REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HAOUARI BOUMEDIENNE

FACULTE DE MATHEMATIQUES

DEPARTEMENT D'ALGEBRE ET THEORIE DES NOMBRES

MEMOIRE DE FIN D'ETUDES

Pour l'obtention du diplôme de PGS en cryptologie

THEME

DEVELOPPEMENT D'UN CRYPTOSYSTEME BASE SUR LE PROBLEME SAT.

Soutenu le:

Etudié par :

Mr BEN-AZIZ Sid-ali

Devant le jury :

Mr	**AÏSSANI Amar**	**(Président).**
Mme	**DRIAS Habiba**	**(Encadreur).**
Mr	**ZITOUNI Mohamed**	**(Examinateur).**
Mr	**FACI Hocine**	**(Examinateur).**
Mr	**MARMOUL Ataf**	**(Examinateur).**

Promotion 2002

Dédicace.

A mes très chers parents,

A ma très chère femme,

A ma très chère belle famille,

A mes très chers frères et ma chère sœur,

A toute la famille,

Et à tous mes amis.

Je dédie ce travail.

Sid-ali.

Remerciements.

Mes remerciements s'adressant tout d'abord à Madame Drias Habiba, ma directrice du projet, pour l'assistance qu'elle m'a prêtée afin de mener à mieux ce projet.

Aux hauts responsables de notre institution, qui m'ont donné cette opportunité de formation.

Aux hauts responsables de la DGSCT, qui grâce à eux a été mené à bien cette formation.

A tous les professeurs, qui ont veillé à nous enseigner leur savoir tout le long de notre formation.

A tous mes collègues de la PGS.

Sid-ali

SOMMAIRE

INTRODUCTION

En 1950, LE MATHEMATICIEN ALAN TURING publiait un article [1] qui donnait naissance à ce que l'on allait appeler plus tard l'intelligence artificielle. Il posait la question : les machines peuvent elles penser ? et proposait, pour tenter d'y répondre, un jeu nommé le jeu de l'initiation. Ce jeu consiste à prendre un homme « A », une femme « B » et un troisième joueur « C ». Les personnes A et B étant cachées, C doit deviner à l'aide de questions : qui est la femme ? et qui est l'homme ?. La personne B contrairement à A, doit faciliter la tâche de C. Alan TURING proposait de remplacer A ou B par un ordinateur et de regarder si le joueur C gagnait plus facilement ou non.

Son article est agrémenté d'argumentations concernant un ensemble de neuf objections (philosophique, mathématique…) que n'allaient manquer de donner les nombreux détracteurs (comme Hubert DREYFUS [2]) pensant que ce test ne serait jamais satisfait. Quant à Alan TURING, il imaginait que 50 ans plus tard, un ordinateur serait capable d'imiter suffisamment bien un des deux joueurs, pour que la personne C n'ait pas plus de 70% de chances de procéder à l'identification exacte de l'homme et de la femme au bout de 5 minutes. On arrive aujourd'hui aux termes de ces 50 ans et le test de TURING est loin d'avoir été satisfait. Les ordinateurs n'ont pas encore un comportement humain. Pour autant, de nombreuses recherches ont permis certaines avancées dans ce que l'on appelle "*l'intelligence artificielle"*.

Aujourd'hui, on peut définir "*l'intelligence artificielle"* comme l'ensemble des disciplines dont le but est d'imiter le comportement intelligent humain. Citons, par exemple, la traduction automatique, la démonstration automatique, la reconnaissance des formes, la compréhension de la parole, la modélisation de forme de raisonnement incertaine, temporelle et la cryptologie…

Une des disciplines de recherche en intelligence artificielle concerne la représentation et le traitement des connaissances. Le langage choisi doit avoir une grande expressivité et doit, en même temps être efficace, c'est à dire que les méthodes de déduction associées doivent être rapides. Malheureusement, ces deux critères sont difficilement cumulables.

Une des logiques les plus utilisées jusqu'ici a été la logique propositionnelle. Ce langage possède un grand nombre de qualités. Il est très simple mais permet de représenter une grande variété de connaissances. De nombreuses autres logiques peuvent s'y ramener. De plus, cette logique est décidable. C'est à dire qu'il existe toujours un algorithme vérifiant en un temps fini qu'une formule de la logique propositionnelle est satisfaisable ou pas.

Un autre atout de la logique propositionnelle est le problème **SAT** qui en est directement issu et auquel nous nous intéresserons dans la première partie de cette thèse. Le problème **SAT** consiste à déterminer si une formule propositionnelle mise sous forme normale conjonctive est satisfaisable. Ce problème a été le premier à être démontré, par Stephen COOK [3], comme étant NP-complet. Beaucoup d'autres problèmes qui s'y ramènent naturellement ont pu être classifiés par la suite. De nombreuses études, théoriques et pratiques, ont porté sur le problème **SAT**. Elles ont permis de créer des algorithmes, souvent basés sur l'algorithme énumératif de DAVID et PUTNAM [4.5] de plus en plus efficaces.

Le problème **SAT**, qui est le premier problème NP-Complet admet aujourd'hui de plus en plus d'applications, notamment dans la résolution d'encodages de problèmes « difficiles » réels, comme par exemple la cryptanalyse, ou encore la vérification de protocoles et de circuits. Les solveurs actuels de plus en plus performants et gérant les spécificités des problèmes réels nous semblent adaptés pour résoudre des problèmes comme le problème de la cryptanalyse « à la Masscci », qui consiste en un encodage du problème de cryptanalyse en problème **SAT**, ou comme les problème issus du Bounded Model Checking qui permettraient de vérifier des circuits liés à la cryptographie. Cependant la seule approche complète du problème **SAT** incarnée par la procédure de David et Putnam reste encore très coûteuse en temps processeur.

Dans une première partie, nous allons présenter globalement le problème **SAT** ainsi que ses principales techniques de résolution, nous y aborderons aussi les principaux problèmes réputés difficiles pour **SAT**.

I. GENERALITES SUR LA CRYPTOLOGIE

1. Introduction

Le passage de la préhistoire à l'Histoire s'est fait dès la création de l'écriture. En effet à partir du moment où l'on a pu conserver les grands faits de l'homme, elle a débuté. Sans l'écriture, la cryptologie n'aurait jamais existé. Cependant son élaboration ne s'est pas faite au hasard du temps. On peut penser comme David Kahn l'a dit que ce doit être dès que la culture a atteint un certain niveau, mesuré par sa littérature, que la cryptographie apparaît spontanément (comme ses parents l'écriture et le langage, l'ont probablement fait). Les besoins multiples de l'homme demandant de la confidentialité entre deux ou plusieurs personnes, au milieu de la société conduit inévitablement à la cryptographie.

Dans le monde actuel, le décryptement, opération qui consiste à rétablir le texte clair d'un document chiffré dont on ne connaît pas la clé, est la source la plus importante de renseignements secrets. Les informations qu'il procure, beaucoup plus nombreuses et plus sûres que celles fournies par l'espionnage, exercent une influence sur la politique des gouvernements. Cependant l'édification de cette science qu'est la cryptologie ne s'est pas faite en un jour. Elle regroupe la cryptographie et la cryptanalyse. Le mot "*cryptologie*" du grec *kruptos* (caché) et *graphein* (écrire) peut être assimilé à "étude des écritures secrètes". La cryptographie, c'est l'art de dissimuler ses intentions ou ses instructions à ses ennemis et pourtant de les transmettre à ses amis au moyen d'un texte chiffré. En face, chez l'adversaire, il s'agit de briser le code, de trouver le système qui préside à son élaboration : c'est la cryptanalyse.

La cryptologie est apparue dans de nombreux domaines tels que l'armée, le commerce, la religion... Elle a donc énormément influencé le cours de l'histoire même si elle est restée dans l'ombre. Mais quels sont les principaux faits historiques que la cryptologie a bouleversé jusqu'à nos jours et par quels moyens ?

Ce chapitre portera sur la cryptologie, un historique succinct sera présenté.

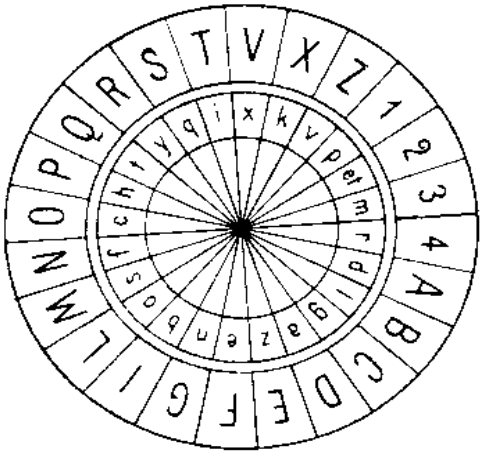
2. La naissance de la cryptologie

La cryptographie est une discipline ancienne. Elle a évoluée suivant différentes civilisations, l'Egypte,la Chine, l'Inde et la civilisation de l'antiquité, la Grèce.

Polybe, écrivain grec est à l'origine du premier procédé de chiffrement par substitution. C'est le carré de 25 cases :

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	ij	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

En 1467, l'italien Leon Batista Alberti inventa la substitution polyalphébetique



Cadran chiffrant d'Alberti

La substitution polyalphabétique évolua encore sous l'impulsion de Giovanni Batista Belaso, qui inventa la notion de clé littérale qu'il appela « mot de passe ».

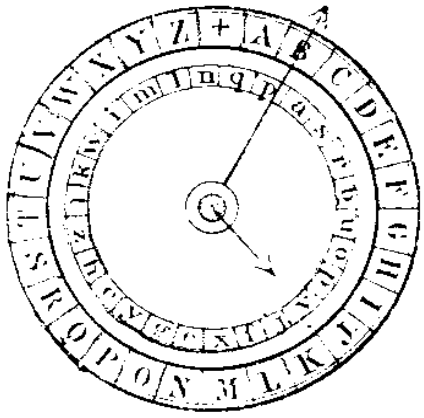
Clé littérale : BEL ASOBELA SOB ELASOB
Texte clair : LES ITALIENS ONT TROUVE

L'inventeur du second procédé est un français du nom de Blaise de vigenère. Il utilise un tableau, c'est le « carré de vigenère »

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tableau carré, dit « Carré de Vigenère »

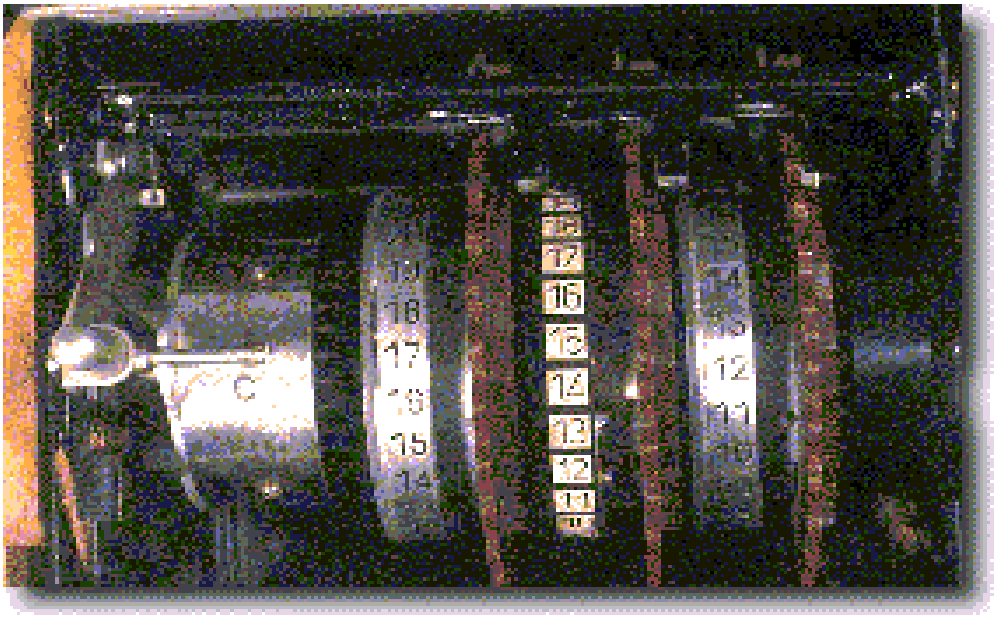
Un savant anglais du nom de Wheastone, enrichissait la cryptographie d'un nouveau procédé. Il s'agissait d'un cryptographe de type Alberti mais avec deux aiguilles semblables à celles d'une montre.



Le cryptographe de Wheatstone (alphabet clair à l'extérieur, cryptographique à l'intérieur)

Pendant les deux guerres mondiales, le savoir en cryptographie et cryptanalyse était très important dans le domaine militaire.

En 1923 fût le début de l'histoire de la machine de codage ENIGMA modèle A; par la suite, trois autres modèles apparurent B, C, D.



3. La Cryptologie actuelle

3.1. Les besoins actuels en cryptographie

De tous temps, les services secrets ont utilisé toutes sortes de codages et de moyens cryptographiques pour communiquer entre agents et gouvernements, de telle sorte que les "ennemis" ne puissent pas comprendre les informations échangées. La cryptologie a alors évolué dans ces milieux fermés qu'étaient les gouvernements, les services secrets et les armées. Ainsi, très peu de gens, voire personne n'utilisait la cryptographie à des fins personnelles. C'est pourquoi, pendant tant d'années, la cryptologie est restée une science discrète.

De nos jours en revanche, il y a de plus en plus d'informations qui doivent rester secrètes ou confidentielles. En effet, les informations échangées par les banques ou un mot de passe ne doivent pas être divulgués et personne ne doit pouvoir les déduire. C'est pourquoi ce genre d'informations est crypté. L'algorithme de cryptographie DES par exemple, est utilisé massivement par les banques pour garantir la sécurité et la

confidentialité des données circulant sur les réseaux bancaires. Le système d'exploitation Unix, lui aussi, utilise ce procédé pour crypter ses mots de passe.

Finalement, la cryptologie est de plus en plus utilisée sur le réseau mondial Internet. Avec l'apparition du commerce en ligne, c'est-à-dire la possibilité de commander des produits directement sur Internet, la cryptographie est devenue nécessaire. En effet, si les différents ordinateurs branchés sur Internet sont sécurisés par des mots de passe, c'est-à-dire à priori inaccessibles pour un ennemi, les transactions de données entre deux ordinateurs distants via Internet sont, quant à elles, facilement interceptées. C'est pourquoi lorsque l'on commande un produit sur Internet en payant avec notre carte bancaire, il est beaucoup plus sûr d'envoyer notre numéro de carte bancaire une fois crypté, celui-ci ne pourra à priori, être décrypté que par la société à laquelle on a commandé ce produit.

C'est pour ces mêmes raisons d'insécurité sur Internet, et par un besoin humain d'intimité que la cryptographie à des fins purement personnelles s'est développée sur le réseau : pour la messagerie électronique. En effet lorsque l'on envoie un message électronique par Internet, on peut préférer qu'il reste discret vis à vis de la communauté Internet, voire qu'il ne soit compréhensible que par le destinataire du message. En d'autres termes, la cryptographie peut servir si l'on veut envoyer un message confidentiel, ou un message intime à quelqu'un. Cela est aujourd'hui possible grâce à la formidable distribution de logiciels gratuits permettant d'utiliser de la cryptographie "forte" très facilement. C'est le cas du logiciel PGP (Pretty Good Privacy = "assez bonne confidentialité") qui est distribué gratuitement sur Internet, développé par Philip R. Zimmerman seul, en 1991. Ce sont pour toutes ces raisons que tout d'abord la cryptologie s'est énormément renforcée, et que finalement elle est passé d'un monde fermé comme les armées ou les services secrets à un monde ouvert à tout utilisateur.

3.2. Les méthodes de cryptographie actuelle

3.2.1. Le chiffrement actuel

Le chiffrement est l'action de transformer une information claire, compréhensible de tout le monde, en une information chiffrée, incompréhensible. Le chiffrement est toujours associé au déchiffrement, l'action inverse. Pour ce faire, le chiffrement est opéré avec un algorithme à clé publique ou avec un algorithme à clé privée.

3.2.2. Les algorithmes à clé privée ou à clé secrète

Les algorithmes à clé privée sont aussi appelés algorithmes symétriques. En effet, lorsque l'on crypte une information à l'aide d'un algorithme symétrique avec une clé secrète, le destinataire utilisera la même clé secrète pour décrypter. Il est donc nécessaire que les deux interlocuteurs se soient mis d'accord sur une clé privée auparavant, par courrier, par téléphone ou lors d'un entretien privé. La cryptographie à clé publique, quant à elle, a été inventée par Whitfield Diffie et Martin Hetlman en 1976 pour éviter ce problème d'échange de clé secrète préalable.

3.2.3. La préparation au cryptage

Une information de type texte, ou n'importe quel autre type d'information a besoin d'être codée avant d'être cryptée à l'aide d'un algorithme à clé publique ou privée. En d'autres termes, il faut fixer une correspondance entre une information et un nombre, puisque les algorithmes à clé (publique ou privée) ne peuvent crypter que des nombres. Le problème se résout facilement, puisque la plupart du temps, ce type de cryptographie est essentiellement utilisé sur des machines. Et comme de toute façon les informations sur une machine sont une suite de nombres, le problème est déjà très simplifié.

3.3. L'algorithme DES

3.3.1. L'histoire de DES

D.E.S., pour Data Encryption Standard ("standard de cryptage de données"), est un algorithme très répandu à clé privée créé à l'origine par IBM en 1977. Il sert à la cryptographie et l'authentification de données. Il a été jugé si difficile à percer par le gouvernement des Etats-Unis qu'il a été adopté par le ministère de la défense des Etats-Unis qui a contrôlé depuis lors son exportation. DES a été pensé par les chercheurs d'IBM pour satisfaire la demande des banques. Il a été conçu pour être implémenté directement en machine. En effet puisque les étapes de l'algorithme étaient simples, mais nombreuses, il était possible à IBM de créer des processeurs dédiés, capables de crypter et de décrypter rapidement des données avec l'algorithme DES. Cet algorithme a donc été étudié intensivement depuis les 15 dernières années et est devenu l'algorithme le mieux connu et le plus utilisé dans le monde à ce jour.

Bien que DES soit très sûr, certaines entreprises préfèrent utiliser le "triple-DES". Le triple-DES n'est rien d'autre que l'algorithme DES appliqué trois fois, avec trois clés privées différentes.

3.3.2. Description de l'algorithme DES

L'algorithme DES est un algorithme de cryptographie en bloc. En pratique, il sert à crypter une série de blocs de 64 bits (8 octets).

3.4. L'algorithme RSA

3.4.1. L'histoire de RSA

R.S.A. signifie Rivest-Shamir-Adleman, en l'honneur de ses inventeurs : Ron Rivest, Adi Shamir et Leonard Adleman qui l'ont inventé en 1977. Le brevet de cet algorithme appartient à la société américaine RSA Data Security, qui fait maintenant partie de Security Dynamics et aux Public Key Parteners, (PKP à Sunnyvale, Californie, Etats-Unis) qui possèdent les droits en général sur les algorithmes à clé publique. RSA est un algorithme à clé publique qui sert aussi bien à la cryptographie de documents, qu'à

l'authentification. Grâce au fait qu'il était à clé publique, et au fait qu'il était très sûr, l'algorithme RSA est devenu un standard de facto dans le monde.

3.4.2. Description de l'algorithme RSA

Tout le principe de RSA repose sur le fait qu'il est très difficile et très long de factoriser un très grand nombre en deux facteurs premiers.

3.5. La cryptanalyse actuelle

La cryptanalyse est l'étude des procédés de décryptage. Ou, plus généralement la science qui étudie la sécurité des procédés cryptographiques. Le cryptologiste est toujours cryptanalyste puisque qu'il doit en créant un algorithme de cryptographie s'assurer de sa sécurité, et pour ce faire, il a besoin de la cryptanalyse. La cryptanalyse tente de tester la résistance d'un algorithme de cryptographie en simulant différents types "d'attaques", qu'un ennemi pourrait effectuer s'il interceptait le document crypté. Un ennemi, en cryptologie, est une personne qui tentera, une fois le document crypté intercepté, d'opérer une attaque passive ou une attaque active.

4. Conclusion

La cryptologie a donc connu une rapide évolution à notre époque et c'est en partie due à deux facteurs essentiels : l'irruption des mathématiques et de l'informatique et le développement des moyens de télécommunications, qui a eu pour effet de multiplier les activités où intervient la cryptologie.

Le rôle de la cryptologie a évolué au fil des siècles. Avant elle n'avait pour rôle que de protéger un texte écrit ; maintenant, la cryptologie s'étend à différents domaines tels que la téléphonie, le télétraitement, le stockage des données, la communication avec les satellites...

Depuis une vingtaine d'années, la cryptologie s'est enrichie de nouvelles techniques de cryptage électronique. Il est vraisemblable que la cryptologie ne disparaîtra pas du fait des nouvelles techniques. Si elle devait disparaître, ce ne pourrait être que par suite d'une nouvelle conception des rapports humains. Le chiffre, lui non plus, n'est pas

prêt de disparaître puisqu'il a été et reste encore le moyen le plus sérieux d'assurer la sécurité des correspondances. Il tend de plus en plus vers une structure mathématique.

Il y a aujourd'hui une perpétuelle remise en cause de la cryptologie par la cryptanalyse. C'est une sorte de véritable combat. Les progrès de la cryptanalyse entraînent nécessairement des progrès en cryptologie et vice-versa. C'est une évolution sans fin. On peut se demander si la confidentialité des messages ne se trouve pas alternée dans ces progressions.

II. COMPLEXITE ET CLASSIFICATION DES PROBLEMES

1. Introduction

La théorie de la complexité a pour but de donner un contenu formel à la notion intuitive de résolution d'un problème; elle permet de dégager plusieurs niveaux de difficulté. C'est une aide très appréciable pour déterminer s'il est raisonnable ou non de se lancer dans la résolution d'un problème, ou s'il faut, au contraire, chercher à le reformuler pour aboutir à un problème facile. Il faut bien voir que le fait de dire qu'un algorithme est fini peut être d'un intérêt nul sur le plan opérationnel si son temps d'exécution est très grand. Ainsi, on sait qu'il existe une procédure finie permettant d'exhiber, à partir d'une situation quelconque du jeu d'échecs, une manière de jouer conduisant à coup sûr au gain de la partie – si toutefois une telle stratégie gagnante existe -. Mais comme on ne connaît pas, pour résoudre ce problème, d'autre voix que l'exploration systématique de tous les coups possibles pour l'un et l'autre joueur, ce résultat n'a guère de portée pratique si le temps de recherche dépasse de loin le temps maximal autorisé dans ce jeu.

2. Complexité

On se rend intuitivement compte que tous les problèmes pouvant être résolus par une machine de Turing ne sont pas de même difficulté... mais comment mesurer cette difficulté?

La difficulté d'un problème peut être mesurée par **le temps d'exécution** d'un algorithme qui le résoud, ou encore la **quantité de mémoire** requise lors de la mise en œuvre de cet algorithme.

Cependant, les notions de temps de traitement et d'espace mémoire sont dépendantes de la machine physique utilisée pour implémenter l'algorithme; on a donc besoin d'une mesure absolue, i.e. indépendante de toute implémentation.

La notion de machine de Turing va nous permettre de définir une telle mesure absolue de la difficulté d'un problème;

Elle sera appelée **la *complexité* (algorithmique) du problème.**

2.1. Définition

Considérons un problème P formalisable et décidable, et une machine de Turing T résolvant P.

Nous appellerons **complexité (temporelle) pire cas** (i.e. dans le cas le plus défavorable) de T pour P, le nombre maximal de déplacements de la tête de lecture/écriture que devra effectuer la machine T pour résoudre une instance de P de taille maximale *n* (où la taille *n* de l'instance est la longueur de la séquence binaire codant cette instance comme entrée de la machine de Turing).

Nous appèlerons **complexité (spatiale) pire cas** de T pour P, la longueur de bande nécessaire à la machine T pour résoudre une instance de P de taille maximale *n*.

2.2. Efficacité des algorithmes

La notion de complexité permet de définir deux grandes classes de problèmes:

- **Les problèmes *faciles :*** pouvant être résolus par une machine de Turing de complexité pire cas bornée par un polynôme en la taille des entrées.

Ces problèmes seront aussi appelés ***problèmes polynomiaux***, et les algorithmes permettant de les résoudre avec une complexité polynomiale seront appelés des ***algorithmes efficaces***.

- **Les problèmes *difficiles :*** (i.e. les problèmes formalisables, décidables, qui ne sont pas *faciles*) pour lesquels la complexité dans le pire cas n'est pas bornée par un polynôme...

Ces problèmes seront aussi appelés ***problèmes non polynomiaux***, et les algorithmes permettant de les résoudre seront appelés des ***algorithmes inefficaces***.

2.3. Influence du codage

Suivant la définition qui a été donnée de la complexité, cette dernière dépend de la manière dont les entrées sont codées... Or, il peut y avoir des codages plus ou moins efficaces !

Exemple: *codage des entiers positifs*

• Codage binaire efficace: i est codé par la séquence $i_0, i_1, ..., i_n$ tq :

$$i = \sum_{k=0}^{n} i_k \cdot 2^k \quad \text{avec } i_k \in \{0,1\}$$

La longueur $l_1(i)$ des codes de i est de l'ordre de $log_2(i)$.

• Codage binaire inefficace: i est codé par une séquence de 1 de longueur i.
La longueur $l_2(i)$ des codes de i est $\approx 2^{l1(i)}$.

En fait, on dira qu'un codage est sympatique si, pour toute entrée, la longueur de la séquence obtenue est bornée par un polynôme en la longueur de la séquence obtenue par un codage minimal.

2.4. Notation O(...)

Tant qu'il s'agit de déterminer la classe de difficulté d'un problème (polynomial ou non), ce qui importe est de savoir si cette complexité est ou non majorée par un polynôme.

Pour cette raison, on introduit généralement la notation de Landau O(...)

Définition

Pour deux fonctions f et g de N dans R, on écrit $f = O(g)$ **ssi,** il existe deux entiers n_0 et k, t.q : $\forall\ n \geq n_0, |f(n)| \leq k\ |g(n)|$

Dans ce cas, on dit que f est en $O(g)$.

L'intérêt de la notation de Landau est qu'on a le résultat suivant:

Un problème P est polynomial **ssi** il existe un entier k et une machine de Turing T résolvant P tq la complexité pire cas de T pour P, pour une entrée de taille n au sens d'un codage sympathique quelconque est en $O(n^k)$

Exemple de problème polynomial

la résolution par machine de Turing du calcul de parité est en $O(n)$ [dans ce cas, on parle d'algorithme ***linéaire***]. C'est donc un problème polynomial.

3. Classification des problèmes

On peut classer les problèmes de décision en trois catégories différentes et ceci selon leurs niveaux de difficulté :

- Problèmes polynomiaux.
- Problèmes non polynomiaux.
- Problèmes NP-Complet.

3.1. Les Problèmes Polynomiaux

Un problème est dit polynomial s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance. Un tel algorithme est dit *efficace* pour le problème en question.

Il existe une sous-classe des problèmes polynomiaux qui sont les problèmes linéaires (**P** est une fonction linéaire), qui constitue une classe de problèmes plus simples que ceux de **P**. Appartiennent à cette classe des problèmes triviaux comme le calcul de la somme de deux nombres de *p* chiffres…

Exemples de problèmes polynomiaux

La classe polynomiale comprend quelques problèmes classiques dont voici des exemples :

a. Tri d'un ensemble de *n* nombres;
b. Recherche des composants connexes d'un graphe ;
c. Recherche d'une chaîne qui passe par toutes les arêtes d'un graphe.

La classe des problèmes polynomiaux est à la fois importante et limitée. Importante parce que nombre de problèmes pratiques indispensables au bon fonctionnement de l'informatique ont des solutions polynomiales, limitées parce que pour nombre de problèmes intéressants, personne n'a réussi à montrer leur caractère polynomial.

3.2. Les problèmes NP

Il reste tout un ensemble de problèmes que nous ne pouvons pas classer dans P, car on ne connaît pas d'algorithme polynomial permettant de les résoudre, et que nous ne pouvons pas non plus les considérer comme intraitables, car on n'a pas prouvé qu'il n'existe pas d'algorithme polynomial permettant de les résoudre. L'introduction de la classe NP permet de pallier en partie à l'absence de ces résultats.

3.3. Les problèmes NP-COMPLETS

Un problème NP-Complet est un problème dont l'algorithme exact de résolution (qui procède par énumération de toutes les solutions possibles) prend un temps exponentiel par rapport à la taille des données.

Exemple

La classe des problèmes NP-Complets est très importante et nous ne citerons que quelques exemples considérés comme étant les plus importants ou les plus intéressants:

- Le problème **SAT**.
- Le problème du sac à dos.
- Partition d'un ensemble de nombres.
- Existence d'un stable dans un graphe.
- Existence d'un transversal dans un graphe.
- Existence d'un cycle hamiltonien dans un graphe.
- Le problème du voyageur de commerce.
- Planification de tâches indépendantes (ordonnancement).
- Le problème de l'affectation quadratique.

4. Conclusion

Au cours de ce chapitre, nous avons montré que la distinction entre les bons algorithmes et les mauvais peut être faite en exécutant les programmes correspondants sur machine (complexité empirique). Une meilleure façon de faire cette distinction est la complexité théorique, elle est définie comme une fonction mathématique qui permet de donner le temps d'exécution des algorithmes, en déterminant le nombre maximum d'instructions exécutées en fonction de la taille de la donnée du problème, et c'est grâce à cette complexité qu'on peut donner une classification aux problèmes.

Le chapitre suivant est consacré à l'état de l'art sur le problème **SAT**.

III. ETAT DE L'ART SUR SAT

1. Introduction

Une instance du problème de satisfaisabilité **SAT** est définie par un ensemble de variables booléennes et un ensemble de clauses formé à partir de ces variables. La question associée est de déterminer s'il existe une affectation de valeurs qui satisfait simultanément toutes les clauses. Le problème de satisfaisabilité appartient à la classe des problèmes NP-Complet.

Parmi les applications de problème **SAT,** la détection des défauts dans les systèmes (de communication, d'ordinateurs, etc.…) de grande taille, ainsi à la cryptologie, qui sera exploité par la suite dans ce Projet.

2. Notions mathématiques

Avant de se lancer dans la définition du problème SAT, quelques définitions issues de la logique mathématique sont peut être nécessaires pour la compréhension et seront utilisées ultérieurement :

- **Connecteurs :** On appelle connecteurs propositionnels, les symboles suivants :
  $\oplus$: où exclusif, $\leftrightarrow$ **:**Equivalence, $\rightarrow$ **:**Implication, $\wedge$ **:** Conjonction, $\vee$ **:** Disjonction.
- **Atomes :** Sont les énoncés dont on ne connaît pas (et on ne cherche pas à connaître) les structures internes. On les appelle aussi **variables propositionnelles.**
- **Littéral :** Un littéral est un atome (littéral positif), ou la négation d'un atome.
- **Formule booléenne :** On appelle formule booléenne, toute formule construite à partir des opérateurs logiques ($\neg$ **,** $\leftrightarrow$**,** $\rightarrow$**,** $\wedge$**,** $\vee$) et des variables propositionnelles. Nous avons les définitions suivantes :
  - Les atomes sont des formules.
  - Si α et β sont des formules alors $(\alpha \rightarrow \beta)$, $(\alpha \leftrightarrow \beta)$, $(\alpha \vee \beta)$ et $(\alpha \wedge \beta)$ sont des formules.
  - Si α est une formule alors $\neg \alpha$ est une formule.
  - $\alpha \wedge \neg \alpha$: est une formule qu'on note $\varnothing$ appelée clause vide.

- **Clause :** Une clause est une disjonction de littéraux $X_1 \vee X_2 \vee \ldots \vee X_n$.

- **Forme Normale Conjonctive (FNC) :** C'est une conjonction de clauses :

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m.$$

Instanciation : Soit E = { $X_1, X_2, X_3, \ldots, X_n$ }un ensemble de variables propositionnelles. On appelle instance de E, une fonction **I** telle que : **I**(E) : {faux ou vrai}.

Théorème : Toute formule booléenne admet une forme normale conjonctive qui lui est logiquement équivalente [15].

3. Définition du problème SAT

Le problème **SAT** est présenté comme étant la détermination de l'existence ou non d'une instanciation qui peut mettre à **Vrai** toutes les clauses d'une formule booléenne écrite sous FNC. Etant donné un ensemble de **n** variables propositionnelles X = { $X_1, X_2, X_3, \ldots, X_n$ }, et un ensemble de **m** clauses C = { $C_1, C_2, C_3, \ldots, C_m$ }, où chacune des clauses est la disjonction d'un ensemble de littéraux associés à X. Le problème **SAT**, consiste à trouver une instanciation de X pour satisfaire la formule suivante :

$$\mathbf{F} = \wedge \mathbf{C_i} \quad \text{avec } 1 \leq i \leq m$$
$$\mathbf{C_i} = \vee \mathbf{X_j} \quad \text{avec } 1 \leq j \leq n$$

C_i : Une clause.
X_j : Un littéral.

Une formule booléenne écrite sous FNC est dite satisfiable, s'il existe une instanciation qui met à vrai toutes les clauses la composant.

- Soit l'exemple suivant, F une formule booléenne écrite sous FNC :

$F = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_3)$.

F est satisfiable car pour (x_1 = vrai ou faux, x_2 = vrai, x_3 = vrai) F est à vrai.

- Soit maintenant la formule E suivante :

$E = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$.

E n'est pas satisfiable car il n'existe pas une instanciation pour les variables x_1, x_2 et x_3 qui satisfait à la fois toutes les clauses de E.

4. <u>Exemple</u> : Soit la donnée **SAT**

$$
\begin{cases}
C_1 = \neg x_1 \vee x_2 \vee \neg x_3 \\
C_2 = x_1 \vee x_4 \\
C_3 = x_1 \\
C_4 = x_2 \vee \neg x_3 \vee \neg x_4 \\
C_5 = \neg x_1 \vee \neg x_2 \vee \neg x_4 \\
C_6 = x_2 \vee x_3
\end{cases}
$$

La table de vérité de cette donnée ***(table I.1)*** montre que la formule est satisfiable pour deux instanciations :

X_1	x_2	x_3	x_4	C_1	C_2	C_3	C_4	C_5	C_6	F
0	0	0	0	1	0	0	1	1	0	0
0	0	0	1	1	1	0	1	1	0	0
0	0	1	0	1	0	0	1	1	1	0
0	0	1	1	1	1	0	0	1	1	0
0	1	0	0	1	0	0	1	1	1	0
0	1	0	1	1	1	0	1	1	1	0
0	1	1	0	1	0	0	1	1	1	0
0	1	1	1	1	1	0	1	1	1	0
1	0	0	0	1	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	0	0
1	0	1	0	0	1	1	1	1	1	0
1	0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	0	1	0
1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	0

Table I.1 : Table de vérité

En revanche, d'un point de vue informatique, sur le plan de la complexité de calcul, lorsque le nombre de variables est assez grand, il est impossible en pratique d'effectuer toutes ces opérations.

De ce fait, l'algorithme permettant de déterminer si une formule booléenne arbitraire est satisfiable, ne s'exécute pas en temps polynomial.

Il existe 2^n interprétations possibles d'une formule f à n variables. La vérification de chaque interprétation demande un temps super-polynomial (exponentiel). En se référant au théorème de Cook [3], on aura tendance à dire qu'un algorithme polynomial a très peu de chance d'exister.

5. La NP-Complétude du problème SAT

Cook à démontrer qu'effectivement le problème **SAT** est NP-Complet et a présenté cela sous forme d'un théorème :

Théorème de Cook :

La satisfiabilité d'une forme booléenne sous forme normale conjonctive est un problème NP-Complet.

6. Le problème 2-SAT

Le problème **k-SAT** peut être défini de la façon suivante :

Soit une proposition logique E sous forme normale conjonctive (i.e. une conjonction de disjonctions) de degré k (i.e. chaque disjonction contient exactement k littéraux).

Question : E est-elle satisfiable, i.e. existe-t-il une affectation de valeurs de vérité aux littéraux qui rende E vraie ?.

Exemple : une instance de **3-SAT**.

E = (A ∨ B ∨ ¬ C) ∧ (¬ A ∨ B ∨ ¬ C) où ¬ X représente la négation de X.

A	B	C	A ∨ B ∨ ¬ C	¬ A ∨ B ∨ ¬ C	E
0	0	0	1	1	1
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	1

E est donc satisfiable par exemple pour :

- A vraie.
- B vraie.
- C vraie.

- Montrons que **2-SAT** est un problème polynomial :

Une instance de **2-SAT** est une proposition logique de la forme :

$$\mathbf{E = (X_1 \vee Y_1) \wedge (X_2 \vee Y_2) \wedge \ldots\ldots\ldots \wedge (X_k \vee Y_k).}$$

Où les Xi, Yi appartiennent à un ensemble de littéraux, $\mathbf{L(E)=\{z_1,..,z_e,\neg z_1,..,\neg z_e\}}$.

Mais comme $\mathbf{(X_i \vee Y_i) \equiv (\neg \neg X_i \vee Y_i) \equiv (\neg X_i \Rightarrow Y_i) \equiv (\neg Y_i \Rightarrow X_i).}$

E peut être représenté sous la forme d'un graphe G(E), appelé graphe des implications, de la façon suivante :

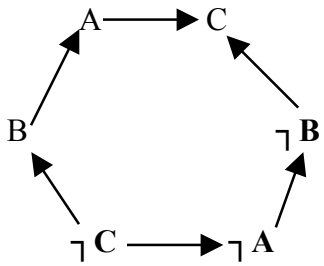
- Les nœuds de G(E) sont les éléments de L(E).
- Si E contient (Xi ∨ Yi) alors G (E) contient les deux arcs $\mathbf{(\neg X_i , Y_i)}$ **et** $\mathbf{(\neg Y_i , X_i).}$

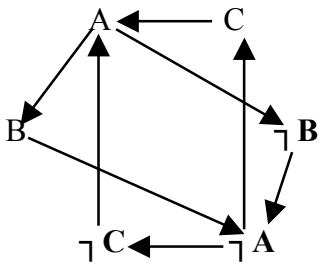
Exemple :

Si **E1 = (A ∨ ¬ B) ∧ (B ∨ C) ∧ (¬ A ∨ C).**

E2 = (A ∨ B ∨ ¬ C) ∧ (¬ A ∨ B ∨ ¬ C).

Alors : **$G(E_1)$** **et $G(E_2)$.**





On peut montrer que :

(1) E est satisfiable **ssi (2)** ∀i, G(E) ne contient pas de circuit passant par Zi et ¬ **Zi.**

Si pour tout graphe orienté **(2)** peut être vérifié en un temps polynomial en le nombre de sommets, cela implique que **(1)** est bien polynomial.

- Un intérêt est porté aux valeurs 2 et 3 de k. En effet 3 est la plus petite valeur de k pour laquelle **k-SAT** est NP-Complet, et **2-SAT** peut être résolu en un temps linéaire.(polynomial).

7. Algorithmes de résolution

7.1. Introduction

Il existe deux classes d'algorithmes de résolution du problème de satisfiabilité : les algorithmes complets et les algorithmes incomplets. On appelle algorithme incomplet un algorithme qui ne parcourt pas tout l'espace de recherche, ces algorithmes sont très efficaces, mais il se peut qu'ils n'aboutissent pas à une solution même dans le cas où le problème admet une solution. Un algorithme complet quant à lui est basé sur l'exhaustivité du parcours de l'espace de recherche et fournit donc une réponse exacte dans tous les cas.

Si l'on veut obtenir très rapidement un modèle d'une formule proportionnelle, on peut donc tout d'abord rechercher une solution avec un algorithme incomplet, si celui-ci ne donne pas de résultat, on pourra alors rechercher une solution avec un algorithme complet. Par contre si l'on cherche à montrer qu'une formule est non-satisfiable, les algorithmes incomplets ne nous sont d'aucune utilité, seul un algorithme complet pourra y parvenir, sachant que pour montrer qu'une formule est non-satisfiable, il est nécessaire de parcourir tout l'espace de recherche.

De manière plus précise les méthodes complètes sont toutes basées sur le principe général du « Frist Fail » qui vise à rechercher une contradiction le plus rapidement possible alors que les méthodes incomplètes sont toutes plutôt basées sur un principe général de « Réparation Locale » qui vise à améliorer une fonction objective.

7.2. Algorithmes incomplets

Les algorithmes incomplets sont principalement basés sur l'idée de la recherche locale : on part d'un certain point dans l'espace de recherche (une instanciation complète des variables), et on se déplace dans le voisinage du point de départ en essayant toujours d'améliorer au maximum la solution courante.

Dans le cas du problème **SAT**, on peut considérer le problème d'optimisation **Max-SAT** qui consiste à maximiser le nombre de clauses satisfaites, et si ce nombre est égal au nombre de clauses de la formule, on a trouvé un modèle pour la formule.

De plus, ces algorithmes mettent en œuvre différentes techniques pour sortir des maxima locaux que l'on appelle des stratégies d'échappement. Il est à noter qu'à l'inverse des algorithmes complets, les techniques de recherche locale ne tiennent pas compte de la longueur des clauses, et les clauses particulières comme les clauses unitaires sont considérées par ces algorithmes avec la même importance que les clauses de longueur supérieure à 2.

- GSAT[16].

Cet algorithme fut proposé par Selman, Levesque et Mitchell en 1992 et il a été à l'origine de l'essor des algorithmes incomplets basés sur la recherche locale pour le problème **SAT**. GSAT (G pour *Greedy*) commence par générer aléatoirement une position initiale, c'est à dire une interprétation de la formule, puis à chaque étape il change la valeur de la variable (il opère un *flip*) qui va dans le sens du meilleur gain.

Si au terme de MAXFLIPS inversions, GSAT n'a pas trouvé de solution, il considère qu'il a atteint un maximum local et redémarre le processus à partir d'une nouvelle position initiale aléatoire. Si au bout de MAXTRIES maxima locaux, GSAT n'a pas trouvé de solution, alors il conclut à un échec. On fixe bien entendu MAXFLIPS et MAXTRIES de manière arbitraire en fonction de la taille du problème à résoudre.

Un certain nombre de résultats théoriques ont été obtenus sur cette famille d'algorithmes en particulier par Papadimitriou [17] ; la simplicité de leur mise en œuvre et leur relative efficacité pour les très grands problèmes en font un algorithme encore très utilisé.

- Recherche tabou [18,19].

Cet algorithme qui est dû à Glover en 1986, fut aussi proposé simultanément par Hansen et Jaumard sous l'appellation « Steepest Ascent Mildest Descent » (SAMD). Il s'agit d'une variante de GSAT qui met en œuvre l'utilisation d'une « *liste tabou* » pour échapper aux maxima locaux. Il prend deux paramètres : MAXFLIPS et p pour effectuer MAXFLIPS mouvements locaux; si durant ces MAXFLIPS mouvement on ne trouve pas de meilleure solution, l'algorithme est terminé, sinon il repart sur MAXFLIPS mouvements. A chaque étape, on prend le flip qui maximise la fonction objective. Si on ne peut pas l'améliorer alors on effectue un mouvement qui va permettre d'échapper au maximum local et on interdit le flip de cette variable pendant p itérations.

- Marche Aléatoire [20].

Appelé couramment « Random Walk » ou Walksat , cet algorithme a été proposé par Selman en 1996 . A chaque étape, la marche aléatoire effectue un mouvement aléatoire, c'est-à-dire un flip aléatoire avec la probabilité p, et avec la probabilité 1-p effectue un mouvement de type GSAT (le meilleur flip possible). Comme dans GSAT on a MAXTRIES et MAXFLIPS pour gérer la boucle globale.

- Recuit Simulé.

Cet algorithme « Simulated Annealing » a été proposé par Kirkpatrick en 1979.

A chaque itération on génère un mouvement aléatoire. S'il augmente la fonction objective il est accepté, sinon il est accepté avec une certaine probabilité p. Ses paramètres, en plus de p sont MAXFLIPS a et t issus de la loi de la thermodynamique, avec MAXFLIPS le nombre d'itération dans un parcours, t la température initiale et a le facteur d'atténuation (élément de [0,1]) qui fait diminuer la température a chaque reprise.

7.3. Algorithmes complets

Présentation Générale

Les algorithmes complets sont basés sur un parcours implicite de tout l'espace de recherche associé a la formule de départ. Dans le cas d'une formule à n variables le parcours explicite reviendrait à évaluer la formule sur les 2^n interprétations possibles (méthode dite de la table de vérité Wittgenstein).

Les principaux algorithmes complets sont bien entendu plus rapides que la simple énumération de toutes les interprétations, mais ils souffrent encore de problèmes de performances (problème en général NP-Complet) face aux algorithmes incomplets dans le cas où le problème est satisfiable . Pour le cas non-satisfiable, ils sont les seuls algorithmes à répondre de manière certaine. Les algorithmes complets les plus courants sont les algorithmes basés sur les travaux fondateurs de David et Putnam.

Les deux principes de base: Saturation et Simplification

- Saturation

On effectue typiquement la saturation d'une formule par résolution. La résolution est un processus qui va générer de nouvelles clauses appelées des résolvantes pour essayer de saturer la formule (en tant qu'ensemble de clauses) jusqu'à obtenir la clause vide □ qui signifie que la formule est non-**SAT**, et le principe de la résolution est le suivant :

On considère deux clauses $C_1 = (x \vee v_1 \vee v_2 \vee \ldots \vee v_n)$ et $C_2 = (\neg x \vee w_1 \vee w_2 \vee \ldots \vee w_n)$.

On génère à partir de ces clauses une clause résolvante $C_3 = (v_1 \vee v_2 \vee \ldots \vee v_n \vee w_1 \vee w_{2 \ldots} w_n)$,cette clause étant la conséquence logique de C1 et C2, on peut donc la rajouter à l'ensemble des clauses de la formule de départ sans changer sa nature logique. Cas particulier : si n = m = 0 alors C3 = □ .

Propriété : F est non-satisfiable ssi □ peut être générée par résolution.

- Simplification.

Les trois principales techniques de simplification sont les suivantes :

Propagation unitaire : La propagation unitaire est une opération qui supprime les clauses unitaires, elle est constituée d'un seul littéral, or on désire satisfaire cette clause, donc on fixe la valeur de vérité du littéral. Cette assignation est ensuite « propagée » à tout le reste de la formule en suivant le principe de simplification habituel, à savoir : toutes les clauses contenant une occurrence de la variable s'évaluant à vrai sont supprimées car elles sont forcément satisfaites, et toutes les occurrences de la variable dans la formule s'évaluant à faux sont supprimées.

Règle du littéral pur : Cette opération est très semblable à la propagation unitaire, on recherche dans la formule tous les littéraux purs et on supprime toutes les clauses dans lesquelles ils apparaissent en assignant la valeur correspondante au littéral à la variable. Cette règle qui est plus coûteuse à mettre en œuvre que la première a été démontrée peu efficace en pratique.

Subsomption : Cette opération consiste à rechercher dans la formule les clauses dont tous les littéraux appartiennent à une autre clause. Si tous les littéraux de C_1 appartiennent à C_2 alors on peut supprimer C_2 sans modifier la nature logique de la formule de départ. Cette règle qui est très coûteuse à mettre en œuvre (complexité O (m^2)) n'est jamais implémentée dans les solveurs .

- La procédure DPLL

DP60 : DP60 est la première procédure de David et Putman [21], qui à été proposée en 1960. Son fonctionnement est des plus simples, on cherche à supprimer d'une part toutes les variables d'une formule propositionnelle en utilisant comme nous l'avions décrite ci-dessus. Ainsi, pour une variable x donnée DP60 va ajouter à la formule tous les résolvants possibles obtenus à partir de clauses faisant intervenir x et ┐x. On génère alors pour chaque variable x, ayant dans la formule n occurrences de x et m occurrences de ┐x, n x m clauses résolvantes, mais on peut supprimer uniquement les clauses dont ces clauses résolvantes sont la conséquence logique, soit n + m. De ce fait à chaque itération de l'algorithme on supprime un certain nombre de variables mais on peut sous certaines conditions augmenter le nombre de clauses de façon exponentielle. Pour simplifier la

formule à chaque itération, on incorpore dans l'algorithme la propagation unitaire et la règle du littéral pur. La terminaison de l'algorithme est gérée de la manière suivante :

- Soit l'algorithme génère un résolvant qui est une clause vide, dans ce cas le problème est non-**SAT**.
- Soit on ne peut plus générer aucun résolvant, le problème est **SAT,** et à ce moment chacune des clauses restantes représente une solution.

DP60 pose surtout des problèmes d'explosion mémoire car il procède par saturation; de ce fait il est difficilement implémentable en pratique, On lui préfère la plus part du temps la procédure DP62 décrite ci-après.

DP62 : Cette version, communément appelée algorithme David et Putnam et en effet due à David, Logimann et Loviland (1962) [22], d'où l'appellation « DPLL ». Contrairement à DP60, dans cette version on n'utilise pas la résolution, mais on fractionne plutôt le sous problème en deux, on branche selon une variable, c'est-à-dire on choisit une variable v dans la formule et on génère les deux formules obtenues à partir de la formule initiale, en instanciant v à vrai et à faux, on fait alors les simplifications nécessaires (soit de supprimer les clauses contenant le littéral à vrai et les occurrences du littéral à faux) et on utilise la propagation unitaire.

DPLL est un algorithme récursif dans lequel on peut voir le parcours comme un arbre binaire dont la racine est constituée de la formule de départ et les nœuds contiennent les formules obtenues suite au instanciations des variables et aux différentes simplifications appliquées. Le parcours de cet arbre s'effectue en profondeur et nécessite la connaissance d'informations de retour en arrière où Backtrak. Aux feuilles de l'arbre on a soit un ensemble vide de clauses, dans ce cas la formule possède une solution correspondant au différentes instanciations de variables de la racine jusqu'à cette feuille, soit un sous-ensemble de clauses contenant une clause contredite, dans ce cas la branche est réfutée et il n'est plus la peine de poursuivre la recherche dans cette branche.

8. CONCLUSION

Nous avons montré dans ce chapitre que la satisfiabilité propositionnelle est un problème important de la théorie de la complexité et de l'intelligence artificielle. Il consiste à déterminer pour une formule du calcul propositionnel, l'existence d'une instanciation de ses variables pour laquelle la formule est évaluée à vrai, et nous avons montré que le problème **SAT** est NP-Complet.

Par la suite, nous avons présenté deux classes d'algorithmes de résolution du problème de satisfiabilité : les algorithmes complets et les algorithmes incomplets, et nous avons montré que la résolution du problème **SAT** est très difficile pour une grande taille.

Dans le chapitre suivant nous allons présenter quelques cryptosystèmes existants, tels que le RSA et le SAC à DOS.

IV. QUELQUES CRYPTOSYSTEMES EXISTANTS

1. Le cryptosystème RSA

Introduction

Le RSA a été inventé par Rivest, Shamir et Adlemen en 1978. C'est l'exemple le plus courant de cryptographie asymétrique, toujours considéré comme sûr, avec la technologie actuelle, pour des clés suffisamment grosses (1024,2048,4096 bits). D'ailleurs le R.S.A128 (algorithme avec des clés de 128 bits), proposé en 1978 par Rivest, Shamir et Adleman, n'a été « cassé » qu'en 1996, en faisant travailler en parallèle de nombreux ordinateurs sur internet.

Mais le concept de chiffrement asymétrique avec une clef publique était légèrement antérieur (1976). L'idée générale était de trouver deux fonctions f et g sur les entiers, telles que f o g = Id, et tel que l'on ne puisse pas trouver f, la fonction de décryptage, à partir de g, la fonction de cryptage. L'on peut alors rendre publique la fonction g (ou clef), qui permettra aux autres de crypter le message à envoyer, tout en étant les seuls à connaître f, donc à pouvoir décrypter.

L'algorithme RSA

Description du protocole

Le but du jeu est bien sûr de pouvoir transmettre un message codé, que seul le récepteur « officiel » puisse décrypter, c'est-à-dire qui ne puisse pas être décrypté par n tiers qui intercepterait ledit message. Nous appellerons Alice la destinatrice du message, et Bernard l'émetteur.

1. Alice génère deux gros nombres premiers p et q, ainsi qu'un gros nombre d premier avec le produit w = (p-1) x (q-1).

2. Alice calcule n = p x q et e tel que d x e = 1 mod w.

3. Alice diffuse n et e, garde d et oublie w.

4. Bernard crypte un message M par $M \rightarrow M^e \bmod n$ et envoie le résultat C à Alice.

5. Alice décode alors le message crypté par $C \rightarrow C^d \bmod n$.

Exemple

Voyons ce qui se passe, si l'on prend pour nombres p et q les valeurs 11 et 17. On a alors n = 187 et w = (11-1) x (17-1) = 160. Comme 161 = 7 x 23, on peut prendre e = 7 et d=23. Alice va rendre publique le couple (187, 7).

Bernard veut transmettre à Alice un message codé plus petit que n = 187, mettons la date à laquelle ils vont faire une surprise à Cédric (par exemple, le 10), message qui ne doit pas être intercepté par ledit Cédric, bien sûr.

Bernard va calculer $10^7 = 187 \times 53475 + 175$, et envoyer le résultat 175 à Alice.

Alice va calculer le reste de la division euclidienne de 175^{23} par 187 :

- Elle calcule d'abord $175^2 = 30625 = 187 \times 163 + 144$, donc $175^2 = 144 \bmod 187$.
- Ensuite, $144^2 = 20736 = 187 \times 110 + 166$, donc $175^4 = 166 \bmod 187$.
- Puis, $166^2 = 27556 = 187 \times 147 + 67$, donc $175^8 = 67 \bmod 187$.
- Et $67^2 = 4489 = 187 \times 24 + 1$, donc $175^{16} = 1 \bmod 187$.
- Enfin, $175^{23} = 175^{16} \times 175^4 \times 175^2 \times 175$.donc $175^{23} = 1 \times 166 \times 144 \times 175 \bmod 187$. or $166 \times 144 \times 175 = 4183200 = 187 \times 22370 + 10$.

Alice retrouve donc bien le message envoyé, à savoir 10.

Preuve de l'algorithme

Le but du protocole est bien sûr qu'Alice retrouve le message d'origine. Les transformations successives appliquées au message d'origine sont :

$M \rightarrow M^e \bmod n \rightarrow (M^e \bmod n)^d \bmod n$.

- Si le message M est premier à n :

On a $(M^e \bmod n)^d \bmod n = M^{ed} \bmod n$, et par hypothèses $d \times e = 1 \bmod w$, c'est-à-dire que $d \times e = 1 + k \times w$ (1) , avec k un entier.

Alors, on peut appliquer le Théorème d'Euler1, M est premier à n donc $M^{\varphi(n)} = 1 \bmod n$, et d'après la proposition 18, comme on a : $n = p \times q$, on sait que $\varphi(n) = (p-1) \times (q-1) = w$. on a donc :

$M^w = 1 \bmod n$, donc $M^{ed} = M^{kw+1} = M \times 1^k \bmod n = M \bmod n$ (2).

Et donc on revient bien ainsi au message original.

- Sinon,M non premier à n=p x q, c'est-à-dire que M est multiple de p ou de q.

Considérons le cas où M est de la forme p^{α} x m (3), avec m entier non multiple de p. Comme M<n, on peut affirmer que m est premier à n (sinon, M serait à la fois multiple de p et de q, donc plus grand que n).

Et on a : $M^{ed} = (p^{\alpha} \text{ x } m)^{ed} \bmod n = p^{\alpha \text{ x } ed} \text{ x } m^{ed} \bmod n = p^{\alpha \text{ x } ed} \text{ x } m \bmod n$ (4).

D'après ce qui procède, appliquer à m

Or $p^{\alpha \text{ x } ed} = p^{\alpha} \bmod p$ (5), et $p^{\alpha \text{ x } ed} = p^{\alpha} \bmod q$ (6).

Alors (5) et (6) impliquent que la différence $p^{\alpha \text{ x } ed} - p^{\alpha}$ est à la fois multiple de p et de q, donc multiple de p x q, donc on a bien $p^{\alpha \text{ x } ed} = p^{\alpha} \bmod n$ (7), et donc :

$$M^{ed} = p^{\alpha} \text{ x } m \bmod n = M \bmod n \quad (8).$$

Alice retrouve bien le message original.

Les nombres p et q jouant des rôles identiques, le cas où M est multiple de q est identique.

2. Le cryptosystème Sac à Dos

2.1. Introduction

Soit un Sac à Dos vide, et m objets de poids respectifs $a_{1,\ldots\ldots\ldots,}a_m$. Quels objets doit-on mettre dans le Sac à Dos pour obtenir un poids total égal à k ?

Il s'agit là d'un problème très difficile, à la fois en pratique (il est insoluble, même avec un ordinateur, pour de grandes valeurs de m), et en théorie (il fait partie des problèmes NP-complets). Il est très difficile, sauf dans le cas particulier où la suite (a_n) est supercroissante , c'est-à-dire si pour tout entier i :

$$\sum_{j=1}^{i-1} a_j < a_i$$

Dans ce cas, on commence par comparer les valeurs de k et de a_m :

• Si $k < a_m$, il ne faut bien sûr pas prendre a_m.

• Sinon, il faut prendre obligatoirement a_m : si on ne le prenait pas, même en prenant tous les autres poids $a_{1,\ldots\ldots,}a_{m-1}$, on aurait un poids total inférieur strict à m, et a fortiori à k.Il suffit ensuite de résoudre le problème du Sac à Dos pour les poids $a_{1,\ldots\ldots,}a_{m-1,}$ avec le poids total $k-a_m$.

2.2. L'algorithme Sac à Dos

Il est intellectuellement séduisant de baser un algorithme de cryptographie sur le problème du Sac à Dos. En effet, contrairement au problème de la factorisation d'entiers, ce problème est prouvé mathématiquement comme étant difficile, c'est à dire NP-complet. Merkle et Hellman ont proposé d'utiliser un Sac à Dos supercroissant comme clé privée, et de le camoufler sous un Sac à Dos général pour en faire une clé publique. Plus en détails, voici ce que cela donne :

● On se donne un entier m, une suite supercroissante $a_1,\ldots\ldots..,a_m$, et un autre entier M vérifiant :

$$\sum_{j=1}^{m} a_j < M$$

On dispose ainsi d'un Sac à Dos supercroissant.

● On choisit au hasard un entier W, compris entre 1 et M-1,et premier avec M-1.

● On choisit au hasard une permutation σ de l'ensemble {1,…..,m}.

● On calcule $bi= a_{\sigma(i)}W \ mod \ M$ pour tous les i. On vient ainsi de camoufler le problème du Sac à Dos supercroissant en Sac à Dos général.

La clé publique du système est la suite (a priori quelconque) $b_{1,}.....,b_m$, la clé privée consiste en les deux entiers M et W, la permutation σ, et la suite supercroissante $a_{1,\ldots\ldots,}a_{m.}$ Si on souhaite envoyer au détenteur du système (Bob) le message constitué de la suite de m bits $x_1,\ldots..,x_m$, on calcule d'abord :

$$C = \sum_{i=1}^{m} x_i a_i$$

puis on lui envoie C. quelqu'un qui intercepte C et connaît la clé publique $b_{1,}.....,b_m$ ne peut remonter aux x_i, car la résolution du Sac à Dos général est un problème très compliqué.

En revanche, Bob qui connaît la clé privée retrouve les x_i de la façon suivante : il calcule d'abord :

$$D = W^{-1}C \bmod M.$$

Il résoud ensuite le Sac à Dos supercroissant avec le poids total D, et les poids $a_{1,}.....,a_m$. Il calcule des nombres e_i valant 0 ou 1 et tels que :

$$D = \sum_{i=1}^{m} e_i a_i$$

Il en déduit les x_i par la relation :

$$x_i = e_{\sigma(i)}.$$

Justifions cette phase de déchiffrement. On a en effet, modulo M :

$$D = W^{-1} C = W^{-1} \sum_{i=1}^{m} x_i b_i.$$

$$= W^{-1} \sum_{i=1}^{m} x_i a_{\sigma(i)} W.$$

$$= \sum_{i=1}^{m} x_i a_{\sigma(i)}.$$

2.3. Application numérique

- **Fabrication des clés publiques et privées :**

  - On choisit m=5 et la suite supercroissante :

    $a_1 = 2, a_2 = 5, a_3 = 11, a_4 = 23, a_5 = 55.$

  - Soit M = 113, W = 27 (premier avec M), et la permutation :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix}$$

  - Le calcul de $b_i = a_{\sigma(i)} W \bmod M$ donne :

    $b_1 = 22, b_2 = 16, b_3 = 71, b_4 = 54, b_5 = 56.$

- **Envoi du message** : on veut envoyer le message 10101 (en binaire). On transmet donc : $C = b_1 + b_3 + b_5 = 149$. Remarquons que, comme prévu, lorsque l'on connaît 149 et les poids 22, 16, 71, 54 et 56, il n'est pas du tout évident de retrouver ceux qui ont servi à former 149.

- **Réception du message** : on calcule W^{-1} par l'algorithme d'Euclide, et on trouve $W^{-1} = 67$. Le calcul de $W^{-1} C$ donne 39 (tous les calculs s'effectuent modulo 113). On redécompose 39 dans la suite supercroissante :

$$\begin{aligned} 39 &= 23 + 11 + 5. \\ &= a_4 + a_3 + a_2 \\ &= a_{\sigma(5)} + a_{\sigma(3)} + a_{\sigma(1)}. \end{aligned}$$

Le message initial était donc 10101.

3. Conclusion

Dans ce chapitre nous avons expliqué le fonctionnement de cryptosystèmes existants tels que le RSA et le SAC à DOS, et nous avons montré que le cryptosystème est l'action de transformer une information claire compréhensible de tout le monde, en une information cryptée, incompréhensible. Le cryptage est toujours associé au décryptage, l'action inverse. Pour ce faire le cryptage est opéré avec un algorithme à clé publique ou avec un algorithme à clé privée.

Dans le chapitre suivant nous allons essayer de fabriquer une fonction à sens unique **2-SAT** à **3-SAT** pour ensuite fabriquer les clés privées et publiques, et par la suite on va proposer un cryptosystème basé sur le problème **SAT**.

V. PROPOSITION D'UN CRYPTOSYSTEME BASE SUR SAT

1. Introduction

Les algorithmes à clé privée sont aussi appelés algorithmes symétriques. En effet, lorsqu'on crypte une information à l'aide d'un algorithme symétrique avec une clé secrète, le destinataire utilisera la même clé secrète pour décrypter. La cryptographie à clé publique, quant à elle, a été inventée par Whitfield Diffie et Martin Hetlman en 1976 pour éviter ce problème d'échange de clé secrète préalable. Les algorithmes à clé publique sont aussi appelés algorithmes asymétriques; c'est à dire que pour crypter un message, on utilise la clé publique (connue de tous) du destinataire, qui sera à priori le seul à pouvoir le décrypter à l'aide de sa clé privée (connue de lui seul).

Dans ce chapitre, on va essayer de fabriquer une fonction à sens unique de **2-SAT** qui est un problème polynomial (facile à résoudre) à **3-SAT** qui fait partie des problèmes NP-Complet (très difficile à résoudre) qu'on va utiliser comme clé publique. Et par la suite, on va proposer un cryptosystème basé sur le problème **SAT**, en utilisant un problème **2-SAT** camouflé sous un problème **3-SAT**.

2. LA FONCTION A SENS UNIQUE DE 2-SAT A 3-SAT

Le problème **2-SAT** peut être défini de la façon suivante :

Soit une proposition logique E sous forme normale conjonctive (i.e. une conjonction de disjonctions) de degré 2 (i.e. chaque disjonction contient exactement 2 littéraux).

Question : E est-elle satisfiable, i.e. existe-t-il une affectation de valeurs de vérité aux littéraux qui rend E vraie ?.

Exemple : E = (A ou B) et (A ou ┐B) et (B ou ┐C).

Et de même, le problème **3-SAT** est une proposition logique E sous forme normale conjonctive de degré 3 où chaque disjonction contient trois littéraux.

Exemple : E = (A ou B ou ┐C) et (┐A ou B ou ┐C).

Sachons que le problème **2-SAT** est polynomial, et le problème **3-SAT** est NP-Complet, la fonction à sens unique consiste à trouver une transformation de **2-SAT** à **3-SAT**.

La méthode de cette transformation est la suivante :

- Soit E une proposition logique **2-SAT** sous forme normale conjonctive, par exemple :

$$\begin{cases} C_1 = \neg x_1 \vee x_2 \\ C_2 = x_1 \vee x_4 \\ C_3 = x_1 \vee \neg x_5 \\ C_4 = x_2 \vee x_3 \\ C_5 = x_4 \vee x_6 \\ C_6 = x_2 \vee \neg x_5 \\ C_7 = x_2 \vee x_4 \end{cases}$$

Les variables de ce problème sont : $X = (x_1, x_2, \ldots, x_6)$

- Soit une affectation X = (0, 1, 1, 1, 0, 1) qui rend ce système satisfait, c-à-d : pour $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$, $x_5 = 0$ et $x_6 = 1$, la proposition logique E = 1.

- Dans cette étape, on va ajouter à chaque clause une variable qui doit avoir une valeur fausse, pour ne pas changer la nature de chaque clause et donc du système **2-SAT**.

- Par la suite, on obtient un système **3-SAT** avec un vecteur V, où chaque élément de ce vecteur représente le rang de la variable ajoutée dans chaque clause du nouveau système **3-SAT**.

- Chaque élément du vecteur V doit avoir une valeur égale à 1, 2, ou 3.

- Donc, en conséquence, à partir d'un problème **2-SAT**, on a obtenu un problème **3-SAT** équivalent (c-à-d : que les deux systèmes ont les mêmes solutions), et un vecteur V qui désigne les rangs des variables ajoutées au système **2-SAT**.

- A la fin, on extrait deux sortes de clés :

  - Le système **3-SAT**, comme clé publique.
  - Le vecteur V, comme clé privée.

2.1. L'algorithme de la fonction à sens unique 2-SAT/3-SAT

Entées

- Soit un problème **2-SAT** de n variables et m clauses :

$X = \{ x_1, x_2, \ldots, x_n \}$ et $C = \{ C_1, C_2, \ldots, C_m \}$.

- Chaque clause est de la forme : $C = x_i + x_j$.

- Soit S le vecteur qui représente la solution de notre système **2-SAT**, donc S est de la forme : $S = (s_1, s_2, \ldots, s_n)$.

Sorties

- Le vecteur V de dimension m qui désigne les rangs des variables ajoutées au système 2-SAT, $V = (v_1, v_2, \ldots, v_m)$.

- Un problème **3-SAT** de n variables et m clauses issu de problème **2-SAT**.

ALGORITHME

P ⟵ 1,

Pour k allant de 1 à m **faire**

Trouve ⟵ faux,

Tant que ┐ Trouve **faire**

Si P < i **alors**

- On ajoute à la clause C_k la variable ayant l'indice P à la première position,
- Si $s_p = 0$, on va ajouter (x_p) à la clause C_k,
- Si $s_p = 1$, on va ajouter (┐ x_p) à la clause C_k,
- V[k] = 1 (c-à-d, (le rang de x_p) = 1),
- Trouve ⟵ vrai,

Sinon

Si (p < j) et (p > i) alors

- On ajoute à la clause C_k la variable ayant l'indice P à la deuxième position,
- Si $s_p = 0$, on va ajouter (x_p) à la clause C_k,
- Si $s_p = 1$, on va ajouter (┐ x_p) à la clause C_k,
- V[k] = 2 (c-à-d, (le rang de x_p) = 2),
- Trouve ⟵ vrai,

Sinon

Si P > j alors

- On ajoute à la clause C_k la variable ayant l'indice P à la troisième position,
- Si $s_p = 0$, on va ajouter (x_p) à la clause C_k,
- Si $s_p = 1$, on va ajouter (┐ x_p) à la clause C_k,
- V[k] = 3 (c-à-d, (le rang de x_p) = 3),
- Trouve ⟵ vrai,

FSi

FSi

FSi

FTQ

Si P < n alors

P ⟵ P+1,

Sinon

P ⟵ 1,

FSi

FPour

En fin, on a généré un problème **2-SAT** camouflé sous un problème **3-SAT** qui est la clé publique, et la clé privée qui est le vecteur V.

2.2. La fonction inverse de la fonction à sens unique 2-SAT/3-SAT

Entrées

- Soit le vecteur V de dimension m qui désigne les rangs des variables supplémentaires au système **2-SAT** : $V = (v_1, v_2, \ldots, v_m)$.
- Soit un problème **3-SAT** de n variables et m clauses

Sorties

- Un problème 2-SAT de n variables et m clauses.

ALGORITHME

Pour chaque clause C_k **faire**

- Enlever la variable x_i qui a le rang v[k] dans la clause C_k,

FPour

2.3. Application numérique

- Soit le système **2-SAT** de 6 variables et 7 clauses suivant :

$C_1 = \neg x_1 \vee x_2$

$C_2 = x_1 \vee x_4$

$C_3 = x_1 \vee \neg x_5$

$C_4 = x_2 \vee x_3$

$C_5 = x_4 \vee x_6$

$C_6 = x_2 \vee \neg x_5$

$C_7 = x_2 \vee x_4$

- Soit S = (0, 1, 1, 1, 0, 1), une affectation qui rend à vrai le système **2-SAT**.

- L'application de l'algorithme de la fonction à sens unique **2-SAT/3-SAT** donne :

- Le système **3-SAT** suivant, qu'on utilisera comme clé publique :

$$\begin{cases} C_1 = \neg x_1 \vee x_2 \vee \neg x_3 \\ C_2 = x_1 \vee x_4 \vee x_5 \\ C_3 = x_1 \vee \neg x_5 \vee \neg x_6 \\ C_4 = x_1 \vee x_2 \vee x_3 \\ C_5 = \neg x_2 \vee x_4 \vee x_6 \\ C_6 = x_2 \vee \neg x_3 \vee \neg x_5 \\ C_7 = x_2 \vee x_4 \vee \neg x_5 \end{cases}$$

- Et le vecteur V = (3, 3, 3, 1, 1, 2, 3), comme clé privée.

- En résumé, on a proposé un système **2-SAT** camouflé sous un problème **3-SAT** qui va être la clé publique, et le vecteur V comme clé privée.

3. Proposition d'un cryptosystème basé sur SAT

3.1. L'algorithme de ce cryptosystème

On se base sur un algorithme du problème **2-SAT** qui est un problème polynomial facile à résoudre, camouflé sous un problème **3-SAT** pour en faire la clé publique, et un vecteur V comme clé privée. Plus en détails, voici ce que cela donne :

- Soit un système **2-SAT** de n variables et m clauses, et soit la solution $S = (s_1, s_2, \ldots, s_n)$ qui rend valide le système **2-SAT**.

En appliquant l'algorithme de la fonction à sens unique **2-SAT/3-SAT**, il en résulte :

- Un système **3-SAT** de n variables et m clauses comme clé publique.
- Un vecteur V de dimension m, où chaque élément doit être compris entre 1 et 3.

Envoie du message :

On veut envoyer le message M = (m_1, m_2,…, m_n)(en binaire).

On calcule MC = (c_1, c_2, …, c_n)(message crypté) de la façon suivante:

- Si $m_i = s_i$ alors $c_i = 1$, sinon $c_i = 0$.

Puis on envoie le message crypté MC.

Réception du message :

Quelqu'un qui intercepte le message crypté MC et connaît le système **3-SAT** utilisé ne peut pas remonter au message initial M, car il ne peut pas connaître la solution S qui est la solution de **3-SAT**.

Pour ce faire :

- On applique l'algorithme inverse de la fonction à sens unique sur le système **3-SAT** et le vecteur V, on obtiendra par la suite le système **2-SAT**, et par la suite la solution S = (s_1, s_2, …, s_n).

- On calcule le message initial M de la façon suivante :

- Si $c_i = s_i$ alors $m_i = 1$, sinon $m_i = 0$.

Et enfin, on obtiendra le message initial M = (m_1, m_2,…, m_n).

3.2. Application numérique :

- **Fabrication des clés publiques et privées :**

- On choisit un système **2-SAT** de 6 variables et 7 clauses.

$$\begin{cases} C_1 = \neg x_1 \vee x_2 \\ C_2 = x_1 \vee x_4 \\ C_3 = x_1 \vee \neg x_5 \\ C_4 = x_2 \vee x_3 \\ C_5 = x_4 \vee x_6 \\ C_6 = x_2 \vee \neg x_5 \\ C_7 = x_2 \vee x_4 \end{cases}$$

- On choisit comme solution de ce problème la solution : S = (0, 1, 1, 1, 0, 1).

- après application de l'algorithme de la fonction unique **2-SAT/3-SAT**, on obtiendra :

➢ Le système **3-SAT** qui va être par la suite clé publique.

$$
\begin{cases}
C_1 = \neg x_1 \vee x_2 \vee \neg x_3 \\
C_2 = x_1 \vee x_4 \vee x_5 \\
C_3 = x_1 \vee \neg x_5 \vee \neg x_6 \\
C_4 = x_1 \vee x_2 \vee x_3 \\
C_5 = \neg x_2 \vee x_4 \vee x_6 \\
C_6 = x_2 \vee \neg x_3 \vee \neg x_5 \\
C_7 = x_2 \vee x_4 \vee \neg x_5
\end{cases}
$$

➢ Le vecteur V de dimension 7 comme clé privée.

V = (3, 3, 3, 1, 1, 2, 3).

- **<u>Envoi du message</u> :**

On veut envoyer le message M en binaire :

M = 1 0 0 1 0 1 0 1 0 1 0 0

$$
\begin{cases}
C_1 = \neg x_1 \vee x_2 \\
C_2 = x_1 \vee x_4 \\
C_3 = x_1 \vee \neg x_5 \\
C_4 = x_2 \vee x_3 \\
C_5 = x_4 \vee x_6 \\
C_6 = x_2 \vee \neg x_5 \\
C_7 = x_2 \vee x_4
\end{cases}
$$

- On calcule le message crypté MC en appliquant la règle d'envoi, on obtiendra :

MC = (0 0 0 1 1 1 1 1 0 1 1 0).

Et on envoie le message crypté MC.

- **<u>Réception du message</u> :**

On calcule **2-SAT** par l'algorithme inverse de la fonction à sens unique **2-SAT/3-SAT**, et on obtiendra le système suivant :

$$\begin{cases} C_1 = \neg x_1 \vee x_2 \\ C_2 = x_1 \vee x_4 \\ C_3 = x_1 \vee \neg x_5 \\ C_4 = x_2 \vee x_3 \\ C_5 = x_4 \vee x_6 \\ C_6 = x_2 \vee \neg x_5 \\ C_7 = x_2 \vee x_4 \end{cases}$$

- On calcule la solution S du système **2-SAT** :

S = (0, 1, 1, 1, 0, 1).

- On applique la règle de réception sur le message MC pour obtenir le message décrypté MD = (1 0 0 1 0 1 0 1 0 1 0 0).

Donc, on obtiendra le message initial M :

M = (1 0 0 1 0 1 0 1 0 1 0 0).

4. <u>Conclusion</u> :

Dans ce chapitre, on a créé une fonction à sens unique **2-SAT/3-SAT**, qui sert à fabriquer une clé publique qui est un système **3-SAT**, et une clé privée qui est un vecteur V, et par la suite on a défini les règles de cryptage et de décryptage du message.

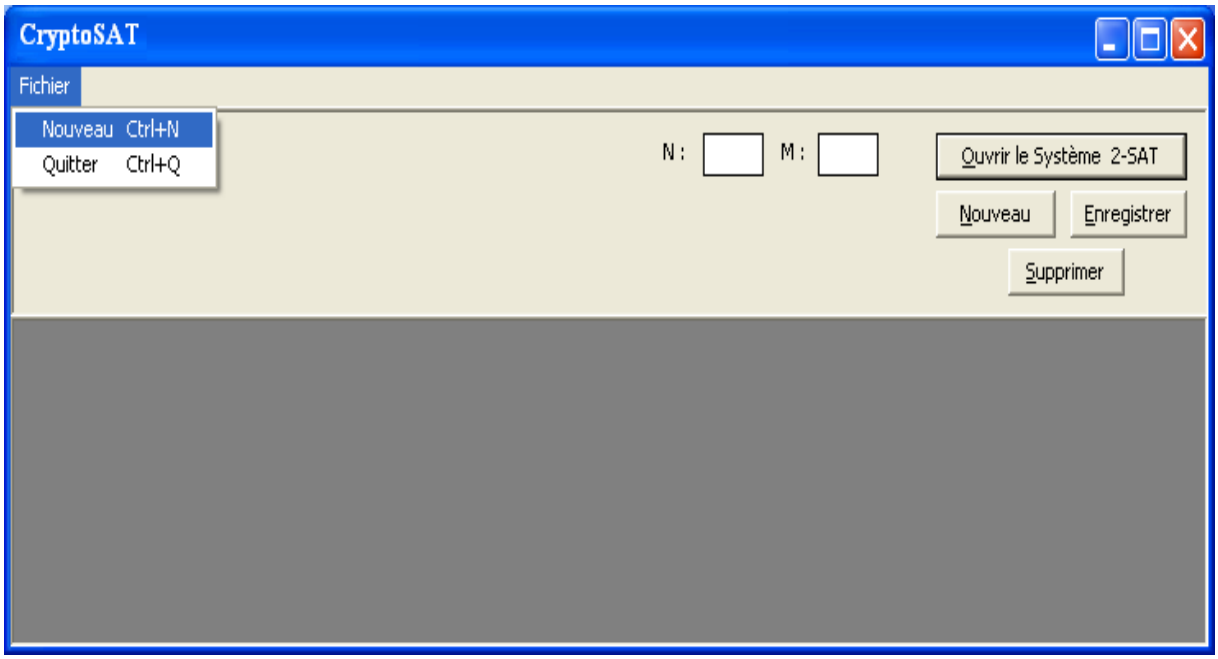
VI. DESCRIPTION DU LOGICIEL

1. INTRODUCTION

Ce chapitre a pour but la présentation de notre logiciel **CryptoSAT**; pour cela nous allons faire une description des différentes opérations que contient ce logiciel.

CryptoSAT est un logiciel concrétisant la plupart des techniques étudiées au long de ce mémoire. Les opérations qu'offre **CryptoSAT** sont regroupées dans un menu rapide est très convivial qui facilite l'interaction entre l'utilisateur et la machine. Nous présentons en détail dans ce qui suit les différents articles de menu du logiciel.

2. La fenêtre mère



La fenêtre mère comporte les commandes suivantes :

- **Fichier :**
  - Nouveau : qui sert à créer un nouveau système 2-SAT avec N variables et M clauses.
  - Quitter : cette commande permet de quitter CryptoSAT.

- **Ouvrir le système 2-SAT :**

Cette commande permet d'ouvrir un système 2-SAT (de N variables et M clauses) prédéfini dans la base de données.

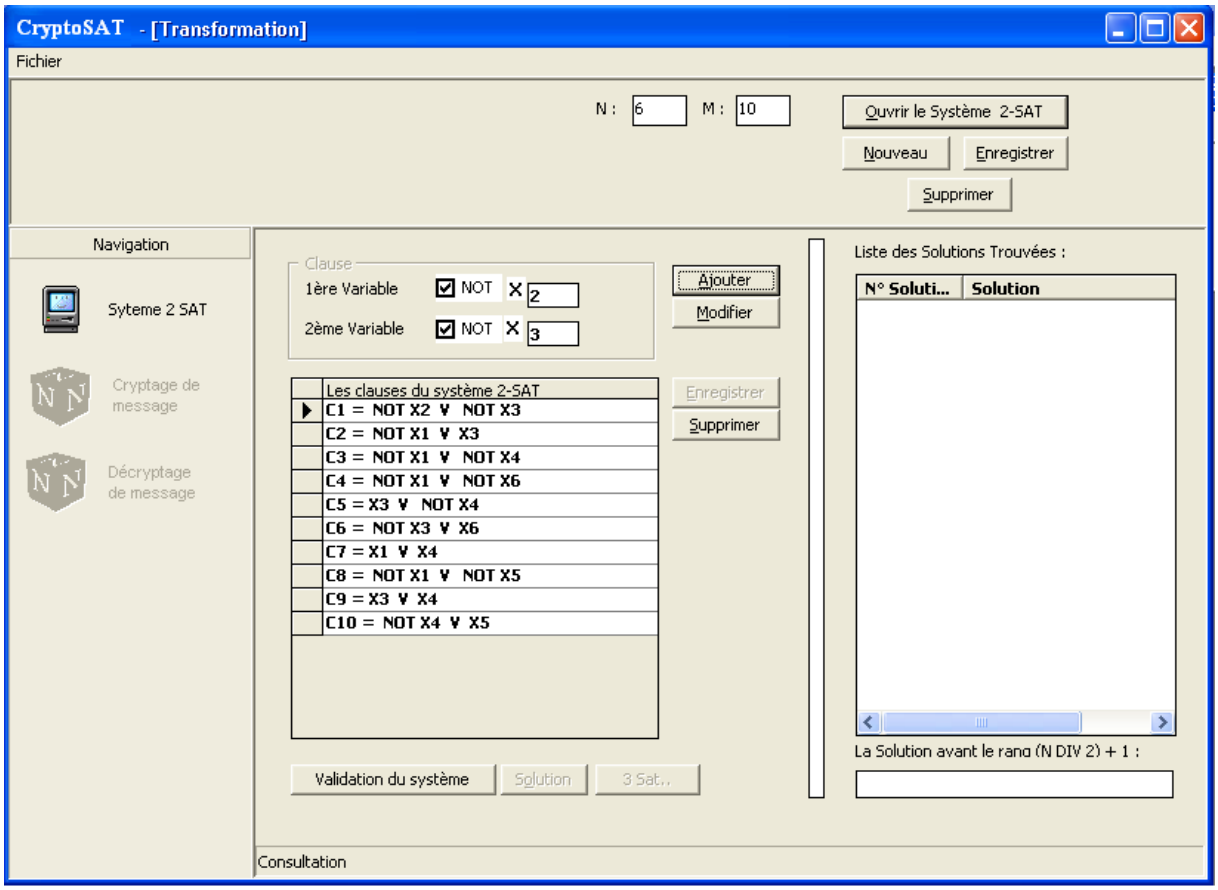
- **Enregistrer :**

L'utilisateur a la possibilité de sauvegarder un nouveau système 2-SAT dans la base de données.

- **Supprimer :**

Cette commande permet à l'utilisateur d'effacer un système 2-SAT de la base de données.

3. La fenêtre Transformation :

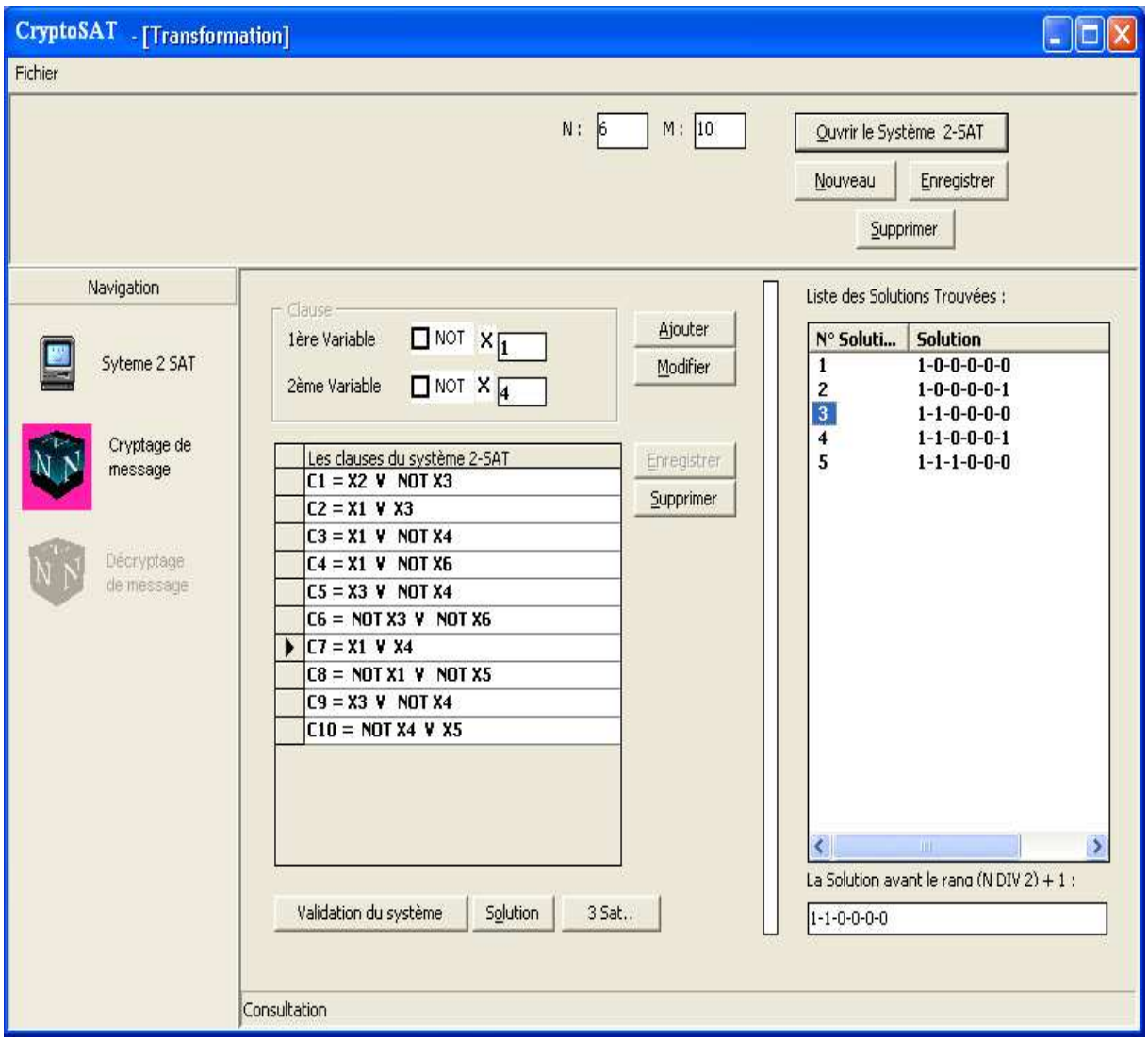


Après l'ouverture d'un système 2-SAT, cette fenêtre permet à l'utilisateur d'ajouter, de modifier, de supprimer ou d'enregistrer une clause.

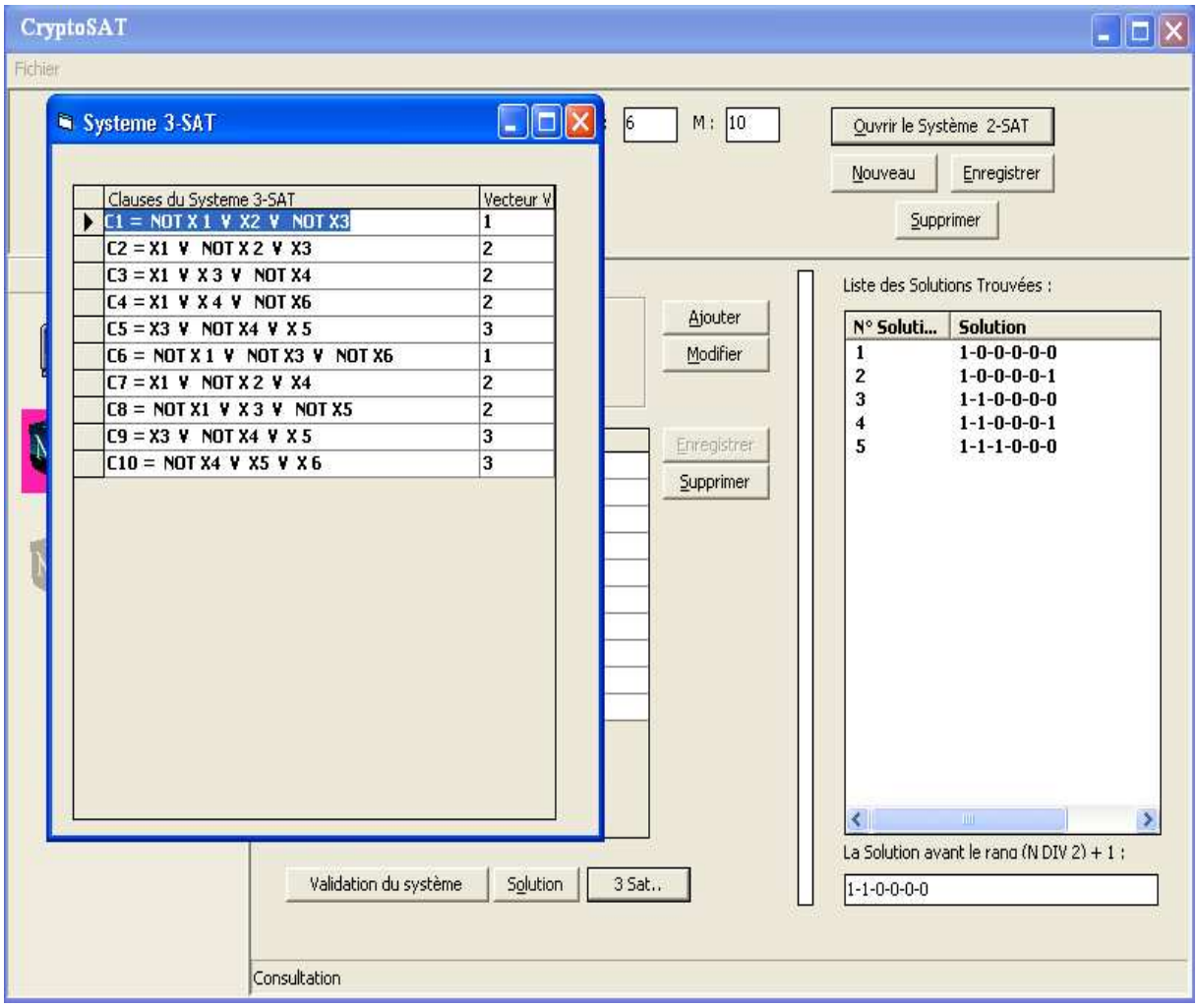
Et après la validation du système 2-SAT choisi, l'utilisateur à la possibilité d'exécuter les commandes suivantes :

- **<u>Solution</u> :**

Cette commande est l'outil de résolution du système 2-SAT ; la figure suivante montre bien l'utilité de cette commande ainsi que le choix de la solution utilisée dans la transformation 2-SAT/3-SAT .



- **3-SAT :**

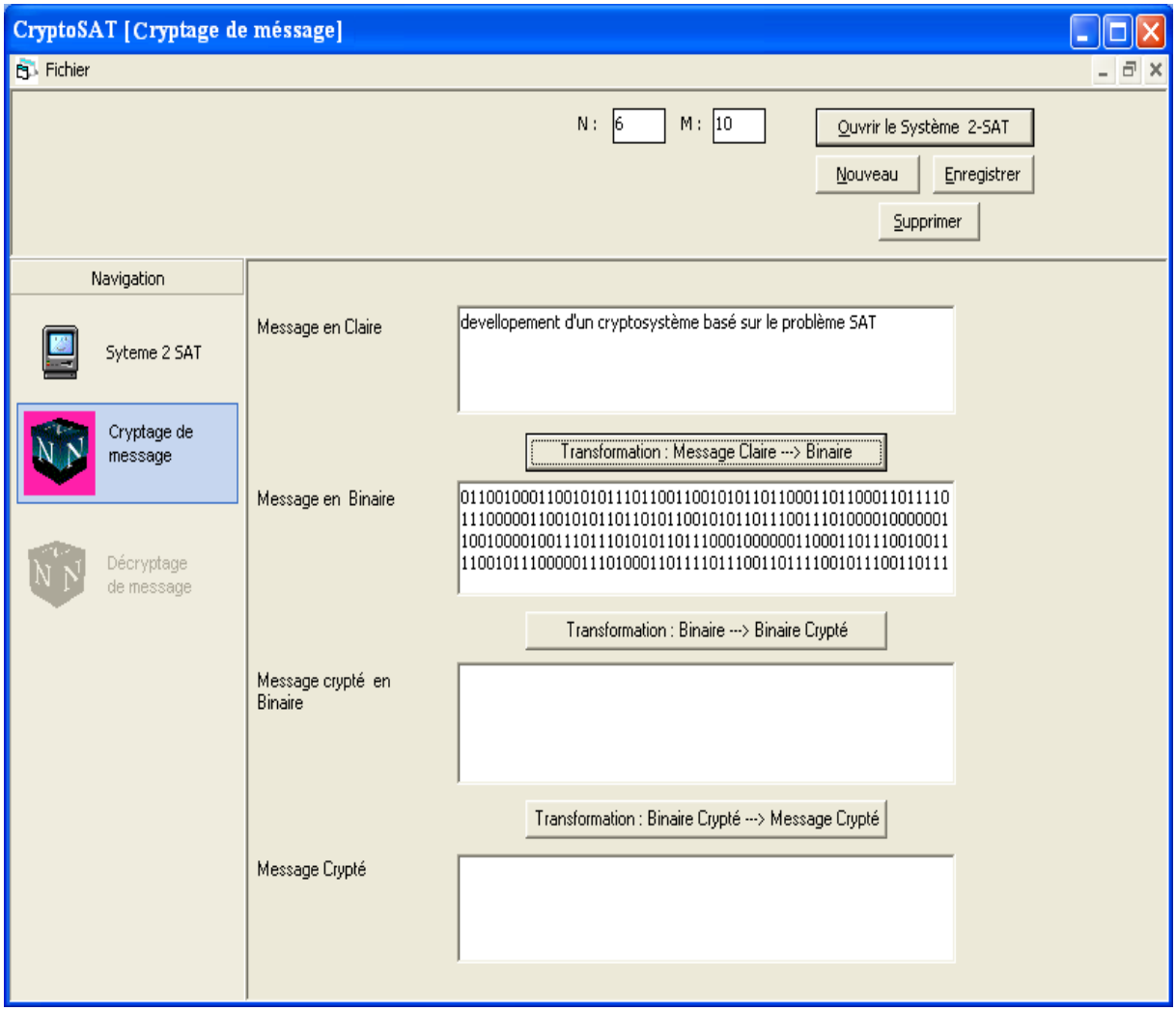


La commande 3-SAT permet de transformer un système 2-SAT en un système 3-SAT en créant un vecteur V qui représente les rangs des variables ajoutées dans le système 3-SAT.

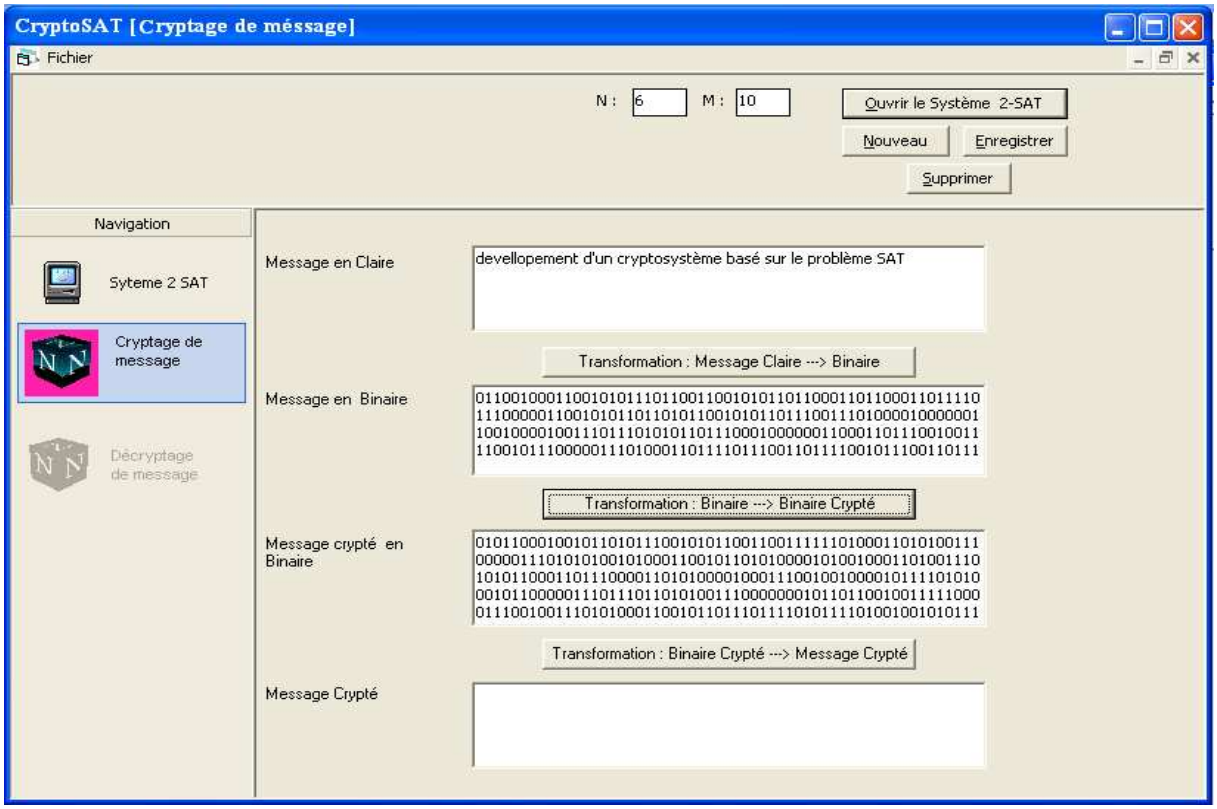
4. La fenêtre Cryptage de message :

A partir de cette fenêtre, l'utilisateur peut obtenir un message crypté selon les étapes suivantes :

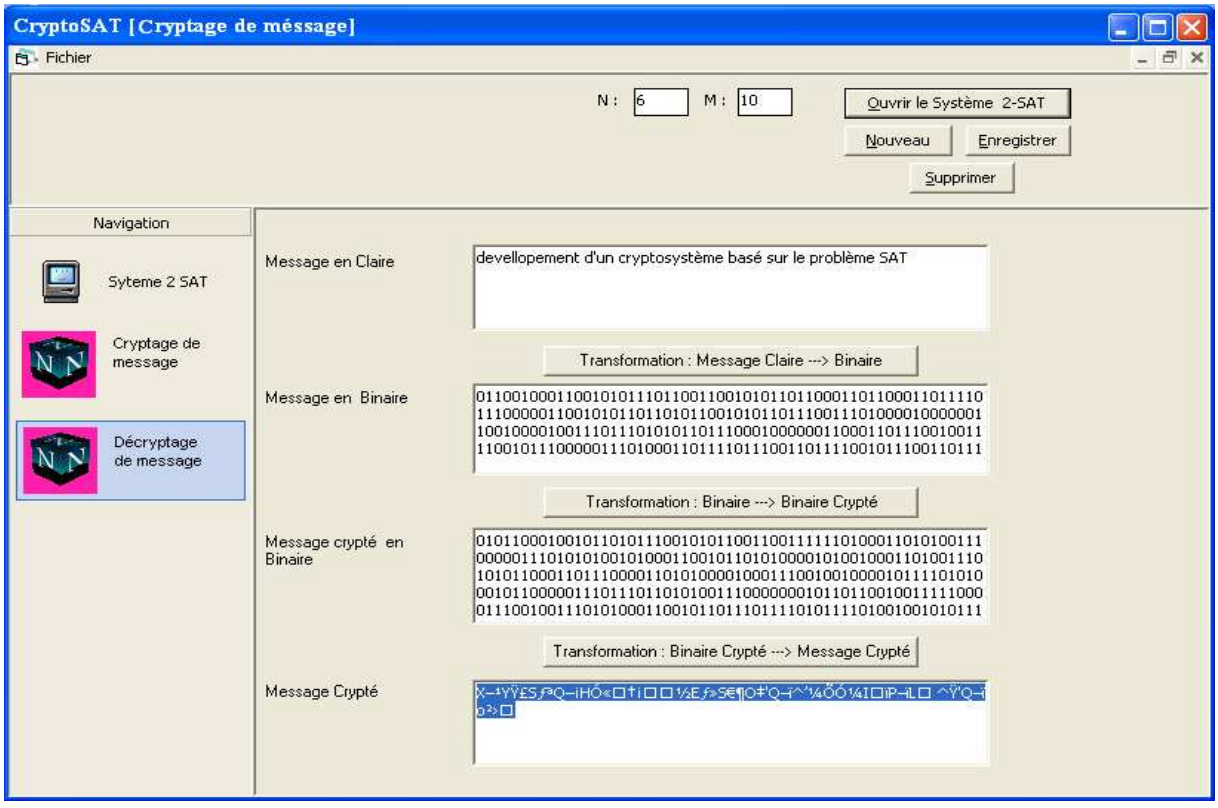
- **Transformation Message clair --> Binaire :**



- **<u>Transformation Binaire --> Binaire crypté</u> :**



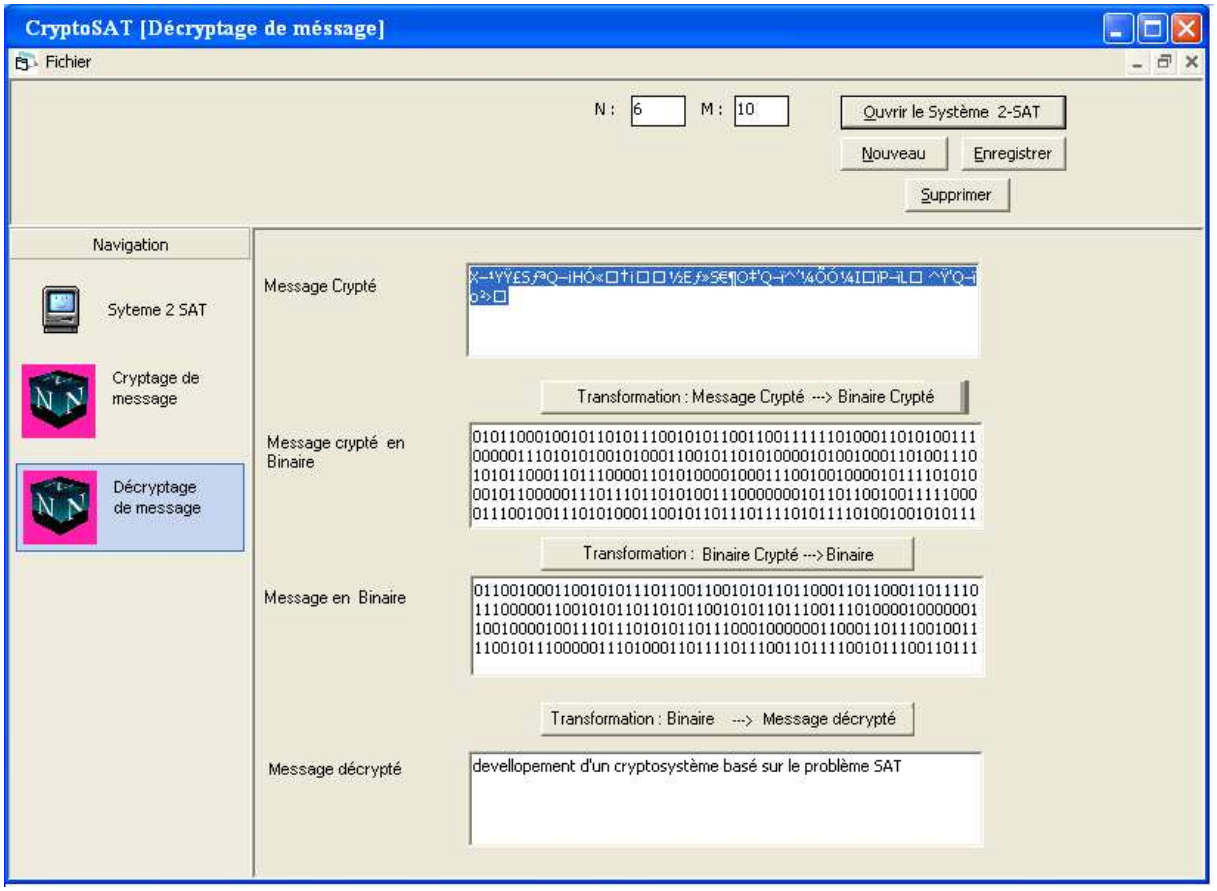
- **<u>Transformation Binaire crypté --> Message crypté</u>:**



5. La fenêtre Décryptage de message :

A partir de cette fenêtre, l'utilisateur peut obtenir un message Décrypté à partir d'un message crypté selon les étapes suivantes :

- Transformation Message crypté --> Binaire crypté.
- Transformation Binaire crypté --> Binaire.
- Transformation Binaire --> Message décrypté.



CONCLUSION

Dans ce mini projet nous avons essayé d'étudier les principales relations existantes entre la cryptologie d'un côté, et la complexité et le problème de satisfiabilité d'un autre coté.

Nous avons commencé par examiner le domaine de la cryptologie : définir la cryptologie et la cryptanalyse, définir formellement les propriétés d'un cryptosystème, et présenter les principaux algorithmes cryptographiques existants. Ensuite, nous avons effectué un survol des principales notions de complexité et ses classes, présenter une définition formelle du problème de satisfiabilité et ses variantes, et résumer ses algorithmes de résolution les plus importants.

Dans la dernière partie de ce travail, nous avons essayé d'illustrer l'utilisation de la logique dans le domaine de la cryptologie, et la représentation des propriétés d'un cryptosystème sous forme de formules booléennes.

Enfin, nous avons présenté un cryptosystème basé sur le problème de satisfiabilité, et mis en évidence les principales équivalences existantes entre un schéma de cryptographie et les éléments du problème de satisfiabilité. Ceci ouvre le champ à divers travaux, nous en citons par exemple :

- essayer de détailler plus le schéma de cryptographie basé sur le problème de satisfiabilité. Et expliciter au mieux les manières de chiffrer et de déchiffrer; ainsi que la construction d'instances de **SAT** à partir d'une solution et la manière de représenter les clés.

Pour ce faire, nous avons créé une fonction à sens unique **2-SAT/3-SAT** qui sert à fabriquer une clé publique qui est un système **3-SAT** et une clé privée qui est un vecteur V, et par la suite, on a défini les règles de cryptage et de décryptage du message.

BIBLIOGRAPHIE

[1] A.TURING « Computing Machinery and Intelligence ».Computers and Thought,E.A, Volume 59, pp.443-460,1950.

[2] H.DREYFUS « What Computers Can't Do: The Limits on Artificial Intelligence ».Herper and Row,New York,1979.

[3] S.A.COOK « The Complexity of Theorem Proving Procedures ».Third Annual ACM Symposium On Theory of Computing, 1971.

[4] M.DAVIS,G.LOGEMANN et D.LOVELOND « A Machine Program For Theorem-Proving ».Communications of The ACM, Volume 5(7) pp.394-397,1962.

[5] M.DAVIS et H.PUTMAN « A Computing Procedure for Quantification Theory». Journal of The ACM, pp.201-215,1960.

[6] G.AUDMARD « Résolution du Problème SAT et Génération de Modèles Finis en Logique du Premier Ordre ». Thèse Doctorat,2001.

[7] C.DUPUIS « Short History of Crypto ».

[8] F.MARIE « Histoire de la Cryptologie ».

[9] D.BLANC « La Cryptographie ».

[10] F.COHEN « A Short History of Cryptography ».

[11] S.IBRI « Parallélisation du Système de Colonie de Fourmis et Application au Problème MAX-SAT ». Mini projet,2001.

[12] K.BENATCHBA , M.KOUDIL « Résolution des Problèmes MAX-SAT et Partitionnement avec les Algorithmes Génétiques et Recherche Dispersée Dans un Environnement Parallèle Simulé sur Réseau ». P.F.E, INI, 2001.

[13] J.P.BARTHELEMY, G.COHEN et A.LOBSTEIN « Complexité Algorithmique». Edition Masson,1992.

[14] M.KHABZAOUI, S.SEBTI « Contribution à la Résolution du Problème MAX-SAT par le Méta-Heuristique 'SCATTER SEARCH' ».PFE,USTHB, 2002.

[15] J.A. ALLIOT et T.SCHIEX. « Intelligence Artificielle et Informatique Théorique ». Cepadus édition,1994.

[16] B.SELMAN,H.LEVESQUE et D.MITCHELL « A New Method for Solving Hard Satisfiabilité Problèmes ». Proceedings AAAI'92, pp.440-446, 1992.

[17] C.H.PAPADIMITRIOU et E.KOUTSOUPIAS « On the Greedy Algorithm for Satisfiability ». Information Processing Letters 43,pp.53-55, 1992.

[18] F.GLOVER « Tabu Search-Part I ».ORSA Journal on computing 1[3],pp.190-206,1989.

[19] P.HANSEN et B. JAUNARD « Algorithms for the Maximum Satisfiability Problem ». 1989.

[20] B.SELMAN, H.KAUTZ et B.COHEN « Local Search Strategies for Satisfiability testing ». 1996.

[21] M.DAVIS et H.PUTNAM « A computing Procedure for Quantification Theory », JACM.(7), pp.201-215, 1960.

[22] M.DAVIS, G.LOGMANN et D.LOVELAND « A Machine Program for Theorem proving Communications ». JACM(5), pp.394-397,1962.

[23] A.VAGNER « Résolution Parallèle du problème SAT, Application à la Cryptographie». DEA,2003.

[24] J.P. DANDRRIEUX « Contributions à l'Analyse de la Complexité de Problèmes de Résolution de Contraintes ». Thèse Doctorat,2002.

[25] S.MIHNEA « Tournées de Véhicules et Satisfiabilité logique ». Philosophie Doctor,1996.