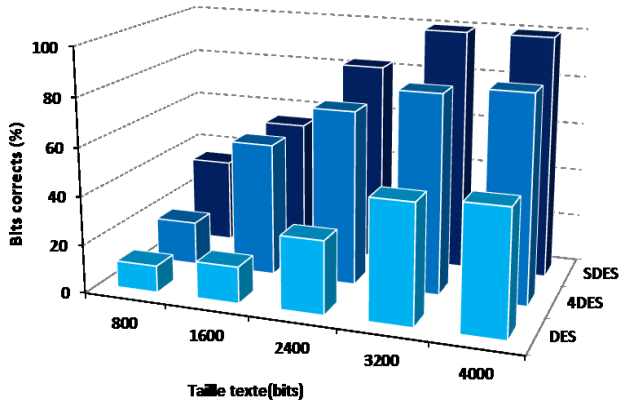

Fig. 7.15. Performance de l'algorithme VMMAS pour l'attaque aux chiffrements symétriques

On remarque qu'à l'exception du chiffrement DES, l'algorithme VMMAS produit un taux de bits correct acceptable pour des échantillons de 2400 bits (300 caractères). Pour des textes de 3200 bits (400 caractères), la quasi-totalité des bits de la clé sont révélées pour les chiffrements SDES et 4DES. Le taux de reconnaissance relatif au chiffrement DES est relativement faible : 51% ; ce qui signifie que 28 bits seulement sont révélés pour des textes de taille importante (500 caractères). Ceci est également évident car, peu de travaux en littérature ont abordé ce problème avec un algorithme ACO, les résultats obtenus étaient moins de 220 bits.

7.7.2 Algorithmes Génétiques

Avec des paramètres similaires à ceux utilisées pour l'attaque des chiffrements classiques, à savoir une population de 20 individus, deux points de croisement, un taux de mutation de 4%, les stratégies de tournoi et stationnaire respectivement pour la sélection des individus et leur remplacement, la moyenne du pourcentage de bits corrects obtenu à base d'une exécution de 2000 itération est présentée par le graphique représenté par la figure 7.16 . Dans cette figure, on remarque que la meilleure performance a été obtenue en utilisant des textes de taille supérieure à 300 caractères en général et pour le SDES. Cependant, les algorithmes génétiques performent moins pour les autres types de chiffrement et spécialement pour le chiffrement DES.

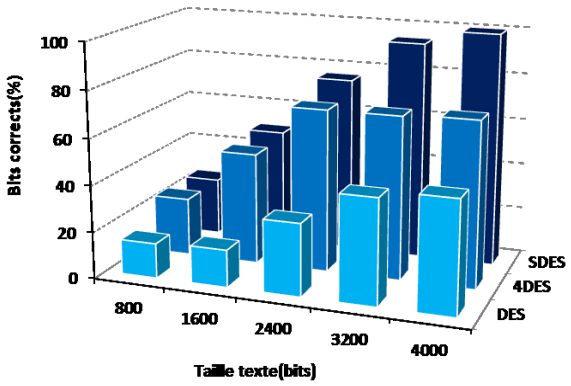

Fig. 7.16. Performance des algorithmes génétique pour l'attaque aux chiffrements symétriques

L'analyse de ces résultats montre que la performance des résultats croit sensiblement avec la taille des textes utilisés. Néanmoins et, en général, les AG sont moins performants que les autres metaheuristiques pour la résolution de ce genre de problèmes. La plupart des travaux similaires en littérature en utilisé ces algorithmes en hybridation avec l'OEP ou les heuristiques classiques afin d'améliorer leur performance.

Les meilleurs résultats ont été obtenus avec le SDES (100%) et le 4DES (70%), soient 10 bits et 32 bits respectivement. En littérature, les résultats similaires ont été respectivement de 10 et 26 bits obtenus avec un nombre largement inférieur d'itérations.

7.7.3 Algorithme OEP

En utilisant un référentiel de 20 particules ayant une vitesse initiale égale à 0.4, les résultats d'attaques à certains chiffrements symétriques sont présentés par le graphique 7.17. L'axe y désigne la moyenne du pourcentage des bits corrects des clés de déchiffrement obtenus à base d'un traitement de 2000 itérations.

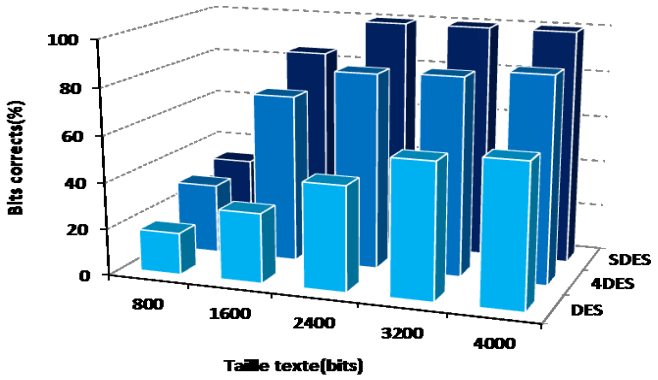

Fig. 7.17 . Performance de l'algorithme OEP pour l'attaque aux chiffrements symétriques

On remarque que l'optimisation par essaims particulaires produit des résultats acceptables à base de textes même de taille réduite et que la performance est son maximum au-delà de 2400 bits (300 caractères) , ce qui réduit nettement les ressources consommés en regard des autres metaheuristiques similaires. Les meilleures valeurs sont 100% et 85%, à savoir 10 bits et 10 et 45 bits respectivement pour les clés SDES et 4DES.

Pour le cas DES, les résultats demeurent acceptables 60% (une trentaine de bits) ; Les résultats similaires en littérature varient entre 31 et 39 bits.

7.7.4 Analyse des attaques aux chiffrements symétriques :

La figure 7.18 illustre la moyenne du pourcentage du nombre de bits révélés au sein des clés relatives aux chiffrements utilisés et ce pour des échantillons de textes de 2400 bits (300 caractères) à l'issue d'un traitement de 2000 itérations.

Selon la figure, l'OEP vient en tête en matière de performance et ce, pour l'ensemble des cas traités, suivi du VMMAS tandis que les AG paraissent moins performants. Ceci s'explique par la manière de progression au sein de l'espace d'exploration

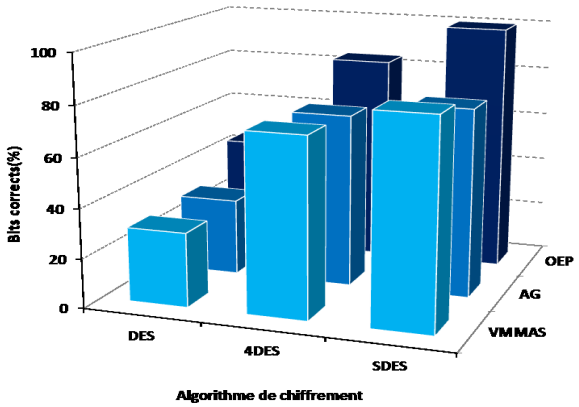

Fig. 7.18.. Performance des metaheuristiques vs divers chiffrements symétriques

Comme le montre les algorithmes 6.4 et 6.6 que, à chaque itération, le choix de la valeur du prochain bit de la clé est enregistrée dans une variable intermédiaire qui sera évaluée ; elle sera retenue si elle présente une performance ; Dans le cas contraire, la fourmi (ou la particule) rebrousse le chemin et choisit un autre point objectif et puisque ce retour arrière ne consomme pas beaucoup de temps (car il existe deux possibilités seulement), la clé construite est correcte et il n' y a pas de blocage en minima locaux. Contrairement aux AG (algorithme 6.5) qui utilisent le croisement qui parfois procède à une transformation erronée dans le corps de la clé, ce qui provoque un temps supplémentaire important afin de la mettre en ordre.

Avec les mêmes données et un traitement réalisé sur un CPU 3.0 ; l'évolution de la consommation en temps d'exécution est présentée par le graphique de la figure 7.19.

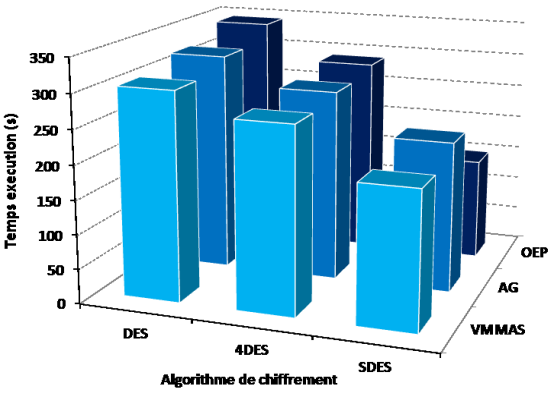

Fig. 7.19. Performance des metaheuristiques vs le temps de traitement pour l'attaque aux chiffrements symétriques

Il apparaît que la consommation en temps est quasi identique pour toutes les metaheuristiques utilisées ; De même, ce temps est également proportionnel au nombre de tours de déchiffrement.

En matière de qualité, les meilleurs résultats sont obtenus par l'OEP où le compromis temps-coût parait le plus prometteur.

7.8 Conclusion

Les expérimentations ainsi accomplies montrent que les attaques fondées sur les metaheuristiques conviennent parfaitement aux chiffrements classiques et ce, par la qualité des résultats obtenus, de l'ordre de plus de 50% pour les méthodes de recherche locale et de 90% pour les metaheuristiques à population. Aussi, il a été démontré que toute méthode peut être optimale dans son environnement spécifique et qu'aucune méthode ne peut être déclarée efficace au sens large. De même, l'usage des techniques de recherche locales peut être limité à un espace réduit avec une implémentation facile, un paramétrage restreint et éventuellement, un résultat limité. Les metaheuristiques à population sont généralement complexes : elles emploient un nombre important de paramètres où il est difficile de les optimiser qu'à titre expérimental. Néanmoins, seule une étude approfondie permet d'évaluer ces paramètres d'une manière optimale vis à vis du problème à résoudre. En contrepartie, une fois les paramètres évalués, ces metaheuristiques peuvent confronter des problèmes à larges instances sans difficultés.

Ainsi, l'étude réalisée a permis de contribuer en premier lieu à fixer certains paramètres essentiels relatifs aux metaheuristiques tels : la taille de la population, les meilleures stratégies de sélection et remplacement des individus ainsi que les valeurs de certains coefficients de pondération et ce, pour le problème de la cryptanalyse. Du côté attaques, l'étude a permis de sélectionner le meilleur format de la fonction de coût, le corpus linguistique et certaines statistiques liées aux chiffrements utilisés.

Les résultats obtenus ont permis de distinguer les heuristiques à base de leurs caractéristiques et de les classer selon leur performance quant à la cryptanalyse de divers chiffrements classiques.

8. Conclusion & Perspectives

La cryptanalyse, un aspect important de la cryptologie, consiste à découvrir les éléments faibles liées à la sécurité des données et permet aux auteurs des algorithmes de chiffrement de connaître les failles qui y sont liés afin de pouvoir les réparer ou plutôt de concevoir de nouveaux algorithmes plus robustes. La cryptanalyse est vue comme un problème combinatoire NP-difficile où la recherche de solutions exige des ressources proportionnelles à la taille du problème.

Dans ce mémoire, l'étude s'est penchée au volet des algorithmes de chiffrements classiques représentés par trois grands axes, à savoir, la substitution monoalphabétique, polyalphabétique et la transposition ainsi qu'un volet de chiffrement moderne : la substitution par blocs. Ces types de chiffrements, désormais jugés comme non fiable grâce à la puissance de calcul, constituent un noyau de base pour la construction des chiffrements symétriques modernes et dont les attaques doivent prendre en compte les techniques de substitution et de permutations élémentaires utilisées en cryptographie classique.

D'un autre côté, les heuristiques constituent une classe de méthodes approchées utilisées en recherche opérationnelle et destinée à une vaste catégorie de problèmes combinatoires. Leur succès est constaté par le fait qu'elles présentent un intérêt croissant dans la recherche actuelle et sont largement utilisées pour la résolution d'instances importantes. En revanche, leur réussite n'est guère justifiée car aucune méthode ne peut garantir qu'une heuristique sera plus efficace qu'une autre sur un problème particulier.

En pratique, certaines heuristiques offrent une mise en œuvre avantageuse, une implémentation rapide et surtout un paramétrage facile telles les méthodes de recherche locale ou de branchement et évaluation. Elles sont dédiées à la résolution de problèmes combinatoires de tailles réduites et offrent des solutions relatives ; d'autres sont plus complexes, nécessitant une implémentation

rigoureuse et un paramétrage assez délicat telles les heuristiques à population et peuvent manipuler les grandes instances. Elles offrent cependant des solutions satisfaisantes moyennant une consommation acceptable de ressources.

Le problème relevé dans ce mémoire consiste à mesurer l'efficacité de certaines heuristiques quant aux attaques de chiffrements classiques. Afin d'éclaircir cette question, un ensemble de tests descriptifs de comparaison a été réalisé dans le but de mettre en relief les facteurs supposés influencer sur la performance et les faire varier dans le but d'optimiser en conséquence les solutions. Les méthodes sélectionnées pour les tests intègrent une large gamme de metaheuristiques disponibles en littérature, à savoir les méthodes de recherche locale, de branchement-évaluation et les méthodes à population. La base de tests consiste en un grand nombre de textes chiffrés par les divers algorithmes proposés. Les résultats obtenus ont permis de fixer la plupart des paramètres importants liés aux heuristiques et classer ces dernières selon leurs caractéristiques et leur efficacité. Ils ont permis également d'évaluer la solidité des algorithmes de chiffrements utilisés.

Du côté littérature, on rencontre diverses alternatives d'attaques sur les chiffrements en utilisant différentes metaheuristiques. Les résultats étaient distincts et divergents par le fait des environnements de tests utilisés, propres à chaque attaque notamment en matière de textes et corpus utilisés où il parait difficile de mesurer l'efficacité d'une contribution par rapport à une autre.

Les travaux en perspectives consistent entre autres, à concevoir un système d'attaque paramétrable pouvant être enrichi par d'avantage de techniques d'attaques et plus de chiffrements avec un outil d'analyse des résultats et mise à jour adéquate des paramètres. L'idée peut être étendue à une conception hybride basée sur des heuristiques à population pour l'exploration dont la solution de départ est générée par certains algorithmes de recherche classiques. De même, un système d'attaque fiable doit pouvoir gérer un ensemble de langages littéraires au lieu d'une seule tel présenté dans ce mémoire et tel présenté dans les divers travaux similaires en littérature.

References

Abbas, E. (2012). Perfectionnement des algorithmes d'Optimisation par Essaim Particulaire. d'Optimisation par Essaim Particulaire. These. Paris.

Abde--Elmoniem, W. G., Ghali, N. I., & Hassanien, A. E. (2011). Known-Plaintext Attack of DES-16 Using Particle Swarm Optimization. Nature and Biologically Inspired Computing, (pp. 12-16).

Alba, E. (2005). Parallel Metaheuristics : A New Class of Algorithms. John Wiley & sons.

Anderson, R. J., Biham, E., & Knudsen, L. R. (2000). The case for serpent. AES Candidate Conference, (pp. 349-354).

Bafghi, A. G., & Sadeghiyan, B. (2004). Finding suitable differential characteristics for block ciphers with Ant colony technique. Ninth International Symposium on Computers and Communications (pp. 418-423). IEEE.

Banks, M. J. (2008). A Search-based Tool for the Automated Cryptanalysis of Classical Ciphers. University of York.

Bauer, F. L. (1997). Decrypted Secrets: Methods nd Maxims of cryptography. NY USA: Springer-Verilag.

Bellare, M., & Rogaway, P. (1995). Optimal Assymetric Encryption-How to ncrypt avec RSA. (Springer, Éd.) Lecture Notes in Computer Sciences, 950.

Ben-Aroya, L., & Biham, E. (1996). Differential cryptanalysis of Lucifer. Journal of Cryptology, 21-34.

Benett, C. H., F, B., Brassard, G., Salvail, L., & Smolin, L. (1992). Quantum Cryptography. Journal of Cryptology, 3-28.

Beni, G., & Wang, J. (1989). Swarm Intelligence in Cellular Robotic Systems. Advanced Workshop on Robots and Biological Systems. Tuscany, Italy.

Biham, E., & Shamir, A. (1991). Differential cryptanalysis of des-like cryptosystems. Journal of cryptology, 3-72.

Biham, E., & Shamir, A. (1993a). Differential cryptanalysis of the full 16-round DES. Advances in Cryptology, 487-496.

Biham, E., & Shamir, A. (1993b). Differential cryptanalysis of the data encryption standard. Springer-Verilag, 8-9.

BinAhmad, B., & BinMaarouf, M. A. (2006). Cryptanalysis using Biological Inspired Computing Approaches. Proceedings of the Postgraduate Annual Research Seminar 2006, (pp. 92-96).

Biryukov, A. (2005). Linear Cryptanalysis. Encyclopedia of Cryptography and Security , Kluwer Academic Publishers.

Biryukov, A., & Kush, A. (1998). From differential cryptanalysis to ciphertext-only attacks. Lecture notes in Computer Sciences, 72-88.

Biryukov, A., & Wagner, D. (1999). Slide Attacks. Fast Software Encryption (pp. 245-269). Rome: Springer-Verilag.

Blaum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 268-308.

Blum, C., & Li, X. (2008). Swarm Intelligence in Optimization. Natural Computing series, 43-85.

Blum, C., Roli, A., & Dorigo, M. (2001). HC-ACO: The Hyper-Cube Framework for Ant Colony Optimization. Metaheuristics International Conference, (pp. 399-403). Porto, Portugal.

Bogdanov, A., Knudsen, R., Leander, G., Paar, P. M., Robshaw, M. J., Seurin, Y., & Charlotte, H. V. (2007). An ultra-lightweight block cipher. Cryptographic Hardware and Embedded Systems (pp. 450-466). LNCS.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). Swarm Intelligence: From natural to artificial systems. Oxford University Press.

Carlisle, M. A. (1997). Constructing of Symmetric ciphers using the CAST design Procedure. Designs, Codes, and Cryptography, 283-316.

Carroll, J. M., & Robbins, L. (1987). The automated cryptanalysis of polyalphabetic. Cryptologia, 11(3), 193-205.

Carter, B., & Magoc, T. (2007). Classical Ciphers and Cryptanalysis. CiteSeerX.

Chatterjee, A., & Siarry, P. (2006). Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization". Computers and Operations Research, 33, pp. 859-871.

Clark, J. A. (1994). Modern Optimization Algorithms for Cryptanalysis. Second Australian and New Zealand Conference on Intelligent Information Systems (pp. 258-262). Brisbane, Qld: IEEE.

Clark, J. A. (1998). Optimisation Heuristics for Cryptology. PHD Thesis, Queensland University of Technology.

Clark, J. A. (2001). Metaheuristic Search as a Cryptological Tool. University of York.

Clark, J. A. (2003). Nature-inspired cryptography: Past, present and future. Congress of evolutionary computation. 3, pp. 1647-1654. IEEE.

Clark, J. A., & Dawson, E. (1997). A parallel genetic algorithm for cryptanalysis of the polyalphabetic substitution cipher. Cryptologia, 129-138.

Clark, J. A., & Jacob, J. L. (2001). Protocols are programs too: the meta-heuristic search for security protocols. Information and Software Technology, 43(14), 891-904.

Clark, J. A., Dawson, E., & bergen, H. (1996). Combinatorial optimization and the knapsack Cipher. Cryptologia, 85-93.

Clark, J. A., Jacob, J. L., & Stepney, S. (2004). The design of S-boxes by simulated annealing. Congress on Evolutionary Computation (pp. 1533-1537). IEEE.

Clark, J. A., Russell, M., & Stepney, S. (2003). Making the most of two heuristics: breaking transposition ciphers with ants. Congress on Evolutionary Computation. IEEE.

Clark, J. A., Russell, M., & Stepney, S. (2003). Using ants to attack a classical cipher. Lecture Notes in Computer Science, 2723, 146-147.

Clerc, M. (2003). TRIBES - Un exemple d'optimisation par essaims particulaires sans paramètres de controle. Optimisation par essaims particulaires, (p.). Paris.

Clerc, M., & Kennedy, J. (2002). The particle swarm : explosion, stability, and convergence in multi-dimensional complex space. IEEE Transactions on Evolutionary Computation, 6, pp. 58-73.

Collard, B., Standaert, F. X., & Quisquater, J. J. (2008). Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. Springer LNCS, 382-397.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. European Conference on Artificial Life, (pp. 134-142).

Coppersmith, D. (1994). The data encryption standard (DES) and its strength against attacks. IIBM Journal of Research and Development, 38(3), 243-250.

Cover, T. M., & Thomas, J. A. (1991). Information theory. Wiley series in communications

Dadhich, A., & Yadav, S. K. (2014). Swarm Intelligence and Evolutionary Computation based Cryptography and Cryptanalysis of 4-round DES algorithm. International Journal of Advanced Research in Computer Engineering & Technology, 1624-1633.

Dadhich, A., Gupta, A., & Yadav, S. (2014). Swarm Intelligence based linear cryptanalysis of four-round data Encryption Standard algorithm. Int. Conf. on Issues and Challenges in Intelligent Computing Techniques (pp. 378-383). Ghaziabad: IEEE.

Daemen, J., & Rijmen, V. (2000). Rijndael for AES. AES Candidate Conference, (pp. 343-348).

Delman, B. (2004). Genetic Algorithms in Cryptography.

Diffie, W., & Hellman, M. (1976). New directions in cryptography. IEEE Transactions on Information Theory , 644-654.

Dimovski, A., & Gligoroski, D. (2003). Attacks on the Transposition Ciphers using Optimization Heuristics. Internationl Conference on Energy Sstems and Technology, (pp. 1-4). Sofia, Bulgaria.

Dorigo, M. (1992). Optimization, learning and natural algorithms. Milano.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents,. IEEE Transactions on Systems, Man, and Cybernetics--Part B, 26(1), 29-41.

Dureha, A., & Kaur, A. (2013). A Generic Genetic Algorithm to Automate an Attack on Classical Ciphers. International Journal of Computer Applications, 64(12), 20-25.

Eberhart, R. C. (1995). A new optimizer using particle swarm theory. International Symposium on Micromachine and Human Science, (pp. 39-43). Nagoya, Japan.

Eberhart, R. C., & Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. Congress of Evolutionary Computation (pp. 94-100). IEEE 2002.

Eberhart, R. C., Simpson, P., & Dobbins, R. (1996). Computational Intelligence PC Tools. AP Professional.

Feigenbaum, E., & Feldman, J. (1963). Computers and thought. McGraw-Hill.

Feistel, H. (1973). Cryptography and computer privacy. Scientific American, 228(5), 15-23.

Feistel, H. (1991). lock Cipher Cryptographic System. US Patent.

Felix, T. S., & Manoj, K. T. (2007). Swarm Intelligence: Focus on Ant and Particle Swarm Optimization. Vienna: Itech Education and Publishing.

Fogel, L., Owens, A., & Walsh, M. (1966). Artificial Intelligence through Simulated Evolution. Wiley.

Forsyth, W. S., & Savafi, R. N. (1993). Automated Cryptanalysis of Substitution Ciphers. Cryptologia, 407-418.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. Australian Journal of Biological Sciences, 484-491.

Friedman, W. (1920). The Index of Coincidence and its Applications in Cryptography. Riverbank Publications.

Gambardella, L. M., & Dorigo, M. (1997). Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1), 53-66.

Ganapathi, S., & Purusothaman, T. (2011). Reduction of Key Search Space of Vigenere Cipher using Particle Swarm Optimization. Journal of Computer Science, 1633-1638.

Gard, P., Varshney, S., & Bhardwaj, M. (2015). Cryptanalysis of Simplified Data Encryption Standard using Genetic Algorithm. American Journal of Networks and Communications, 32-36.

Garg, P. (2005). Genetic algorithms, Tabu Search and Simulated annealing: A comparison between three approaches for the cryptanalysis of Transposition Cipher. Journal of Theoritical and Applied Information Technology, 387-392.

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. STOC'09, (pp. 169-178). Maryland, USA.

Gheraibia, Y., & Moussaoui, A. (2013). Search Optimization Algorithm (PeSOA). recent Trends in Applied Artificial Intelligence: Lecture Notes in Computer Science, 222-231.

Giddy, J. P., & Savafi, N. R. (1994). Automated cryptanalysis of transposition ciphers. The Computer Journal, 429-436.

GLover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. Comput. & Ops, 13(5), 533-549.

Glover, F., & Laguna, M. (1997). Tabu search. Norwell, MA: Kluwer Academic Publishers.

Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional.

Goldberg, D., & Richardson, J. (1987). Genetic Algorithms whih Sharing for Multimodal Function Optimization. 2nd International Conference on Genetic Algorithms (pp. 41-49). Lawrence Earlbaum Associates.

Grassé, P. (1959). La reconstruction du nid et les coordinations inter-individuelles chez Belicositermes natalensis et Cubitermes sp. La théorie de la Stigmergie : Essai d'interprétation du comportement des termites constructeurs. Insectes Sociaux, 6, 41-80.

Grosek, O., & Zajak, P. (2009). Automated Cryptanalysis of Classical Ciphers. Encyclopedia of Artificial Intelligence, 186-191.

Hamdani, S. A., Shafiq, S., & Khan, F. A. (2010). Cryptanalysis of Four-Rounded DES Using Binary Artificial Immune System. ICSI (pp. 338-346). LNCS.

Hart, G. (1994). To decode short cryptograms. CACM, 37(9), 102-108.

Hart, P. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics , 100-107.

Hernández, J. e. (2002). Easing collision finding in cryptographic primitives with genetic algorithms. CEC2002, (pp. 535-539). Honolulu.

Heydari, M., Shabgani, G. L., & Heydari, M. M. (2013). Cryptanalysis of Transposition Ciphers with Long Key Using an Improved Genetic Algorithm. World Applied Sciences Journal, 21(8), 1194-1199.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press.

Hunter, D. G., & McKenzie, A. R. (1983). Experiments with relaxation algorithms for breaking simple substitution ciphers. The Computer Journal, 26(1), 68-71.

Husein, H. M., Bayoumi, B. I., Holail, F. S., Hasan, B. E., & Abd El-Mageed, M. Z. (2009). A Genetic Algorithm for Cryptanalysis with Application to DES-like Systems. International Journal of Network Security, 8(2), 177-186.

Jackobsen, T. (1995). A fast method for cryptanalysis of substitution ciphers. Cryptologia, 265-274.

Jadon, S. S., Sharma, H., Kumar, E., & Bansal, J. C. (2011). Application of Binary Particle Swarm Optimization in Cryptanalysis of DES. International Conference on SocProS, (pp. 1061-1071).

Jean-Marc, A., & Thomas, S. (1993). Intelligence Artificielle et Informatique Théorique. Éditions Cepadues.

Kahn, D. (1996). The Codebreakers — The Comprehensive History of Secret Communication from Ancient Time to the Internet. New York: Scribner.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. IEEE International Conference on Neural Networks (pp. 192-1948). Piscataway, NJ: IEEE Press.

Kennedy, J., Eberhart, R., & Shi, Y. (2001). Swarm Intelligence. Morgan Kaufmann Academic Press.

Kerckhoffs, A. (1883). La cryptographie militaire. Journal des sciences militaires, 5-38.

Khan, S., Ali, A., & Durrani, M. (2013). Ant-Crypto, a Cryptographer for Data Encryption Standard. International Journal of Computer Science Issue, 400-406.

Khan, S., Shahzad, W., & Khan, F. A. (2010). Cryptanalysis of Four-Rounded DES Using Ant Colony Optimization. International Conference on Information Science and Applications (pp. 1-7). Seoul: IEEE.

Kiefer, J. C., & Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. Annals of Mathematical Statistics, 23(3), pp. 462-466.

King, J. C. (1994). An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers. Cryptoogia, 18(4), 332-355.

Kirkpatrick, S., Gelatt, C., & Vecch, M. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.

Land, H A, & Doig, G A. (1960). An automatic method for solving discrete programming problems. Econometrica, 28, pp. 497-520.

Laskari, E. C., Meletiou, G. C., Stamatio, Y. C., & Vrahatis, M. N. (2005). Evolutionary computation based cryptanalysis: A first study. Nonlinear Analysis: Theory, Methods and Applications, e823-e830.

Luthra, J., & Pal, S. K. (2011). A hybrid Firefly Algorithm using genetic operators for the cryptanalysis of a monoalphabetic substitution cipher. World Congress on Information and Communication Technologies (pp. 202-206). Mumbai: IEEE.

Matsui, M. (1993). Linear cryptanalysis method for DES cipher. EuroCrypt93 (pp. 356-397). LNCS Springer.

Matsui, M. (1994). The first experimental cryptanalysis of the data encryption standard. Advances in Cryptology, 1-11.

Matthews, R. A. (1993). The use of genetic algorithms in cryptanalysis. Cryptologia, 17(2), 187-201.

Mayzner, M. S. (1965). Tables of Single-letter and Digram Frequency Counts for Various Word-length and Letter-position Combinations. Psychonomic Press.

Mekhaznia, T.,Menai, M.B, & Zidani,A (2012).Group swarm optimization for cryptanalysis of classical ciphers.META12. Sousse,Tunisia.

Mekhaznia, T.,Menai, M.B, & Zidani,A (2013).Nature inspired heuristics for cryptanalysis of Feistel ciphers.ICIST13, (pp. 48-55),.Tanger,Morocco.

Mekhaznia,T.,Menai, M.B, & Zidani,A (2014).Cryptanalysis of classical ciphers with ant algorithms. Int J. of Metaheuristics, 3(3), pp. 175-197.

Mendes, R., Kennedy, J., & Neves, J. (2004). The Fully Informed Particle Swarm : Simpler,Maybe Better. IEEE Trans. Evolutionary Computation, 8, pp. 204-210.

Menezes, A. J., Oorschot, P. C., & Scott, A. V. (1996). Handbook of Cryptography. CRC Press.

Mezmaz, M., Melab, N., & Talbi, E. (2007). grid-enabled branch and bound algorithm for solving chal lenging combinatorial optimization problems. IEEE Intl. Parallel and Distributed Pro cessing Symp. (IPDPS). California.

Michael, L., & Charles, R. (1988). How to cinstruct pseudorandom permutations from pseudorandom functions. SIAM Journal of Computing, 373-386.

Mishra, G., & Kaur, S. (2015). Cryptanalysis of Transposition Cipher Using Hill Climbing and Simulated Annealing. Dans S. India (Éd.), International Conference on Soft Computing for Problem Solving, 336, pp. 289-298.

Morelli, R. (2004). A Study of Heuristics Approaches for Breaking Short Cryptosystems. International Journal on Artificial Intelligence Tools, 45-64.

Mülhenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distribution I. Binary parameters. Lectures Notes in Computer Science 1411: Parallel Problem Solving from Nature, 179-187.

Nalini, N., & Rao, G. R. (2007). Attacks of simple block ciphers via efficient heuristics. Information Sciences, 177(12), 2553-2569.

Nelson, G., Sean, W., & Bas, A. (2000). Exploring Natural Language. British component-International Corpus of English.

Newell, A., Shaw, J., & Simon, H. (1957). Empirical explorations of the logic theory machine: A case study in heuristics. Western Joint Coputer Conference (pp. 218-230). New York: Institute of Radio Engineers, New York.

Nyberg, K. (1996). Generalized feistel networks. ASIACRYPT 1996 (pp. 91-104). LNCS Springer.

Olamaei, J., Niknam, T., & Gharehpetian, G. (2008). Application of particle swarm optimization for distribution feeder reconfiguration considering distributed generators. Applied Mathematics and computation, 201(1), 575-586.

Omran, S. S., Al-Khalid, A. S., & Al-Saady, D. M. (2011). A cryptanalytic attack on Vigenère cipher using genetic algorithm . IEEE Conference on Open Systems (pp. 59-64). Langkawi, Malysia: IEEE.

Omran, S. S., Al-Khalid, A., & Al-Saady, D. (2010). Using Genetic Algorithm to break a mono-alphabetic substitution cipher. Conference on Open Systems (pp. 63-67). Kuala Lumpur: IEEE.

Osman, I., & Kelly, J. (1996). Metaheuristics: theory and applications. Boston: Kluwers Academic Publishers.

Pandey, S., & Mishra, M. S. (2012). Particle Swarm Optimization in Cryptanalysis of DES. Int. J. of Advanced Research in Computer Engineering & Technology, 1(4), 379-381.

Papadimitriou C. (1994). Computational Complexity. Addison Wesley.

Patarin, J. (2004). Security of random feistel schemes with 5 or more rounds. CRYPTO 2004 (pp. 106-122). LNCS.

Pear, l. J. (1996). Heuristics: intelligent search strategies and applications. Boston: Kluwers Academic Publishers.

Peleg, S., & Rosenfeld, A. (1979). Breaking substitution ciphers using a relaxation algorithm. Communication of the ACM, 22(11), 598-605.

Picek, S., & Golub, M. (2011). On evolutionary computation methods in cryptography. Information Systems security MIPRO (pp. 1496-1501). Opatija, Croatia: IEEE.

Ramesh, R. S., Athithan, G., & Thiruvengadam, K. (1993). An automated approach to solve simple substitution ciphers. Cryptologia, 17(2), 202-218.

Raphael, C. P. (2007). Reducing the exhaustive key search of the Data eNCRYPTION sTANDARD (DES). Computer Standards and Interfaces, 528-530.

Rechenberg, I. (1965). Cybernetic Solution Path of an Experimental Problem . Royal Aircraft Establishment Library Translation.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. ACM Computer Graphics, 21(4), 25-34.

Rivest, R. (1992). The RC4 encryption algorithm. RSA Data Security.

Rivest, R. L. (1992). The MD4 Message-Digest Algorithm. Récupéré sur http://www.ietf.org/rfc/rfc1320.txt.

Rivest, R., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 120-126.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. Annals of Mathematicals & Statistics, 22(3), pp. 400-407.

Robert, L. (2000). Cryptological mathematics. The matimatical Association of America.

Russel, M. D., Clark, J. A., & Stepny, S. (2003). Making the most of two heristics: Breaking transposition ciphers with ants. Congress of Evolutionary Computation, (pp. 2653-2658).

Sacco, L. (1947). Manuel de cryptographie. Payot.

Sadiq, A. T., Ali, L., & Kareem, H. (2014). Attacking Transposition Cipher Using Improved Cuckoo Search. Journal of Advanced Computer Science and Technology Research, 4(1), 22-32.

Schaefer, E. (1996). A Simplified Data Encryption Standard Algorithm. Cryptologia, 20(1), 77-84.

Schneier, B. (1993). Description of a New Variable-Length Key, 64-Bit Block Cipher Blowfish. Fast Software Encryption, Cambridge Security Workshop Proceeding, 191-204.

Schneier, B. (1994). Applied cryptography. John Wiley & Sons.

Schneier, B., & Kelsey, J. (1996). Unbalanced feistel networks and block cipher design. Fast Software Encryption - FSE 1996 (pp. 121-144). LNCS Springer.

Shahzad, W., Siddiqui, A. B., & Khan, F. (2009). Cryptanalysis of Four-Rounded DES using Binary Particle Swarm Optimization. Genetic and Evolutionary Computation Conference (pp. 1757-1758). NY: ACM .

Shannon, C. E. (1949). Communication theory of secrecy systems. Bell System Technical Journal, 656-715.

Sharma, L., Pathak, B. K., & Sharma, N. (2012b). Breaking of Simplified Data Encryption Standard using Binary Particle Swarm Optimization. International Journal of Computer Science Issues, 307-313.

Sharma, L., Pathak, B. K., & Sharma, R. (2012a). Breaking os Simplified Data Encryption Standard Using genetic Algorithms. Global Journal Of Computer Science And Technology, 54-59.

Sharvani, G. S., Cauvery, N. K., & Rangaswamy, T. M. (2009). Different Types of Swarm Intelligence Algorithm . Int. Conf. on Advances in Recent Technologies in Communication and Computing. Kottayam, Kerala, India.

Shi, Y., & Eberhart, R. (1999). Empirical study of particle swarm optimization". Congress on Evolutionary Computation, (pp. 1945-1950).

Song, J., Zhang, H., Meng, Q., & Wang, Z. (2007). Cryptanalysis of Four-Round DES Based on Genetic Algorithm. International Conference on Wireless Communications, Networking and Mobile Computing (pp. 2326-2329). Shangai: IEEE.

Song, J., Zhang, H., Meng, Q., & Wang, Z. (2007). Cryptanalysis of Two-Round DES Using Genetic Algorithms. ISICA 2007 (pp. 583-590). Heidelberg: Springer.

Spillman, R. (1993). Cryptanalysis of knapsack ciphers using genetic algorithms. Cryptologia, 367-377.

Spillman, R., Janssen, M., Nelson, B., & Kepner, M. (1993). Use of a genetic algorithms in the cryptanalysis of simple substitution ciphers. Cryptologia, 31-44.

Standards, N. B. (1977). Data encryption Standard. Publication 46.

Stinson, D. R. (2006). Cryptography, Theory and Practice (Third Edition). BOCA RATON Floride: Chapman & Hall/CRC.

Storn, R., & Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization, 341-359.

Stütle, T., & Hoos, H. (2000). MAX-MIN Ant System. Future generation Computer Systems, 889-914.

Talbi, E. (2002). A taxonomy of Hybrid Metaheuristics. Journal of Heuristics, 8(5), 541-564.

Tariq, S. A., & Faez, H. A. (2014). Modification of Some Solution Techniques of Combinatorial Optimization Problems to Analyze the Transposition Cipher. Mathematical Theory and Modelling, 4(9).

Teytaud, F., & Fonlupt, C. (2014). A critical Reassessment of Evolutionary Algorithms On The Cryptanalysis of The Siplified data Encryption Standard. Journal on Cryptography and Information Security.

Tilborg, H. (2005). Encyclopedia of Cryptography and Security. Springer.

tlo@mit.edu, M. T. (2003, september 8). http://sourceforge.net. Consulté le Janvier 12, 2014, sur http://java-galib.sourceforge.net/

Toemeh, R., & Arumugam, S. (2007). Breaking Transposition Cipher with Genetic Algorithm. Electronic and Electrical Engineering, 75-78.

Toemeh, R., & Arumugam, S. (2008). Applying Genetic Algorithms for Searching Key-Space of Polyalphabetic Substitution Ciphers. The International Arab Journal of Information Technology, 87-91.

Uddin, M. F., & Youssef, A. M. (2006a). Cryptanalysis of simple substitution ciphers using particle swarm optimization. IEEE Congress on Evolutionary Computation, (pp. 677-680). Vancouver, CA: IEEE Press.

Uddin, M. F., & Youssef, A. M. (2006b). An Artificial Life Technique for the Cryptanalysis of Simple Substitution Ciphers. Canadian Conference on Electrical and Computer Engineering (pp. 1582-1585). IEEE.

Urszulan, B., & Dworak, K. (2014). Genetic Transformation Techniques in Cryptanalysis. (Springer, Éd.) Lecture Notes in Computer Science, 8398, 147-158.

Vaudenay, S. (2006). A classical Introduction to Cryptograph. Applications for communications security.

Verma, A. K., Dave, M., & Joshi, R. C. (2007). Genetic Algorithm and Tabu Search Attack on the Mono-Alphabetic Substitution Cipher in Adhoc Networks. Journal Of Computer Science, 134-137.

Vernam, G. S. (1926). Cipher printing Telegraph Systems for Secret Wire and Radio telegraphic Communications. Journal of the American Institute of electrical Engineers, 109-115.

Vimalathithan R., Valarmathi M. L. (2011b). Cryptanalysis of DES using Computational Intelligence. European Journal Of Scientific Research, V55(2):237-244.

Vimalathithan, R., & valarmathi, M. L. (2011a). Cryptanalysis of Simplified-DES using Computational Intelligence. WSEAS Transactions on Computers, 210-219.

Vimalathithan, R., & Valarmathi, M. L. (2012). Cryptanalysis of Simplified-AES using Particle Swarm Optimisation. Defence Science Journal, 117-121.

Vob, S. (1993). Intelligent search. TU Darmstadt.

Wafaa, G. A., Ghali, N. I., Hassanien, A. E., & Abraham, A. (2011). Known Plaintext Attack of DES-16 using Paticle Swarm Optimization. Third World Congress of nature and Biologically Inspired Computing (pp. 12-16). IEEE.

Wagner, D. (1999). The boomerang attack. (Springer, Éd.) lecture Notes in Computer Science, 1636, 156-170.

Wang, X., & Yu, H. (2005a). How to break MD5 and other hash functions. Lecture Notes in Computer Science, 19-35.

Wang, X., Yin, Y. L., & Yu, H. (2005b). Finding collisions in the full SHA-1. Advances in cryptology-CRYPTO 2005 (pp. 17-36). Berlin: Springer.

Wu, H., & Zhang, F. (2014). Wolf Pack Algorithm for Unconstrained Global Optimization. Mathematical Problems in Engineering.

Yang, X. (2010). A New Metaheuristic Bat-Inspired Algorithm. (Springer, Éd.) Studies in Computational Intelligence,, pp. 65-74.

Yean Li, H., Samsudin, A., & Belaton, B. (2005). Heuristic Cryptanalysis of Classical and Modern Ciphers. International Conference on Networks (pp. 1-6). Kuala Lumpur: IEEE