Université de Montréal

Sur des méthodes et algorithmes de factorisation et leur application en cryptologie

par

Alain Slakmon

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de

Maître ès Sciences (M.Sc.)

en informatique

août 2000

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Sur des méthodes et algorithmes de factorisation

et leur application en cryptologie

présenté par:

Alain Slakmon

a été évalué par un jury composé des personnes suivantes:

M. Yoshua Bengio.....président du jury

M. Claude Crépeau.....membre du jury
et directeur de recherche

M. Pierre McKenzie.....membre du jury

Mémoire accepté le:.....30 octobre 2000.....

Sommaire

Notre exposé aborde deux grands thèmes: le problème de la factorisation de nombres entiers, et la cryptologie. Chacun d'eux a une longue et riche histoire, individuellement captivante et parsemée de faits marquants. Leurs routes se sont croisées vers le milieu des années 1970, quand l'invention du concept de cryptographie à clef publique mena au développement de cryptosystèmes dont la sécurité est fondée sur la difficulté, encore actuelle, de factoriser efficacement un nombre produit de deux grands facteurs premiers.

Après un bref aperçu du problème de la factorisation, nous développons, dans un cadre mathématique spécifique, un modèle qui nous mène à démontrer quelques relations, simples mais fondamentales, de la théorie des nombres. Celles-ci permettent d'énoncer un critère de factorisation, basé sur une notion de distance. Ce critère conduit à une fonction, efficace pour factoriser certains nombres considérés difficiles, parfois encore. Il permet aussi d'énoncer et de démontrer un critère d'exclusion de facteurs, conséquence de l'échec d'une application de cette fonction.

Nous poursuivons par une généralisation de certains des concepts présentés, avec en chemin un retour sur l'histoire. Nous analysons, à travers les étapes, des méthodes de factorisation, fondées sur des propriétés démontrées ou observées, élargissant progressivement leur portée. Il en découle des algorithmes spécialisés, capables de factoriser efficacement des classes identifiables de nombres, certaines étant considérées inattaquables.

Nous complétons notre exposé en examinant quelques applications des idées présentées, cette fois dans le contexte de la cryptologie. Du point de vue de la cryptographie, nous présentons, à titre illustratif, un cryptosystème probabiliste à clef privée, utilisant des propriétés démontrées. De la perspective cryptanalytique, nous voyons comment certaines de ces idées pourraient être utilisées pour attaquer un cryptosystème à clef publique, si certaines précautions ne sont pas prises quant au choix des paramètres du système.

Table des matières

Sigles et abréviations

ANSI	American National Standards Institute
Maple	Logiciel mathématique Maple V© de Waterloo Maple inc.
RSA	Cryptosystème de **R**ivest-**S**hamir-**A**dleman Brevet: U.S. patent # 4 405 829 Massachusetts Institute of Technology

À mes parents

Remerciements

Un merci tout spécial au professeur Claude Crépeau, qui a dirigé cette recherche. Il nous a gratifié de ses connaissances, de ses lumières, de sa perspicacité, de sa largesse d'esprit, de son temps précieux, et de sa grande gentillesse. Nous lui sommes redevable pour ces petites questions « *empoisonnées* » qui allaient droit au cœur du problème. Tout comme un phare dans le brouillard, il a su nous guider à bon port.

Merci aussi à mes filles Cybèle et Catherine, à mes frères et sœur et famille, à ma compagne Maude Lavoie, à mon ami Michel Moran de l'École Polytechnique, à mes collègues du département de mathématiques du Collège de Bois de Boulogne, et aux professeurs Richard Duncan et Martin Goldstein du département de mathématiques de l'Université de Montréal. Sans leurs encouragements et leur appui tout au long de cette démarche, ce parcours aurait été autrement difficile.

Merci finalement au département d'informatique et de recherche opérationnelle de l'Université de Montréal, pour la qualité de son environnement.

Introduction

Le problème de la factorisation consiste à trouver, en *temps raisonnable*, les facteurs premiers d'un entier positif quelconque. Il reste, aujourd'hui encore, un problème ouvert, quoiqu'il ait suscité l'intérêt des chercheurs, néophytes ou savants, depuis des siècles, voire des millénaires; l'abondance de littérature sur le sujet en fait foi. Pour un traité général de la question nous en référons à [3]. De nombreux algorithmes de factorisation ont été élaborés à travers l'histoire [4], certains par des mathématiciens renommés tels que Fermat, Gauss et Legendre. Quoique généralement capables de factoriser un nombre quelconque, ces algorithmes ne réussissent toutefois pas à le faire systématiquement en temps efficace, pour tout nombre, peu en importe la taille. Autrement dit, chacun d'eux parvient à factoriser efficacement certaines classes de nombres, mais, pour le complément de ces classes, l'algorithme nécessite généralement un temps de traitement d'ordre exponentiel. De puissants algorithmes, basés sur des résultats avancés de la théorie des nombres ([17], p. 137), ont été développés plus récemment; ils permettent de factoriser un nombre quelconque en temps *sous-exponentiel* ([12], def. 2.60) à la frontière d'un temps de traitement polynomial. Certaines classes de nombres résistent toutefois au calcul efficace de ces algorithmes, en particulier les nombres produits de deux grands facteurs premiers, de taille égale ou à peu près. Ce sont les *durs* (*hard*) ([2], p.362).

Le problème prend une importance particulière dans un contexte de sécurité cryptographique et de communications tous azimuts. Pour un aperçu récent de l'historique du sujet, nous en référons à [16]. Le développement de la cryptographie à clef publique [8], dans les années 1970, et l'avènement de réseaux télématiques planétaires, tel qu'internet, combiné à des besoins exigeants en matière de confidentialité dans les communications, et de sécurité pour les transactions électroniques, ont donné lieu à l'élaboration de cryptosystèmes *calculatoirement sécures*. Pour ces systèmes, comme RSA [14], la sécurité repose essentiellement sur un ou des problèmes considérés difficiles, comme celui de la factorisation d'un nombre *dur*.

La première partie de notre exposé traite de fonctions de type « *split* », capables de calculer efficacement un facteur non trivial d'un nombre, quand celui-ci appartient à une classe ou une autre, selon les cas. Nous identifions en particulier une fonction *split* qui paraît *curieusement* efficace sur certaines classes, comprenant des *durs*. Plus loin, nous analysons en détail les mécanismes de cette fonction.

Dans la section 1.2, nous énonçons et démontrons quelques propriétés (simples mais fondamentales) des nombres, dont nous nous servons tout au long de notre exposé. Elles nous permettent d'identifier avec précision la classe sur laquelle cette fonction agit efficacement, puis, dans la section 1.3, de formuler un critère de factorisation basé sur une notion de distance, auquel nous référons fréquemment.

Nous avons cherché à quantifier, à l'aide d'une mesure, la valeur de l'information acquise et possiblement utilisable advenant un échec de la procédure *split*. Nous le faisons, dans la section 1.4, en termes de nombre de candidats que nous pouvons éliminer comme facteurs possibles d'un nombre donné. On constate, non sans surprise, que celui-ci est considérable: de l'ordre de la racine quatrième du nombre à factoriser. Nous formalisons nos observations en énonçant et en démontrant un critère d'exclusion.

La deuxième partie de l'exposé vise à développer et généraliser certaines des idées présentées. Nous effectuons (section 2.1) un retour sur un algorithme proposé par Fermat, un algorithme généralement considéré comme à valeur historique seule. On constate, qu'en fait, celui-ci incorpore (à sa façon) la fonction *split* étudiée, et plus encore. Nous l'analysons à la lumière des résultats précédents, puis nous donnons une mesure de sa complexité, basée sur une mesure de distance entre les facteurs. Nous effectuons une brève analyse comparative du comportement de cet algorithme versus celui, plus classique, de division, puis nous identifions une fonction indicatrice de son efficacité.

Dans la section 2.2, nous introduisons des notions de *clefs de factorisation* et d'*algorithme de factorisation paramétrisé*, afin de faciliter la présentation des

concepts qui suivent. Nous élargissons la portée de l'algorithme de Fermat, en y introduisant un facteur multiplicatif agissant comme clef (de factorisation). Nous discutons d'un théorème et d'un algorithme de factorisation de R. Sherman Lehman, parus dans un article [11], publié en 1974, et nous soulignons les liens qui s'imposent avec les concepts déjà vus. Nous remarquons aussi la relation que l'on peut établir entre des algorithmes paramétrisés et des classes spécifiques de nombres, ouvrant la porte à des applications dont nous donnons des exemples au dernier chapitre.

Dans la section 2.3, nous poursuivons l'élargissement des concepts, en les portant cette fois dans un contexte d'expressions quadratiques et polynomiales un peu plus générales. Nous revenons d'abord sur l'algorithme de Fermat, pour le situer dans ce contexte. Nous énonçons et démontrons une propriété des nombres, qui, en fait, englobe celles démontrées en première partie. Nous terminons la section en présentant un algorithme, de performance globale plutôt modeste, mais capable de factoriser efficacement de vastes classes de nombres, de toutes tailles, certains appartenant même à la classe de ceux possiblement utilisés dans un contexte cryptographique.

La dernière partie de notre exposé traite d'applications à la cryptologie. De la perspective cryptographique, nous présentons à la section 3.1, à titre d'exemple, un petit cryptosystème probabiliste à clef privée, fondé sur les principes discutés. Nous effectuons une analyse sommaire de sa sécurité, et nous en soulignons les principales caractéristiques.

Nous voyons finalement, dans la section 3.2, comment appliquer certains de ces concepts à la perspective cryptanalytique. Nous situons notre analyse dans le contexte du cryptosystème RSA, actuellement d'utilisation courante. Après un court rappel de ses caractéristiques importantes, nous décrivons des situations hypothétiques où sa sécurité est à risque, si certains critères ne sont pas (ou sont trop) précisés. Nous décrivons quelques attaques, en rapport avec la factorisation, qui mettent en évidence certaines faiblesses du système.

Notre attention se porte ensuite, plus spécifiquement, sur une attaque contre l'exposant secret (un paramètre du système), mais en relation avec la notion de distance mentionnée plus haut. Ce sujet fait l'objet d'un article [7] récent de Benne de Weger. Dans cet article, l'auteur propose un modèle mathématique qui permet d'établir, en fonction de la distance entre les facteurs premiers d'un nombre *dur* (le modulus RSA faisant aussi partie des paramètres du système), une borne en dessous de laquelle l'exposant secret peut être calculé efficacement. Nous discutons de ce résultat puis, notre analyse nous conduit à proposer un modèle un peu plus général, tenant compte d'informations partielles dont on pourrait possiblement disposer. Nous démontrons formellement un résultat inclusif de celui établi par de Weger. Nous terminons par un court scénario, léger d'apparence, mais faisant appel aux concepts discutés. Nous constatons qu'il est tout de même porteur d'une question inquiétante. Un brève conclusion relève les faits saillants de notre exposé.

Chapitre I

Un critère de factorisation basé sur une notion de distance

Ce chapitre a pour but d'établir les fondements sur lesquels repose notre étude. Nous commençons par une brève description des cadres informatique et mathématique, à l'intérieur desquels nous travaillons. Nous identifions les représentations de nombres qui nous intéressent plus particulièrement. Notre attention se tourne alors vers des fonctions de type *split*, capables parfois de factoriser efficacement un nombre donné. Nous en identifions une qui paraît *curieusement* efficace. Notre analyse va porter alors sur certaines propriétés de cette fonction. Nous les explorons à travers un ensemble de relations que nous démontrons formellement. Ceci nous mène à énoncer un critère de factorisation basé sur une notion de distance. Il en découle également un critère d'exclusion.

1.1 Environnement informatique et mathématique

Notation et rappel mathématique

Nous utilisons, dans notre exposé, une notation mathématique conventionnelle. Toutefois, pour éviter toute ambiguïté, nous précisons (en **Annexe A**) le sens de certains des symboles dont nous nous servons. Nous rappelons également quelques résultats utiles, de la théorie des nombres.

Environnement informatique

Le logiciel mathématique Maple V© de Waterloo Maple inc. a servi tout au long de notre étude. Ses multiples et puissantes fonctions arithmétiques ont permis le traitement facile des très grands nombres avec lesquels nous avons travaillé, sans

autre appréhension. Son module de la théorie des nombres (*numtheory*), qui inclut les algorithmes les plus connus et performants de cette branche des mathématiques, nous a permis d'identifier efficacement de grands nombres premiers, de factoriser, d'effectuer des calculs en arithmétique modulaire et de nombreuses autres opérations indispensables à notre analyse. Nous en avons fait grandement usage, sans plus de formalité. Nous identifions, à l'**Annexe B**, les principales fonctions utilisées.

Les procédures et algorithmes

Nous nous sommes également servi de Maple V et de son langage de programmation, pour le développement des procédures que nous présentons. Les fonctions Maple V utilisées sont celles décrites en Annexe B. Nous avons placé les procédures en **Annexe C**, afin de ne pas encombrer le corps du mémoire. Elles ne sont pas commentées ou presque. Les structures utilisées sont courantes, et reflètent les algorithmes associés qui eux, par contre, se trouvent dans le texte et sont généralement commentés.

Des exemples, des exemples

Nous présentons, dans notre exposé, de nombreux exemples numériques. Il nous sont utiles pour diverses raisons : introduire une idée, souligner un point, nuancer une situation, etc.. Nous avons généralement travaillé avec des nombres de grande taille, souvent de plusieurs centaines de chiffres. Nous avons placé tous ces exemples, et bien d'autres encore, en **Annexe D**, ici aussi pour éviter l'encombrement. Toutefois, pour ne pas nuire à la continuité du texte, nous reproduisons parfois, à des endroits appropriés, les premiers chiffres de nombres utilisés. Dans ces cas, nous faisons référence aux exemples associés, mais leur consultation n'est pas essentielle.

Représentations des nombres

Tout nombre peut se représenter de diverses façons et, selon les contextes et les objectifs d'une étude, être vu sous l'une ou l'autre de ces perspectives. Dans le cadre de notre étude nous nous intéressons particulièrement à trois représentations d'un

entier positif quelconque : soit comme un produit de facteurs, soit comme une différence de carrés, soit sous une forme polynomiale plus générale.

Produit de facteurs

Soit n un nombre entier positif *composé* (i.e. produit de facteurs) quelconque. Le théorème fondamental de l'arithmétique nous assure de l'existence de facteurs premiers distincts $p_1, p_2, \ldots, p_k$, et d'exposants entiers positifs $e_1, e_2, \ldots, e_k$ tels que n peut s'écrire sous la forme d'un produit $n = \prod_{i=1}^{k} p_i^{e_i}$, cette représentation étant unique, à l'ordre des facteurs près. La connaissance, de tous ces facteurs premiers et de leur exposant, constitue (par définition) une *factorisation* de n.

Mais nous pouvons également représenter n, d'une ou plusieurs façons, comme produit de deux facteurs $n = pq$, où p et q sont des nombres produits de facteurs premiers de n, ou possiblement premiers. Par exemple: $60 = (2^2)(3)(5)$, mais aussi $60 = (2)(30) = (3)(20) = (4)(15) = (5)(12) = (6)(10)$.

La représentation de n sous la forme pq n'est unique que si n est le produit de deux nombres premiers, comme $15 = (3)(5)$. En fait, il est facile de constater que le nombre de représentations possibles croît exponentiellement, en fonction du nombre de facteurs premiers distincts p_i et des valeurs des exposants e_i. La connaissance de p ou q, pour une représentation quelconque $n = pq$, constitue une *factorisation partielle* de n, sauf si n est le produit de deux nombres premiers, auquel cas la factorisation est complète. Lorsque, pour un n donné, nous cherchons à trouver des nombres $p > 1$ et $q > 1$ tels que $n = pq$, nous considérons que nous cherchons à *séparer* (*split*) n. Notre exposé ne portant que sur la *séparation* de nombres, et cette nuance étant établie, nous utilisons généralement et abusivement, le terme *factorisation* au lieu de *séparation*. Remarquons toutefois, comme il est mentionné dans ([2], p. 362) que si l'on dispose d'un algorithme capable de séparer un nombre, alors un appel récursif à cet algorithme, sur les facteurs déjà trouvés, donne lieu à un algorithme de factorisation.

Différence de carrés

Tous les entiers positifs impairs, ainsi que tous les multiples de quatre (4), peuvent s'écrire sous la forme d'une différence de carrés $n = x^2 - y^2$, où x et y sont des entiers non négatifs tels que $0 \le y < x$. Si n peut s'écrire sous cette forme, la connaissance x ou y permet alors de factoriser n puisque $x^2 - y^2 = (x+y)(x-y)$. On remarque, ici aussi, que cette représentation n'est pas toujours unique (par exemple $60 = 8^2 - 2^2 = 16^2 - 14^2$). De plus, on peut constater qu'une condition nécessaire et suffisante pour l'existence de nombres x et y vérifiant les relations précitées est l'existence de deux entiers positifs p et q, de même parité (pair ou impair) et tels que $n = pq$. En effet, s'il existe deux tels nombres, alors (on suppose ici $p \ge q$) $n = pq = \left(\frac{p+q}{2} + \frac{p-q}{2}\right)\left(\frac{p+q}{2} - \frac{p-q}{2}\right) = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$. Puisque p et q sont de même parité alors $\left(\frac{p+q}{2}\right)$ et $\left(\frac{p-q}{2}\right)$ sont des entiers, disons x et y respectivement, et l'on a ainsi $n = x^2 - y^2$. On note que si p et q sont impairs alors n est impair et si p et q sont pairs alors n sera un multiple de 4. Inversement, si n est impair ou multiple de 4 et peut s'écrire sous la forme $n = x^2 - y^2$ où x et y sont des entiers non négatifs, avec $0 \le y < x$, il suffit de poser $p = x + y$ et $q = x - y$ pour avoir $n = pq$; on remarque que si x et y sont de même parité alors p et q seront tous deux pairs, tandis que si x et y sont de parité contraire alors p et q seront impairs. Ceci établit, dans le cas où n est impair ou multiple de 4, une adéquation entre les deux représentations, à laquelle nous faisons appel fréquemment.

Forme polynomiale

Tout entier peut se représenter, de multiples façons, sous des formes polynomiales $n = \sum_{i=0}^{k} a_i x^i$ où x, la *base*, est un entier positif, et où les a_i sont des entiers quelconques. Par exemple, $60 = 6(10) + 0$, mais nous avons aussi

$60 = 7^2 + 7 + 4 = 13^2 - 9(13) + 8$. On remarquera que certaines expressions donnent lieu à un polynôme factorisable, d'autres pas. Ainsi, $7^2 + 7 + 4$ est une instance de $x^2 + x + 4$ qui n'est pas factorisable dans l'anneau des polynômes à coefficients entiers tandis que $13^2 - 9(13) + 8$, est une instance de $x^2 - 9x + 8$, qui peut lui se factoriser comme $(x-1)(x-8)$. Nous regardons, au chapitre suivant, la factorisation de nombres exprimés comme formes quadratiques, ou sous d'autres formes comme $x^n - y^n$.

Les fonctions split

Un appel récursif à un oracle, pouvant séparer un nombre composé quelconque, résulterait en un algorithme efficace de factorisation. Dans l'attente et la recherche d'un hypothétique oracle *split*, nous examinons et comparons quelques fonctions (procédures simples et efficaces) capables parfois de répondre à la manière d'un oracle. D'un point de vue probabiliste, chacune de ces fonctions peut factoriser efficacement un entier positif composé quelconque, avec une certaine probabilité. Celle-ci est généralement nulle ou faible, mais sous certaines conditions elle peut être élevée ou certaine.

Notre modèle d'une telle fonction est :

split (n) = succès, et retourne [p, q], si un calcul efficace génère $p > 1$ et $q > 1$ vérifiant $n = pq$

split (n) = échec, et retourne [1, n], si le calcul prévu ne permet pas d'obtenir $p > 1$ et $q > 1$ vérifiant $n = pq$

On remarque que si n est composé on obtient *succès* ou *échec*, avec une certaine probabilité qui dépend de n et des spécificités de calcul de la fonction *split*. Si n est premier (ou $n = 1$) on obtient sûrement *échec* mais, en général, on ne peut pas conclure que n est premier, puisque calcul efficace, dans le contexte, sous-entend une limitation *raisonnable* du temps alloué au calcul. On ne cherche toutefois pas à

évaluer les probabilités de succès ou d'échec pour les quelques exemples présentés ci-après, quoique la question de mesure mérite attention . On vise plutôt à explorer les possibilités et limitations de chacune des fonctions, sous diverses conditions. On constate ainsi des différences quant à la *performance* de ces fonctions, certes, mais certaines de ces différences paraissent quelque peu surprenantes. Ceci nous amène à analyser plus en détail une de ces fonctions.

Un exemple trivial

Le schéma général d'une procédure *split* consiste en un calcul suivi d'un test. Si la condition du test est vérifiée, alors *succès* et on retourne $[p, q]$ qui factorise n, sinon *échec* et on retourne $[1, n]$. On définit donc *pair_split* (n) comme suit:

- Calculer $q = \frac{n}{2}$
- Si q est un entier alors succès et retourne $[2, \frac{n}{2}]$,
- Sinon échec et retourne $[1, n]$.

Il est clair, de par sa définition, que *pair_split*(n) va réussir à factoriser n si et seulement si n est un nombre pair.

Ainsi : *pair_split* $(60) = [2, 30]$ tandis que *pair_split* $(15) = [1, 15]$

Si on ne tenait compte que des valeurs *succès* ou *échec* de la procédure, sans aucune autre considération, celle-ci serait assez performante puisque, pour un entier positif aléatoire n, petit ou grand, le taux de succès est de 50%. En considération de critères plus qualitatifs, on pourrait même constater que si le nombre à factoriser est de la forme $n = 2p$, où p est premier, alors *pair_split* va trouver le plus grand facteur premier de n et, en fait, factoriser entièrement n. Les limitations de la procédure sont toutes aussi évidentes.

L'exemple classique

L'algorithme *classique* de factorisation consiste tout simplement à diviser successivement un nombre n donné par des valeurs entières et, si le résultat d'une des divisions est un entier, alors le diviseur est un facteur de n. Cet algorithme, parfois associé au *crible d'Ératosthène* ([17], p.133), a des origines remontant à la Grèce antique ([4], p.274). Il est d'usage de commencer par réduire le nombre à factoriser à la forme $n = 2^k m$ où m est impair (en effectuant de fait des divisions successives par deux jusqu'à l'obtention d'un quotient m impair), puis de chercher une factorisation de m en le divisant successivement par les entiers de la suite des impairs 3, 5, 7, 9, ... jusqu'à l'obtention d'un facteur. Il est clair qu'il serait préférable de n'avoir à effectuer les divisions qu'avec des diviseurs premiers puisque, si l'on détermine qu'un nombre premier n'est pas un facteur de m, alors aucun de ses multiples ne peut l'être; la division par ses multiples est donc inutile. Au cours de l'histoire, plusieurs méthodes ont été proposées pour réduire le nombre de divisions, dont l'utilisation de tables de nombres premiers (cela nécessite du temps de calcul et/ou de l'espace mémoire), ou l'utilisation de cribles pour filtrer, autant que possible, les nombres premiers. Remarquons, tel que mentionné dans ([2], p.362) que si m n'est pas un nombre premier il faut alors que son plus petit facteur premier, disons p, soit tel que $p \leq \sqrt{m}$. Il est donc inutile d'essayer la division par des valeurs p supérieures à $\sqrt{m}$, et, si cette borne est atteinte sans avoir trouvé de facteur de m, on peut, dans ce cas, en conclure que m est premier. Il est à noter que, même si on utilise une table contenant tous les nombres premiers jusqu'à $\sqrt{m}$, le nombre moyen d'itérations de l'algorithme, sur des nombres composés de deux grands facteurs premiers, est de $O\left(\sqrt{m}\right)$ (voir remarque ci-après). Si m est premier, le pire des cas, et que l'on essaie (naïvement) tous les nombres de 2 à $\sqrt{m}$, alors l'algorithme s'exécute en temps $\Omega\left(\sqrt{m}\right)$ ([2], p.362). **Remarque** : ici et par la suite, les temps d'exécution cités ne réflètent que le nombre d'itérations requis par les algorithmes; ils ne tiennent donc pas compte des coûts pour effectuer les opérations élémentaires (divisions et autres). Ceux-ci sont d'ordre polynomial en $\log m$ ([12], table 2.1) et se multiplient à ceux discutés, devenant de plus en plus significatifs à mesure que la taille des nombres augmente.

Il découle de sa formulation que l'algorithme *classique* est, en fait, très efficace quand le nombre à factoriser, disons n, possède un *petit* facteur premier. En effet, si l'on ne tient compte que du nombre d'itérations nécessaire à l'algorithme pour trouver un facteur de n, sans égard aux variations dans les temps d'exécution des divisions, cet algorithme trouve, tout aussi facilement, un même facteur p, que n ait 10 ou 100 ou...plus de chiffres. Cette propriété confère à l'algorithme *classique* une place de premier rang quand il s'agit de factoriser un nombre dont on ne connait pas la composition car, en limitant arbitrairement le nombre d'itérations (de l'algorithme) par une borne B pouvant être atteinte en temps raisonnable, il permet soit de trouver efficacement un facteur p de n, soit d'éliminer comme facteurs possibles toutes les valeurs jusqu'à B. Remarquons toutefois, qu'avec cette approche, si $B < \sqrt{n}$ et que la borne est atteinte sans succès, on ne pourra pas conclure que n est premier, et il faudra avoir recours à d'autres algorithmes pour tenter une factorisation.

Comme contrepartie de son efficacité à trouver les *petits* facteurs premiers d'un nombre n donné, l'algorithme *classique* devient de plus en plus inefficace à mesure que le plus petit des facteurs de n est proche de $\sqrt{n}$, et ainsi devient tout à fait inutile pour factoriser un nombre, ne serait-ce que d'une centaine de chiffres, si ce nombre était composé de deux facteurs premiers d'environ la même taille (les *durs*). Cette incapacité à pouvoir traiter efficacement la factorisation de nombres *durs* relègue l'algorithme *classique* au seul fait historique face aux problèmes contemporains reliés à la sécurité de systèmes cryptographiques, qui eux utilisent (presque) exclusivement des nombres *durs*. Il n'en demeure pas moins utile pour fin de comparaison ou, comme nous l'avons déjà remarqué, dans un contexte plus général.

Une instance remarquable

Considérons (pour l'instant sans justification!) la fonction *split* (n) suivante : Selon le modèle, elle attend en entrée un nombre n (à factoriser), et retourne une paire de nombres, deux facteurs de n ou la paire $[1, n]$. On définit *split* (n) =

- Calculer $x = \lceil \sqrt{n} \rceil$

- Vérifier si $x^2 - n$ est un carré
- Si la condition est vérifiée, retourner $\left[x + \sqrt{x^2 - n},\ x - \sqrt{x^2 - n}\right]$
- Sinon retourner $[1, n]$

La procédure associée *split*, en Maple, se trouve en **Annexe C**. Voyons maintenant quelques applications de cette fonction sur certaines classes de nombres.

Remarque : Dans les exemples qui suivent, et par la suite, les nombres *entiers aléatoires* mentionnés et utilisés sont, en fait, pseudo-aléatoires générés suivant une loi uniforme, à l'aide des fonctions **Rnd** ou **rand** (voir **Annexe B**).

- **Nombres *n* de la forme $n = x^2 - 1$ où *x* est un entier aléatoire** (Pour l'exemple numérique correspondant, voir **Annexe D**, 1.1- Exemple 1).

Pour cet exemple, on choisit d'abord (pseudo) aléatoirement un nombre entier x (ici d'une quarantaine de chiffres), puis on calcule $n = x^2 - 1$, que l'on soumet ensuite à la procédure *split*. On constate ici que *split* retourne systématiquement (pour tous les cas essayés!) la factorisation $n = (x+1)(x-1)$, qui est non triviale puisque elle factorise n en deux grands facteurs. Mais, si la portée de *split* était restreinte à cette classe de nombres, elle serait d'utilité limitée. Voyons alors comment la fonction se comporte quand nous élargissons la classe de nombres.

- **Nombres de la forme $n = x^2 - y^2$ où *x* et *y* sont aléatoires** (Pour les exemples numériques, voir **Annexe D**, 1.1 - Exemples 2a et 2b).

Dans un premier temps, nous voyons comment *split* se comporte quand y est choisi relativement petit par rapport à x. On choisit d'abord aléatoirement un nombre l, puis aléatoirement encore, un nombre x de taille l, puis y de taille $\left\lfloor \frac{l}{3} \right\rfloor$ (voir exemple 2a).

On constate ici aussi que, dans tous les cas (essayés), *split* retourne une factorisation non triviale de n. On peut observer, de manière expérimentale, que *split* va toujours réussir à factoriser n quand la taille de y est inférieure à environ la moitié de celle de

x. À l'opposé, si l'on choisit y plus proche de x on constate que *split* va systématiquement échouer. Ainsi, on peut observer que si l'on choisit y avec environ un quart moins de chiffres que x on obtient effectivement un échec (voir exemple 2b).

Ces observations suggèrent l'existence d'une valeur frontière, pour y en fonction de x, en dessous de laquelle la fonction *split* va toujours réussir à trouver une factorisation de n. L'identification de cette valeur fait l'objet, en partie, de la section suivante. Regardons avant le comportement de *split* quand nous choisissons $n = pq$ comme produit de deux nombres (premiers ou pas) de même parité.

- **Nombres de la forme $n = pq$ avec p et q de même parité** (pour l'exemple numérique, voir Annexe D, 1.1- Exemple 3).

On peut constater, des premiers exemples, que les facteurs trouvés par *split*, quand la procédure réussit, coïncident sur leurs premiers chiffres. On vérifie ici que si l'on choisit d'abord p aléatoirement, puis q aléatoirement mais coïncidant sur environ la moitié des premiers chiffres avec p, et de même parité que p, on obtient effectivement les facteurs p et q en appliquant la fonction *split* sur $n = pq$. Cette propriété s'explique par l'adéquation établie précédemment entre les représentations de n comme différence de carrés ou comme produit de facteurs. Nous précisons plus loin (proposition 1.3.2.) la mesure de proximité nécessaire, entre p et q, pour que la procédure *split* réussisse.

1.2 Sur certaines propriétés des nombres entiers

Dans cette section, nous identifions et démontrons quelques relations, simples mais fondamentales, portant sur les entiers. Mentionnons que nous n'avons pas vu de résultat semblable dans la documentation consultée pour notre recherche. Ces relations vont permettre de préciser la classe de nombres sur laquelle la fonction *split* agit efficacement. Elles nous sont également utiles tout au long de notre exposé.

La perspective d'une différence de carrés

1.2.1 Lemme

Soit x et y des nombres entiers tels que $0 \leq y < x$.

Les relations $y < \sqrt{2x-1}$ et $\left\lceil \sqrt{x^2 - y^2} \right\rceil = x$ sont équivalentes.

Remarque: Cette équivalence permet de préciser la mesure relative selon laquelle, si y est petit par rapport à x, alors le calcul de $\sqrt{x^2 - y^2}$ (arrondi par le haut) va produire x.

Démonstration

Remarquons d'abord que l'on a toujours $\sqrt{x^2 - y^2} \leq x$ puisque $0 \leq y < x$.

On a également $y < \sqrt{2x-1}$

$\Leftrightarrow \quad y^2 < 2x - 1$

$\Leftrightarrow \quad x^2 - 2x + 1 < x^2 - y^2$

$\Leftrightarrow \quad (x-1)^2 < x^2 - y^2$

$\Leftrightarrow \quad (x-1) < \sqrt{x^2 - y^2}$.

Le résultat suit par définition de la fonction $\lceil x \rceil$. *C.Q.F.D.*

La perspective d'un produit de facteurs

1.2.2 Lemme

Soit p et q deux entiers positifs quelconques mais de même parité (pair ou impair) .

Les relations suivantes sont équivalentes:

1. $|p-q| < \sqrt{8\max(p,q)} - 2$

2. $|p-q| < 2\sqrt{p+q-1}$

3. $\left\lceil \sqrt{pq} \right\rceil = \frac{p+q}{2}$

4. $\left\lceil \sqrt{pq} \right\rceil^2 - pq = \left(\frac{p-q}{2}\right)^2$

5. $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$

6. $\left|\sqrt{\frac{p}{q}} - 1\right| < \sqrt{\frac{2}{q}}$ ou $\left|\sqrt{\frac{q}{p}} - 1\right| < \sqrt{\frac{2}{p}}$

Remarques: Ces relations peuvent s'interpréter de diverses façons. Si nous considérons $n = pq$, les deux premières indiquent que la distance entre les facteurs est inférieure à $n^{1/4}$, à une constante près.

La relation $\lceil \sqrt{pq} \rceil = \frac{p+q}{2}$ révèle que la moyenne géométrique des facteurs p et q, arrondie par le haut, coïncide avec leur moyenne arithmétique.

La relation $\lceil \sqrt{pq} \rceil^2 - pq = \left(\frac{p-q}{2}\right)^2$ signale que la différence entre la moyenne géométrique (arrondie puis élevée au carré) et le produit des facteurs, donne lieu à un carré.

La relation $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$ précise une borne supérieure absolue ($\sqrt{2}$) pour la distance entre les racines carrées des facteurs.

La dernière de ces relations nous indique que le rapport entre les facteurs est significativement proche de l'unité.

Démonstration

Montrons d'abord que **1.** $\Leftrightarrow$ **2.**. Il y a deux cas à traiter:

Cas: $p > q$

Puisque $p > q$ alors $|p-q| = p-q$ et $\max(p,q) = p$

Nous avons alors $p - q < \sqrt{8p} - 2$ par hypothèse,

$\Leftrightarrow \quad p - q + 2 < \sqrt{8p}$

$\Leftrightarrow \quad (p-q+2)^2 < 8p$

$\Leftrightarrow \quad (p-q)^2 + 4(p-q) + 4 < 8p$

$\Leftrightarrow \quad (p-q)^2 < 8p - 4(p-q) - 4$

$\Leftrightarrow \quad (p-q)^2 < 4(p+q-1)$

$\Leftrightarrow \quad |p-q| < 2\sqrt{p+q-1}$

Cas : $p \leq q$

La preuve est similaire, par symétrie, en utilisant $q-p$ en lieu de $p-q$, ce qui établit l'équivalence entre 1. et 2.

Montrons maintenant que **2.** $\Leftrightarrow$ **3.**. Il y a ici aussi deux cas à traiter:

Cas: $p > q$

Nous avons déjà remarqué que $pq = \left(\frac{p+q}{2} + \frac{p-q}{2}\right)\left(\frac{p+q}{2} - \frac{p-q}{2}\right)$

d'où $pq = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$.

Posons $x = \frac{p+q}{2}$ et $y = \frac{p-q}{2}$. On a alors $pq = x^2 - y^2$.

Puisque $p > q$ nous avons $|p-q| = p-q$.

Alors $p-q < 2\sqrt{p+q-1}$ par hypothèse,

$\Leftrightarrow \quad \frac{p-q}{2} < \sqrt{2\frac{p+q}{2} - 1}$

$\Leftrightarrow \quad y < \sqrt{2x-1}$ en effectuant la substitution

$\Leftrightarrow \quad \left\lceil \sqrt{x^2 - y^2} \right\rceil = x$ par le lemme 1.2.1

$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil = \frac{p+q}{2}$ en substituant à nouveau

Cas : $p \leq q$

Ici aussi, la preuve est similaire, par symétrie, en utilisant $q - p$ en lieu de $p - q$.

Montrons maintenant que **3.** $\Leftrightarrow$ **4.**.

Ici, par hypothèse nous avons $\lceil \sqrt{pq} \rceil = \frac{p+q}{2}$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 = \left(\frac{p+q}{2} \right)^2$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \left(\frac{p+q}{2} \right)^2 - pq$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \frac{(p+q)^2 - 4pq}{4}$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \frac{p^2 + 2pq + q^2 - 4pq}{4}$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \frac{p^2 - 2pq + q^2}{4}$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \frac{(p-q)^2}{4}$$

$$\Leftrightarrow \quad \lceil \sqrt{pq} \rceil^2 - pq = \left(\frac{p-q}{2} \right)^2$$

On montre maintenant que **5.** $\Leftrightarrow$ **3.**.

On remarque d'abord que l'on a toujours $0 \leq \left(\sqrt{p} - \sqrt{q}\right)^2$

$$\Leftrightarrow \quad 0 \leq p + q - 2\sqrt{pq}$$

$$\Leftrightarrow \quad \sqrt{pq} \leq \frac{p+q}{2}$$

Ici, par hypothèse $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$

$$\Leftrightarrow \quad \left(\sqrt{p} - \sqrt{q}\right)^2 < 2$$

$$\Leftrightarrow \quad p + q - 2\sqrt{pq} < 2$$

$$\Leftrightarrow \quad \frac{p+q}{2} < \sqrt{pq} + 1$$

$$\Leftrightarrow \quad \frac{p+q}{2} = \left\lceil \sqrt{pq} \right\rceil$$

L'équivalence **5.** $\Leftrightarrow$ **6.** est triviale à établir; il suffit de diviser chaque membre de la relation 5. par $\sqrt{q}$ ou $\sqrt{p}$ respectivement.

C.Q.F.D.

Équivalence des représentations

1.2.3 Lemme

Soit n un nombre entier positif quelconque. Les énoncés suivants sont équivalents:

1. Il existe deux entiers x et y, tels que $0 \leq y < x$, $n = x^2 - y^2$, et $y < \sqrt{2x-1}$

2. Il existe deux entiers positifs de même parité, p et q, tels que $n = pq$ et tels que $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$

Démonstration

Le résultat suit de l'adéquation entre les représentations démontrée précédemment, et des lemmes 1.2.1 et 1.2.2 .

C.Q.F.D.

1.3 Un critère de factorisation

Les relations établies dans la section précédente nous permettent maintenant d'énoncer et de démontrer deux critères de factorisation. Le premier est pour une différence de carrés; le second est pour un produit de facteurs, en fonction de la distance absolue entre leur racine carrée.

Un critère de factorisation pour une différence de carrés

1.3.1 Proposition

Soit n un entier positif quelconque.

Soit $X = \lceil \sqrt{n} \rceil$, et soit $Y = \sqrt{X^2 - n}$

S'il existe deux entiers x et y, tels que $0 \le y < x$, $n = x^2 - y^2$, et $y < \sqrt{2x-1}$

alors $X + Y$ et $X - Y$ sont des facteurs de n.

Démonstration

S'il existe deux tels entiers alors les conditions du théorème sont celles du lemme 1.2.1,

donc $X = \lceil \sqrt{n} \rceil = x$.

Puisque $n = x^2 - y^2$ alors $y = \sqrt{x^2 - n}$, d'où $Y = y$.

Comme $n = x^2 - y^2 = (x+y)(x-y)$

alors $X + Y$ et $X - Y$ sont des facteurs de n.

C.Q.F.D.

Un critère de factorisation basé sur une notion de distance

1.3.2 Proposition

Soit n un entier positif quelconque. S'il existe deux entiers positifs de même parité, p et q, tels que $n = pq$ et tels que $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$ alors $\max(p,q) = x + \sqrt{x^2 - n}$ et $\min(p,q) = x - \sqrt{x^2 - n}$ où $x = \left\lceil \sqrt{n} \right\rceil$.

Démonstration

Soit n un entier positif quelconque et supposons qu'il existe p et q, deux nombres répondant aux critères de l'énoncé, c'est à dire $n = pq$, où p et q sont de même parité et $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$. Posons $x = \left\lceil \sqrt{n} \right\rceil$. Alors, par le lemme 1.2.2, on aura aussi $x = \dfrac{p+q}{2}$. En substituant $q = \dfrac{n}{p}$ on obtient $x = \dfrac{p + \dfrac{n}{p}}{2}$.

Après quelques transpositions on obtient $p^2 - 2px + n = 0$ une équation du second degré en p, ayant comme solutions $\max(p,q) = x + \sqrt{x^2 - n}$ et $\min(p,q) = x - \sqrt{x^2 - n}$.

C.Q.F.D.

1.4 Un critère d'exclusion de facteurs

Les relations démontrées dans les sections précédentes permettent d'apprécier la portée de la procédure *split* définie en 1.1. Les critères 1.3.1 et 1.3.2 précisent la classe des nombres qui sont factorisables efficacement par l'application de la procédure. On constate que le succès de la procédure dépend de la distance *relative* entre les facteurs, ou, vu selon le critère 1.3.2., de la distance absolue entre les racines carrées des facteurs; il est donc indépendant de la taille des nombres. L'algorithme est capable, en quelques opérations élémentaires, de factoriser une infinité de nombres composés, soit tous ceux pour lesquels il existe deux facteurs p et q de même parité et tels que $\left|\sqrt{p}-\sqrt{q}\right|<\sqrt{2}$, et retourne ainsi deux facteurs non triviaux.

Ces considérations soulèvent une autre question: Quelle information (sur les facteurs) peut-on tirer d'un échec de la procédure ? Il est simple de constater que si *split* échoue on peut alors éliminer, comme facteurs possibles d'un nombre donné, toutes les *paires* de nombres (p,q) tels que $\left|\sqrt{p}-\sqrt{q}\right|<\sqrt{2}$. Il est moins évident de préciser quels facteurs *individuels* peuvent être éliminés. Nous en donnons la réponse dans l'énoncé suivant:

1.4.1. Proposition

Soit n un entier positif impair ou multiple de 4. Soit $x=\lceil\sqrt{n}\rceil$. Si x^2-n n'est pas un carré alors n n'a aucun facteur dans l'intervalle d'entiers $\left[(x+1)\pm\left\lfloor\sqrt{2x-1}\right\rfloor\right]$.

Démonstration

Si n est composé, alors le théorème fondamental de l'arithmétique, et le fait que n est impair ou multiple de 4, nous assurent de l'existence d'une ou de plusieurs paires de nombres p et q, chacun composé de facteur(s) premier(s) de n, tous deux de même parité, et tels que $n = pq$. Si n est premier, une seule paire est possible, soit 1 et n.

Parmi toutes les paires possibles, pour n donné, appelons p et q celle dont la distance $\left|\sqrt{p}-\sqrt{q}\right|$ est minimum. Il est établi que, s'ils sont distincts, p et q doivent être situés de part et d'autre de $\sqrt{n}$. Supposons de plus, sans perte de généralité, que $p \le q$. Il est clair aussi que $x^2 \ge n$ et, si $x^2 - n$ n'est pas un carré, il faut alors que $\left|\sqrt{p}-\sqrt{q}\right| \ge \sqrt{2}$ ou, par équivalence, que $q - p \ge 2\sqrt{p+q-1}$.

Le fait que $x^2 - n$ n'est pas un carré élimine d'emblée la possibilité que n en soit un; si c'était le cas on aurait $x = \left\lceil \sqrt{n} \right\rceil = \sqrt{n}$ et par suite $x^2 - n = 0$, un carré, d'où contradiction. Il en découle également que $\frac{p+q}{2} > x$ ou $\frac{p+q}{2} \ge (x+1)$ car on a toujours $\frac{p+q}{2} \ge \left\lceil \sqrt{pq} \right\rceil = x$ et, puisque $x^2 - n$ n'est pas un carré, alors $\frac{p+q}{2} \ne x$. Remarquons aussi que pour tout $a > 1$, si $p = (x+1) - a$, il faut alors que $q \ge (x+1) + a$. En effet, si l'on avait $q < (x+1) + a$ on aurait alors $p + q < 2(x+1)$ et par conséquent $\frac{p+q}{2} < (x+1)$, ce qui est impossible du fait que $x^2 - n$ n'est pas un carré.

Montrons d'abord que l'on ne peut avoir p et q simultanément dans l'intervalle $\left[(x+1) \pm \left\lfloor \sqrt{2x-1} \right\rfloor\right]$. En effet, si p et q sont tous les deux dans cet intervalle alors $q - p \le 2\left\lfloor \sqrt{2x-1} \right\rfloor < 2\sqrt{p+q-1}$ puisque l'on a toujours $p + q \ge 2x$ et que $p + q \ne 2x$, et cela conduit à une contradiction, car il faut avoir $q - p \ge 2\sqrt{p+q-1}$.

Montrons maintenant que ni p ni q ne peuvent se trouver individuellement dans l'intervalle spécifié.

Supposons que p est tel que $(x+1) - \left\lfloor \sqrt{2x-1} \right\rfloor \le p < x$.

Il faut alors que $q \geq (x+1) + \left\lfloor \sqrt{2x-1} \right\rfloor$, d'où

$$n = pq \geq \left((x+1) - \left\lfloor \sqrt{2x-1} \right\rfloor\right)\left((x+1) + \left\lfloor \sqrt{2x-1} \right\rfloor\right)$$

$$= (x+1)^2 - \left\lfloor \sqrt{2x-1} \right\rfloor^2 \geq (x+1)^2 - \left(\sqrt{2x-1}\right)^2 = x^2 + 2 > n.$$

Cela donne ici aussi une contradiction. On ne peut donc avoir p seul dans l'intervalle.

Finalement, on remarque que l'on ne peut avoir q tel que $x \leq q \leq (x+1) + \left\lfloor \sqrt{2x-1} \right\rfloor$ et $p < (x+1) - \left\lfloor \sqrt{2x-1} \right\rfloor$ car si $p < (x+1) - \left\lfloor \sqrt{2x-1} \right\rfloor$ il faudrait alors que $q > (x+1) + \left\lfloor \sqrt{2x-1} \right\rfloor$, menant une fois encore à une contradiction.

C.Q.F.D.

La proposition 1.4.1 précise, en cas d'échec de la procédure *split*, l'intervalle d'exclusion des facteurs possibles d'un nombre n donné et répondant aux critères énoncés. Cet intervalle est vaste, de l'ordre de $\sqrt[4]{n}$ autour de $\sqrt{n}$, puisque il est de rayon $\left\lfloor \sqrt{2x-1} \right\rfloor \cong \sqrt{2\sqrt{n}-1} \cong \sqrt{2}\sqrt{\sqrt{n}} = \sqrt{2}\sqrt[4]{n}$. Ainsi une application de la procédure *split* va, soit permettre de trouver deux facteurs non triviaux d'un nombre donné, soit permettre d'exclure comme facteurs possibles tout nombre à l'intérieur de l'intervalle discuté. Pour un nombre *dur* de 200 chiffres, cette exclusion revient à pouvoir éliminer, en quelques opérations algébriques, une fraction (certes petite) de l'ensemble des facteurs possibles, mais en nombre absolu plus de 10^{50} candidats! Au chapitre suivant, nous voyons comment en élargir la portée.

Chapitre II

Élargissement des concepts

Dans ce chapitre, nous commençons par voir comment élargir la portée de la fonction *split* définie en 1.1. Notre analyse débute par un bref retour sur l'histoire, sur une méthode de Fermat. Nous reconnaissons, dans cette *vieille* méthode, un algorithme qui inclut la fonction discutée, et plus encore, il la prolonge. Nous en faisons une analyse détaillée, et les observations qui en découlent sont utiles à l'ensemble de notre étude. La section 2.2 porte sur une généralisation de l'algorithme en question, proposée par R. Sherman Lehman dans les années 1970. Elle implique l'introduction d'un facteur multiplicatif, pour faciliter la factorisation. Nous l'abordons par le biais de clefs de factorisation et de paramétrisation, qui permettent, par la suite, d'établir des associations entre des classes spécifiques de nombres, et des méthodes capables de factoriser efficacement les nombres qui en font partie.

2.1 À la recherche d'une moyenne : algorithme de Fermat

Sur un fond d'histoire

En 1643, Fermat répondait à un défi lancé par le Père Mersenne. Celui-ci lui demandait de factoriser un certain nombre constitué d'une dizaine de chiffres. La réponse de Fermat, qui bien sûr en donnait les facteurs, constituait en fait un algorithme de factorisation. Un extrait de la lettre originale de Fermat décrivant les détails de cet algorithme se retrouve dans ([4], p.300). Essentiellement, les étapes peuvent se résumer comme suit :

- En entrée : un nombre n à factoriser

- Calculer $x = \lfloor \sqrt{n} \rfloor$, la racine carrée de n (arrondie par le bas):
- Calculer $r = n - x^2$
- Calculer $z = 2x + 1 - r$
- Vérifier si z est un carré.
- Si la condition est vérifiée, la factorisation s'obtient par $n = (x + 1 + \sqrt{z})(x + 1 - \sqrt{z})$. On retourne ces valeurs et on termine
- Sinon, on incrémente x puis on retourne à la deuxième étape.

Remarque : Tel que défini, l'algorithme va itérer jusqu'à obtention des facteurs. Si le nombre n est premier, il finit par retourner la factorisation triviale $n = 1 \times n$.

Fermat utilisait les derniers chiffres d'un nombre pour déterminer si celui-ci était possiblement un carré. On constate aisément que cet algorithme cherche à trouver x et y tels que $(x + y)(x - y) = n$ et qu'il procède en cherchant x itérativement à partir de $x_0 = \lfloor \sqrt{n} \rfloor$, incrémentant sa valeur à chaque itération. Le nombre soumis à Fermat par Mersenne, soit n = 2027651281, nécessita douze itérations pour arriver aux facteurs 44021 et 46061, deux nombres premiers. Certains ont conclu que Fermat avait «[...] bien choisi son exemple » ([4], p.301). Il est facile de constater que, pour cet algorithme, le nombre d'itérations requises est de $x - \lfloor \sqrt{n} \rfloor$, puisque que l'on cherche x à partir de $\lfloor \sqrt{n} \rfloor$. On remarque aussi que si n est premier, le pire des cas, alors l'algorithme de Fermat nécessite $\frac{n+1}{2} - \lfloor \sqrt{n} \rfloor$ itérations pour trouver x car, en fait, la valeur cherchée est $x = \frac{p+q}{2}$, et dans la représentation $n = x^2 - y^2 = (x + y)(x - y) = pq$, on n'a qu'une seule possibilité, soit $n = 1 \times n$ quand n est premier. C'est donc un algorithme de complexité exponentielle (en log n) et l'on peut vérifier, qu'en moyenne, il est $O(\sqrt{n})$, une performance générale comparable à celle de l'algorithme classique, moins bonne que celui-ci dans le pire des cas, et, par rapport aux algorithmes développés plus récemment, une

performance générale plutôt modeste. Knuth ([10], 4.5.4, Algorithm C, p 342) en propose une version légèrement modifiée qui remplace la vérification sur z (carré?) par un test sur r (= 0?) mais le tout nécessite $x + y - \lfloor \sqrt{n} \rfloor$ itérations. Il décrit ensuite comment l'améliorer en y introduisant la notion de crible, et poursuit en donnant un algorithme ([10], 4.5.4, Algorithm D) plus complexe lui, basé sur des calculs modulaires servant à alimenter des cribles. On remarque toutefois que cet algorithme, selon l'analyse même de l'auteur, nécessite $x - \lfloor \sqrt{n} \rfloor$ itérations.

C'est probablement le fait que l'exécution de l'algorithme de Fermat est généralement très longue qui lui a valu de tomber en désuétude au profit d'algorithmes globalement plus performants. On reconnaît ([10], p.342) toutefois qu'il est plus approprié que l'algorithme classique pour chercher les grands facteurs d'un nombre et, quoique de valeur historique, il a servi de modèle au développement de plusieurs générations d'algorithmes, certains parmi les plus performants actuellement (par exemple le crible quadratique ou le crible algébrique, mentionnés dans ([17], p.137)). Un fait toutefois distingue ces algorithmes de celui de Fermat: alors que Fermat cherchait x et y tels que $n = x^2 - y^2$, ces algorithmes, généralement non-déterministes, factorisent n en cherchant x et y tels que $x^2 \equiv y^2 \bmod n$. En effet, si l'on réussit à trouver deux entiers x et y vérifiant cette relation, et que $x \neq y \bmod n$, alors $x - y$ et n ont un facteur commun que l'on peut calculer efficacement à l'aide de l'algorithme d'Euclide ([12], fact 3.18). La recherche des valeurs x et y s'effectue généralement aléatoirement, selon des stratégies déterminées de manière heuristique. Ainsi, l'algorithme de Dixon [9], commence par la détermination (arbitraire) d'une base B composée de *petits* nombres premiers, puis sélectionne au hasard des valeurs x et calcule les résidus quadratiques $r = x^2 \bmod n$. On tente ensuite de factoriser chaque résidu et, si l'on parvient à déterminer que r se factorise entièrement avec des facteurs appartenant à la base B, on garde alors dans une table la valeur r et sa factorisation. On essaie ensuite de trouver une combinaison de r_i dans la table, qui, multipliés ensemble, donnent un carré. Cet algorithme, qui peut être vu comme une version (simplifiée) du crible quadratique, s'exécute en temps sous-exponentiel. La

performance de ces algorithmes devant des nombres *durs* est largement illustrée et commentée dans la littérature. Ils réussissent à repousser de manière significative la frontière qui nous sépare de la possibilité de factoriser un nombre *dur* en temps raisonnable. Ainsi, on mentionne dans ([2], p.363) la factorisation d'un nombre *dur* de 129 chiffres (en décimal). Plus récemment (août 1999) on a réussi à factoriser un nombre dur de 155 chiffres [19], servant de *test* pour la sécurité du cryptosystème RSA. Ce nombre (RSA-155), a été factorisé à l'aide du *crible algébrique*. Mais, mentionnons aussi que, selon la référence, cet exploit a nécessité une attaque concertée impliquant une équipe multinationale (six pays) de chercheurs scientifiques, disposant de quelques 300 stations de travail performantes (SGI ou SUN), aidées d'un super-ordinateur (Cray C916), et que la tâche a requis plus de 35 années-cpu de calculs.

La nette supériorité de performance des algorithmes sus-mentionnés, comparativement à celui de Fermat, est clairement établie. Il peut alors paraître naturel de ne pas inclure ce dernier dans l'élaboration d'un algorithme de factorisation de portée très générale. Ainsi, une stratégie intégrée de factorisation pourrait consister à exécuter, en premier, l'algorithme classique jusqu'à une certaine borne préétablie, afin de chercher d'abord les petits facteurs premiers du nombre, puis d'exécuter l'algorithme de Dixon sur le facteur résultant de la division du nombre à factoriser par le produit des facteurs déjà trouvés. Remarquons toutefois ici que l'algorithme de Dixon, et ses successeurs, de par leur nature probabiliste et les constructions utilisées, et, bien que fondé sur des concepts généralisant ceux de l'algorithme de Fermat, n'englobent pas strictement celui-ci. En effet, l'algorithme de Fermat est déterministe et cherche une valeur x par incrémentation à partir de $\lfloor \sqrt{n} \rfloor$, tandis que ces algorithmes cherchent x (et y) aléatoirement. Ce détail, à première vue insignifiant, s'avère être d'une certaine importance, comme nous le voyons maintenant.

La fonction split se prolonge

Rendons à Fermat ce qui lui revient : Ô Mage! Ô Maître! Son algorithme est capable de factoriser quasi-instantanément une infinité de nombres *durs*, sans limite de taille, s'ils appartiennent à certaines classes, et ce aussi facilement que l'algorithme classique va réussir à trouver les petits facteurs d'un nombre composé, peu en importe la taille. En fait, à la lumière des considérations précédentes, nous constatons qu' il incorpore la fonction *split* analysée au Chapitre I et, plus encore, il la prolonge. Il est en effet facile de voir que l'algorithme de Fermat consiste à chercher itérativement, à partir de la moyenne géométrique $\sqrt{pq}$, la moyenne arithmétique $\frac{p+q}{2}$ et que, d'entrée, il y a, en fait, une application de la fonction *split*. L'algorithme va donc systématiquement trouver immédiatement la factorisation de tout nombre pouvant s'exprimer comme un produit de deux facteurs p et q, de même parité, premiers ou pas, si $\left|\sqrt{p}-\sqrt{q}\right| < \sqrt{2}$. Cette condition, quoique relativement restrictive, se traduit tout de même par une vaste classe de nombres. Un échec de la fonction *split* pour un n donné, permet aussi, comme nous l'avons déjà constaté, l'élimination d'un grand nombre de facteurs possibles, de l'ordre de $\sqrt[4]{n}$.

Voyons maintenant comment l'algorithme de Fermat, en fait, prolonge la portée de la fonction *split*. Que se passe-t-il, par exemple, si le nombre à factoriser peut s'exprimer comme un produit de facteurs de même parité $n = pq$ mais que $\sqrt{2} \le \left|\sqrt{p}-\sqrt{q}\right| = c$?

Remarquons ici que $\left(\sqrt{p}-\sqrt{q}\right)^2 = p+q-2\sqrt{pq}$ donc, si $\left|\sqrt{p}-\sqrt{q}\right| = c$, alors on aura $\frac{c^2}{2} = \frac{p+q}{2} - \sqrt{pq}$. Cette relation permet donc d'exprimer le nombre d'itérations requis par l'algorithme de Fermat, pour trouver $\frac{p+q}{2}$, en termes de la distance entre $\sqrt{p}$ et $\sqrt{q}$. Notons qu'il est nécessaire d'arrondir par le haut, la relation s'exprimant plus précisément par *nombre d'itérations* = $\left\lceil \frac{c^2}{2} \right\rceil$. Ainsi, dans

l'exemple original de Fermat, on a p = 44021 et q = 46061, alors $c = \left|\sqrt{44021} - \sqrt{46061}\right| \cong 4.8064454$ et $\left\lceil \frac{c^2}{2} \right\rceil$=12, ce qui correspond effectivement au nombre d'itérations requis pour trouver les facteurs p et q. Nous voulons souligner, au risque de répétition, que cette relation est indépendante de la taille des nombres considérés. Fermat aurait donc pu, tout aussi facilement (en 12 itérations), réussir à factoriser n'importe quel nombre pouvant s'exprimer comme produit de deux facteurs de même parité $n = pq$ pourvu que $\sqrt{2 \times 11} < \left|\sqrt{p} - \sqrt{q}\right| \leq \sqrt{2 \times 12}$, par exemple si (ici voir **Annexe D**, 2.1-Exemple 1a) $p = 2317...$, et $q = 2317...$, deux nombres premiers de 50 chiffres chacun, on a alors $n = 5372...$, un nombre *dur* de 100 chiffres. Ici $c = \left|\sqrt{p} - \sqrt{q}\right| \cong 4.88958...$, d'où $\left\lceil \frac{c^2}{2} \right\rceil = 12$. Il est facile de vérifier que l'algorithme réussit à trouver les facteurs p et q en 12 itérations exactement. La procédure *Fermat*(n, B) suivante (ici voir **Annexe C**, procédure *Fermat*(n, B)) est une implantation de l'algorithme de Fermat dans l'environnement de Maple V. Nous y avons inclus une borne B, pour limiter le nombre d'itérations. La procédure retourne (entre crochets) le nombre [b] d'itérations effectuées, et les facteurs p et q si elle réussit à les trouver à l'intérieur des limites fixées.

Ainsi, si nous appliquons la procédure sur le nombre n ci-haut, avec une borne B, ici arbitrairement fixée à 20, on obtient effectivement une factorisation en 12 itérations: (ici voir encore **Annexe D**, 2.1-Exemple 1a).

On constate également que choisir des nombres p et q premiers, au hasard, dans un voisinage (assez vaste quand même) des nombres précédents, ne change pas significativement la vitesse d'exécution de la procédure. Par exemple: (ici voir **Annexe D**, 2.1-Exemple 1b).

Ces observations permettent d'apprécier toute la puissance de l'algorithme de Fermat, quant à sa capacité de pouvoir exécuter efficacement la factorisation d'une multitude de nombres, petits et grands, *durs* ou pas, et de surcroît, en retournant deux facteurs

non triviaux. Elles lèvent également le voile sur l'apparent paradoxe avec sa performance générale plutôt modeste. Le nombre d'itérations croît en fonction du carré de la distance entre $\sqrt{p}$ et $\sqrt{q}$. Si l'on considère un nombre $n = pq$, où p et q sont des nombres premiers distincts de même taille (disons l chiffres en décimal), chacun choisi au hasard, alors pour un c donné, la probabilité d'avoir $\left|\sqrt{p} - \sqrt{q}\right| < c$ varie nécessairement en fonction de c, qui lui ici, peut varier approximativement selon $0 < c < \sqrt{10^l} - \sqrt{10^{l-1}}$. Sans chercher à formuler explicitement cette probabilité, il est clair qu'elle sera très petite si c est choisi petit. Mais remarquons par contre, que si l'un ou les deux de p ou q sont composés, et que $n = pq$, il peut exister alors p' et q' tels que $n = p'q'$ et $\left|\sqrt{p'} - \sqrt{q'}\right| < c$, auquel cas la procédure retourne les facteurs p' et q'. Par exemple, si $p = 3$ et $q = (5)(31) = 155$, on a alors $n = (3)(155) = 465$; l'exécution de l'algorithme de Fermat va produire les deux facteurs $p' = (3)(5)=15$ et $q' = 31$, soit ceux qui minimisent la distance $\left|\sqrt{p} - \sqrt{q}\right|$, parmi toutes les combinaisons possibles de facteurs p et q de n. Ainsi:

- **Fermat(465,20);**
- [[2], 31, 15]

Une légère modification à l'algorithme (on le laisse simplement continuer jusqu'à la borne B) permet d'obtenir toutes les factorisations possibles jusqu'à B itérations. Appelons cette procédure *Fermat_B*(n, B) (ici voir **Annexe C**). Ainsi, en augmentant la borne B (ici on pose B = 250) on obtient toutes les factorisations possibles de n (ici $n = 465$).

- **Fermat_B(465,250);**
- [[2], 31, 15]
- [[28], 93, 5]
- [[58], 155, 3]
- [[212], 465, 1]

On note ici l'ordre dans lequel les facteurs sont trouvés, soit selon l'accroissement de la distance entre p et q, et l'on remarque également la croissance rapide du nombre d'itérations requis, au fur et à mesure que les facteurs p et q s'éloignent l'un de l'autre.

L'algorithme de Fermat : un autre classique

À la lumière de ce qui précède, il nous a paru intéressant de comparer l'algorithme de Fermat et l'algorithme classique. Si nous nous permettons une analogie, ils sont en quelque sorte complémentaires. Pour le voir, considérons un intervalle d'entiers $1, \ldots, n$. L'algorithme classique procède, par divisions successives, en augmentant le diviseur à chaque itération, jusqu'à ce qu'il trouve un facteur. S'il atteint $\lfloor \sqrt{n} \rfloor$ sans succès, c'est que le nombre est premier. Au fur et à mesure qu'il progresse, nous pouvons bien sûr éliminer tous les nombres déjà essayés, disons de 2 à k; mais nous pouvons aussi éliminer comme facteurs possibles de n tous les nombres de $\frac{n}{k}$ à n. Donc, après k itérations, il reste à chercher les facteurs de n dans un intervalle réduit soit $k+1,\ldots,\frac{n}{k+1}$, les intervalles successifs étant emboîtés. On constate ainsi, que l'algorithme classique fonctionne de l'extérieur vers l'intérieur, chacune des bornes de l'intervalle se rapprochant progressivement de $\lfloor \sqrt{n} \rfloor$. Remarquons ici l'asymétrie de ces intervalles par rapport au pivot $\sqrt{n}$, celle-ci s'amenuisant progressivement.

L'algorithme de Fermat, lui, fonctionne de l'intérieur vers l'extérieur. Si la première itération échoue, nous pouvons alors éliminer, comme facteurs possibles de n, tous les entiers compris dans un intervalle symétrique, disons $a,\ldots,b$ autour de $\sqrt{n}$, d'un rayon d'environ $\sqrt[4]{n}$. Au fur et à mesure que l'algorithme progresse, cet intervalle d'exclusion s'élargit, mais asymétriquement autour de $\sqrt{n}$, la borne gauche allant vers un (1) de plus en plus lentement, celle de droite allant vers n de plus en plus vite.

Un fait paraît toutefois amoindrir l'analogie entre les deux algorithmes, soit la différence importante quant au nombre maximum d'itérations requis, ie. quand n est un nombre premier. Alors que l'algorithme classique nécessite au plus $\lfloor \sqrt{n} \rfloor$ itérations pour le déterminer, celui de Fermat, nous l'avons déjà souligné, en requiert $\frac{n+1}{2} - \lfloor \sqrt{n} \rfloor$. Les remarques du paragraphe précédent font ressortir le fait que, dans l'algorithme de Fermat, certaines des valeurs considérées sont en fait inutiles. Celui-ci part à la recherche de la moyenne arithmétique $\frac{p+q}{2}$ à partir $\lfloor \sqrt{n} \rfloor$, essayant systématiquement toutes les valeurs entières successives, alors que le plus petit des deux facteurs, disons p, se situe entre 1 et $\lfloor \sqrt{n} \rfloor$, et par ce fait impose un maximum de $\lfloor \sqrt{n} \rfloor$ valeurs possibles pour $\frac{p+q}{2}$, n étant fixe. Ainsi, les valeurs possibles pour $\frac{p+q}{2}$ peuvent se calculer avec la formule $\left\lceil \frac{\frac{n}{k}+k}{2} \right\rceil$, avec k variant de 1 à $\lfloor \sqrt{n} \rfloor$.

Voici un exemple: si $n = (7)(43) = 301$, alors $\lfloor \sqrt{n} \rfloor = 17$ et $\frac{p+q}{2} = \frac{7+43}{2} = 25$. En appliquant la formule ci-haut avec des valeurs k décroissantes, allant de 17 à 1, on obtient les valeurs possibles pour $\frac{p+q}{2}$:

18, 18, 18, 18, 19, 19, 20, 21, 22, 23, 25, 29, 33, 40, 52, 77, 151

On constate ici que l'écart, entre deux valeurs successives de cette suite, est supérieur ou égal à 1 pour les valeurs de 23 et plus. Les valeurs manquantes (24, 26, 27, 28, 30,...) sont les valeurs inutiles. Mais il apparaît, de surcroît, que certaines valeurs sont répétées, réduisant ainsi encore plus le nombre de candidats possibles pour $\frac{p+q}{2}$ (ici 25) et qui, bien sûr, se trouve dans la liste. En fait, cette duplication n'est

pas inattendue puisque nous avons établi que, dès les premières itérations, l'algorithme de Fermat permet l'élimination d'un grand nombre de candidats or, nous remarquons que dans la suite ci-dessus, ce sont justement les nombres proches de $\lfloor\sqrt{n}\rfloor$ qui se répètent le plus, la liste allant en s'éclaircissant à mesure que les valeurs augmentent. Nous avons déterminé (on le démontre plus loin) que cette réduction était de l'ordre de $\left(2-\sqrt{3}\right)\sqrt{n}$, ramenant l'algorithme de Fermat, s'il était modifé pour tenir compte des points soulevés, à un nombre d'itérations maximum d'environ $\left(\sqrt{3}-1\right)\sqrt{n}$. Mais, cette modification semble en fait nécessiter une application de l'algorithme classique puisque la production de cette liste requiert, en fait, $\lfloor\sqrt{n}\rfloor$ divisions. Cet obstacle peut toutefois être contourné en combinant, d'une manière ou d'une autre, les deux algorithmes.

Nous présentons ci-après une procédure (voir *Fermat_Classique*(n, B), **Annexe C**) qui traite les deux algorithmes en parallèle, chacun allant à la rencontre de l'autre. On exclut d'entrée le cas *n probablement premier* (voir **isprime, Annexe B**) en effectuant un test efficace à cet effet. On remarque aussi que la composante *Classique* de la procédure ne parcourt que les nombres impairs, car on vérifie aussi d'entrée si $n=2p$ (avec p impair), auquel cas la procédure cherche à factoriser $N=n/2$. Rappelons que l'algorithme de Fermat ne peut factoriser de tels nombres ($n=2p$ avec p impair) puisqu'il nécessite que les facteurs soient de même parité; mais il est capable de retourner rapidement une factorisation non triviale d'un nombre multiple de 4, s'il existe deux facteurs p et q du nombre, tous deux pairs et relativement proches l'un de l'autre. Cette approche permet de réduire environ de moitié le nombre maximum d'itérations de l'algorithme sans exclure la possibilité de trouver rapidement une factorisation non triviale de tels nombres (multiples de 4). Mais remarquons aussi, qu'en traitant les deux algorithmes en parallèle, nous effectuons deux fois plus d'opérations à chacune des itérations. Nous donnons quelques exemples faisant ressortir la contribution de chacun des deux algorithmes (ici voir **Annexe D**, 2.1-Exemple 2).

Une courbe d'efficacité

Nous avons noté au paragraphe précédent que, dans son parcours à la recherche d'une moyenne arithmétique, l'algorithme de Fermat utilisait, à partir d'un certain point, des valeurs inutiles. Dans ce qui suit, nous identifions la valeur critique à partir de laquelle l'algorithme de Fermat devient inefficace, en comparaison avec l'algorithme classique. Pour arriver à ce résultat, nous examinons le comportement d'une fonction (déjà vu!) qui, en fait, définit un intervalle d'exclusion pour le plus petit facteur p d'un nombre donné $n = pq$, à mesure que l'algorithme s'exécute.

Remarquons d'abord que, si l'algorithme de Fermat a effectué $k \geq 1$ itérations à partir de $x_0 = \lfloor \sqrt{n} \rfloor$ sans réussir à trouver $\frac{p+q}{2}$, on a alors $\frac{p+q}{2} > x$ où $x = x_0 + k$.

Cette relation est équivalente à $\left(p + \frac{n}{p} \right) > 2x$

ce qui entraîne $p^2 + n > 2px$ et par suite $p^2 - 2px + n > 0$, une inéquation du second degré en p avec comme solutions $p < x - \sqrt{x^2 - n}$ ou $p > x + \sqrt{x^2 - n}$. Nous devons rejeter cette dernière, si nous considérons que p est le plus petit des deux facteurs, donc plus petit que $\sqrt{n}$; toutefois, un raisonnement similaire, utilisant q au lieu de p, mène à conclure que $q > x + \sqrt{x^2 - n}$. En combinant ces deux relations nous obtenons un intervalle (symétrique autour de x) d'exclusion des facteurs p et q de n, en fonction de la dernière valeur x vérifiée par l'algorithme : $p,q \notin \left[x \pm \sqrt{x^2 - n} \right]$. La représentation graphique (pour un exemple de tracé voir **Annexe D**, 2.1 Exemple 3) des fonctions $f = (x,n) \rightarrow x - \sqrt{x^2 - n}$ et $g = (x,n) \rightarrow x + \sqrt{x^2 - n}$ pour des valeurs $x \geq \sqrt{n}$ permet de visualiser le parcours suivi par l'algorithme de Fermat et met en évidence l'intervalle d'exclusion de p et q au fur et à mesure que l'algorithme progresse.

Remarquons maintenant, que cette représentation permet aussi de voir le tracé de la fonction f comme courbe d'efficacité (décroissante!) de l'algorithme. En effet, la pente de cette courbe, entre deux valeurs entières successives de x, donne le rapport du nombre de candidats facteurs p éliminés par itération, l'algorithme de Fermat devenant inefficace (par rapport à l'algorithme classique) quand celle-ci devient supérieure à -1. En traitant la fonction f comme une fonction réelle continue et différentiable, on détermine approximativement cette valeur critique en cherchant la valeur x pour laquelle la dérivée $\frac{df}{dx} = -1$. Le calcul de la dérivée donne $\frac{df}{dx} = 1 - \frac{x}{\sqrt{x^2 - n}}$, et par suite $1 - \frac{x}{\sqrt{x^2 - n}} = -1$ donne la solution $x = \frac{2\sqrt{3}}{3}\sqrt{n}$. Ainsi, au-delà de cette valeur, il faudra que l'algorithme effectue plus d'une, et de plus en plus d'itérations, pour éliminer chaque candidat facteur supplémentaire. Pour atteindre cette valeur critique, l'algorithme de Fermat doit effectuer $\frac{2\sqrt{3}}{3}\sqrt{n} - \sqrt{n} = \left(\frac{2\sqrt{3}}{3} - 1\right)\sqrt{n}$ itérations, et si cette valeur est atteinte sans succès, alors nous pouvons éliminer tous les entiers compris entre $f\left(\frac{2\sqrt{3}}{3}\sqrt{n}, n\right) = \frac{\sqrt{3}}{3}\sqrt{n}$ et $\sqrt{n}$, soit $\left(1 - \frac{\sqrt{3}}{3}\right)\sqrt{n}$ candidats facteurs. Afin d'apprécier ces valeurs, remarquons que $\frac{2\sqrt{3}}{3} - 1 \cong 0.154700539$ et que $1 - \frac{\sqrt{3}}{3} \cong 0.4226497310$.

L'identification de cette valeur critique, permet de combiner l'algorithme de Fermat et l'algorithme classique de manière séquentielle, en appliquant l'algorithme de Fermat sur les valeurs x comprises entre $\sqrt{n}$ et $\frac{2\sqrt{3}}{3}\sqrt{n}$, et l'algorithme classique sur les valeurs comprises entre 1 et $\frac{\sqrt{3}}{3}\sqrt{n}$. Ainsi, le nombre maximum d'itérations des deux algorithmes combinés serait d'environ $\left(\frac{2\sqrt{3}}{3} - 1\right)\sqrt{n} + \frac{\sqrt{3}}{3}\sqrt{n}$

$=\left(\sqrt{3}-1\right)\sqrt{n}$, ce qui se traduit par une réduction de $\left(2-\sqrt{3}\right)\sqrt{n}$ itérations, par comparaison à $\sqrt{n}$, tel que mentionné précédemment.

L'application d'un algorithme combiné, tel que décrit ci-dessus, demeure, bien sûr, d'utilité limitée (sinon nulle) face à certaines classes de nombres, tels les *durs*. Dans les pires cas, la recherche d'une solution se traduit encore par un nombre d'itérations de $O\left(\sqrt{n}\right)$, ce qui est inapplicable quand n est grand. Mais elle peut s'avérer utile pour certaines situations:

- Si le nombre à factoriser est composé et relativement petit, alors l'une ou l'autre des composantes de l'algorithme trouve rapidement un facteur.

- Si le nombre à factoriser est composé et relativement grand, mais autrement quelconque, il y a de fortes chances qu'il ait au moins un petit facteur, en quel cas la composante classique le trouve rapidement.

- Si le nombre à factoriser est composé de deux grands facteurs premiers (*dur*), et que ceux-ci sont relativement proches l'un de l'autre, ou si le nombre a plusieurs facteurs et qu'il existe une représentation du nombre comme produit de deux facteurs de même parité, relativement proches l'un de l'autre, alors la composante Fermat trouve facilement une factorisation.

La combinaison a un autre avantage : si la procédure est arbitrairement limitée en nombre d'itérations et qu'elle échoue dans sa recherche d'un facteur, il est possible et facile d'identifier précisément des intervalles d'exclusions de facteurs possibles, en fonction du nombre d'itérations effectuées. Pour la partie classique, ce sont évidemment toutes les valeurs déjà vérifiées, tandis pour la composante Fermat l'intervalle se calcule à partir de la fonction f définie précédemment. Il est intéressant de constater, par exemple, que si l'algorithme de Fermat est utilisé sans succès pendant $\sqrt[3]{n}$ itérations, nous pouvons éliminer comme facteurs possibles toutes les

valeurs entre $f\left(\sqrt{n}+\sqrt[3]{n},n\right)$ et $\sqrt{n}$. Un calcul explicite révèle que cela permet d'éliminer $n^{\frac{1}{3}}\left(\sqrt{2n^{\frac{1}{6}}+1}-1\right) \cong n^{\frac{5}{12}}$ candidats.

Dans un contexte général de factorisation, l'algorithme classique a depuis fort longtemps trouvé sa place comme algorithme de première ligne pour identifier, puis éliminer les petits facteurs d'un (grand) nombre. C'est là que réside sa force, mais c'est aussi la limite de son utilité. Des algorithmes plus performants sont ensuite utilisés pour tenter de factoriser le facteur demeurant. Il est quelque peu étonnant de constater que l'algorithme de Fermat, considéré aujourd'hui de valeur historique seule, ne soit pas utilisé de manière similaire. Notre analyse a montré qu'il est, en fait, très performant dans ses premières itérations, et réussit efficacement à trouver une factorisation non triviale pour une vaste classe de nombres. Notre analyse a, en fait, permis de préciser cette classe, en termes de distance entre certains facteurs et en nombres d'itérations de l'algorithme. Ainsi, nous avons vu que si on limite l'exécution de l'algorithme à k itérations (posons ici $k=\frac{c^2}{2}$), l'algorithme de Fermat retourne une factorisation non triviale d'un nombre n, s'il existe deux entiers de même parité, p et q, tels que $n=pq$ et $\left|\sqrt{p}-\sqrt{q}\right|<c$. Nous avons également vu comment, suite à une exécution limitée de l'algorithme, un échec peut se traduire par l'exclusion d'un grand nombre de facteurs possibles, tous situés à l'intérieur d'un intervalle précisé par les fonctions f et g discutées plus haut. Cette information peut permettre de limiter une recherche ultérieure à l'aide d'algorithmes plus performants. Dans la prochaine section, nous voyons comment élargir, un peu plus encore, la portée de cette méthode.

2.2 À un facteur multiplicatif près : algorithme de Lehman

Le critère de factorisation établi précédemment a permis de constater l'efficacité de l'algorithme de Fermat quand il s'agit de factoriser un nombre n pour lequel il existe deux facteurs p et q, de même parité, *relativement* proches l'un de l'autre. Mais nous avons également constaté qu'il est tout autant inefficace pour traiter un nombre n qui n'a pas de tels facteurs. La combinaison Fermat_Classique surmonte en partie cette difficulté, puisque elle permet de trouver rapidement un *petit* facteur, s'il en est; mais elle reste généralement inefficace devant un nombre *dur*, produit de deux facteurs premiers choisis au hasard. Nous montrons ici comment l'introduction d'un facteur multiplicatif dans l'algorithme de Fermat permet d'élargir la classe de nombres factorisables efficacement. Nous discutons d'une méthode et d'un théorème dus à Lehman et qui, de fait, généralise la méthode de Fermat à l'aide de multiplicateurs, pour en tirer un algorithme de factorisation de complexité significativement réduite. Nous en élaborons une version paramétrisée, ce qui nous permet ensuite d'établir une association avec une classe de nombres factorisables efficacement par cet algorithme. Nous revenons, en fin de section, sur la question d'intervalles d'exclusion de facteurs.

À propos de clefs et de paramètres

Nous définissons en premier lieu, de manière générale, deux notions utiles:

Définition : Nous disons qu'un *algorithme de factorisation* est *paramétrisé*, s'il est capable d'accepter, en entrée, une information sous forme de paramètre(s), et d'utiliser celle-ci pour diriger ou limiter sa recherche de facteurs.

Définition : Soit n un entier quelconque et soit A un algorithme de factorisation paramétrisé quelconque. Une *clef de factorisation* pour le couple (A, n) est une information X qui, fournie à A, lui permet de trouver efficacement une factorisation de n.

Si l'on modifiait l'algorithme de Fermat de façon à permettre la recherche d'une solution $\frac{p+q}{2}$ non pas à partir de $\lfloor \sqrt{n} \rfloor$ mais en commençant par une valeur x quelconque, passée en paramètre, alors un exemple immédiat de *clef de factorisation*, pour cet algorithme et ce nombre n, serait une valeur $x \leq \frac{p+q}{2}$, suffisamment proche de cette dernière, pour arriver à une factorisation en temps raisonnable. Avant de donner un deuxième exemple, moins évident celui-là, nous allons modifier l'algorithme de Fermat pour permettre le passage d'un nouveau paramètre : *clef*. Les changements par rapport à l'algorithme original sont peu nombreux. Nous multiplions en premier le nombre à factoriser n par la *clef*, pour obtenir n', puis on applique la procédure originale sur n'. Si on parvient à séparer n' en $[p', q']$, dans les limites fixées, on calcule alors un facteur p de n par pgcd(n, p'), ce dernier calcul s'effectuant efficacement. On remarquera que si $clef = 1$, on retombe alors dans l'algorithme original. Nous appelons cette procédure *Fermat_Clef* (n, *clef*, B) : (ici voir **Annexe C**). Comme avec la procédure *Fermat*, celle-ci retourne [b], le nombre d'itérations effectuées, et les facteurs p et q, si elle les trouve.

Considérons maintenant les deux nombres premiers p et q suivants, chacun de 100 chiffres, et leur produit n, un nombre *dur* de 200 chiffres. Pour ces nombres, on peut vérifier que la distance $\left|\sqrt{p} - \sqrt{q}\right|$ est de l'ordre 10^{50} ce qui implique que la factorisation de n est est hors de portée (efficace) de l'algorithme original de Fermat puisque qu'elle nécessiterait environ 10^{100} itérations. Pourtant: (ici voir **Annexe D**, 2.2 Exemple 1)

p := 8057..., q := 5755..., n := 46368...,

- **Fermat_Clef(n,35,100);**
- [[1], 5755..., 8057...]

On constate de cet exemple, que l'utilisation de la *clef de factorisation* (ici 35), permet une factorisation efficace du nombre n, la procédure étant effectuée en une seule itération! On constate maintenant qu'en fait, cette même clef conduit à la factorisation efficace d'une multitude de nombres, *durs* et autres. Ainsi, si on prend les nombres premiers p et q suivants (ici voir **Annexe D**, 2.2 Exemple 2), chacun maintenant de 150 chiffres, on vérifie que la factorisation de leur produit n (300 chiffres!) s'effectue tout aussi rapidement avec la même clef, et l'on pourrait en faire tout autant avec une infinité d'autres, sans égard au nombre de chiffres qui les compose. Ainsi :

p := 876798..., q := 626284..., n:= 549125233...

- **Fermat_Clef(n,35,10);**
- [[1], 626284..., 876798...]

On constate maintenant que cette clef (ici 35) n'est pas la seule qui permet de factoriser le nombre n précédent. Si, par exemple, on prend $clef = (p-1)(q-1) = \phi(n)$ on obtient encore la factorisation en une seule itération (la connaissance de $\phi(n)$ permet évidemment la factorisation de n puisqu'elle peut se ramener à une équation quadratique qui se résout efficacement ([17], p.122)). Mais on constate qu'ici on peut tout autant utiliser une approximation de $\phi(n)$ comme $(p+1)(q+3)$ par exemple, ou encore utiliser un multiple (carré) de la clef. Voici ces quelques exemples : (ici voir **Annexe D**, 2.2 Exemple 3)

On constate ainsi que dans chaque cas la procédure trouve les facteurs dès la première itération, d'autres exemples pouvant nécessiter tout au plus quelques itérations supplémentaires. La clef de cette fonctionalité réside entièrement dans le rapport $\frac{q}{p}$. En effet, ici nous avons (bien sûr!) choisi nos exemples de la manière suivante:

- On a d'abord choisi au hasard un nombre premier p d'une longueur prédéterminée (dans les exemples: 100 puis 150 chiffres).
- On a choisi ensuite q selon $q = nextprime\left(\left\lfloor \frac{5}{7}p + Rnd(40) \right\rfloor\right)$ assurant ainsi entre q et p un rapport proche de $\frac{5}{7}$.

On souligne que l'ajout d'un nombre aléatoire, ici d'une quarantaine de chiffres, dans le calcul de q, n'a pas affecté la vitesse d'exécution de la procédure, laissant entrevoir l'ampleur de la classe couverte par une *clef* donnée. En multipliant le nombre à factoriser par la clef, ici 35, on obtient un nombre $n' = 35n = 35pq = (5p)(7q)$ et puisque q et p sont dans un rapport proche de $\frac{5}{7}$ on aura alors $\left|\sqrt{5p} - \sqrt{7q}\right| = c$, avec c petit, ce qui permet à la procédure de séparer n' rapidement et, par suite d'obtenir p ou q en calculant le pgcd de n et d'un des deux facteurs trouvés. De manière équivalente, on peut observer qu'en fait, on transpose le problème de factoriser le nombre n à celui de factoriser n', pour lequel il existe deux facteurs, ici $(5p)$ et $(7q)$, tels que le rapport $\sqrt{\frac{5p}{7q}}$ sera suffisamment proche de l'unité (1), pour rendre possible une factorisation efficace par l'algorithme. L'utilisation de $\phi(n)$, ou de ses *voisins* comme $(p+1)(q+3)$, comme multiplicateur (*clef*), permet tout autant d'obtenir un n' composé de deux facteurs dont le rapport sera proche de un, ici $p' = p(q+3)$ et $q' = q(p+1)$. Dans le cas de la multiplication de la *clef* (35) par un carré a^2 quelconque, chaque a se distribue sur chacun des facteurs, et si a et c sont relativement petits, l'effet est, comme nous l'avons observé, négligeable.

La connaissance du rapport exact $\frac{q}{p}$ constitue bien sûr la clef parfaite (la solution!) puisque une simple manipulation algébrique, étant donné $n = pq$, permet alors de calculer p et q. Ce que nous avons toutefois observé ici, c'est que la connaissance d'une valeur approximative du rapport permet aussi une factorisation efficace

(quoique l'on ait utilisé le rapport 5/7 dans la construction de q dans nos exemples, q et p étant deux nombres premiers différents de 5 et 7, il est clair que leur rapport n'est pas exactement 5/7). Cette propriété, qui découle directement de celles démontrées au chapitre précédent, nous le voyons plus loin, conduit à diverses applications. Au chapitre suivant, nous nous en servons dans un contexte cryptographique, pour l'élaboration d'un cryptosystème probabiliste; mais voyons d'abord comment elle s'applique à l'algorithme de Fermat.

La méthode de Lehman

Le fait, que la méthode de Fermat est performante quand le nombre à factoriser est composé de deux facteurs *premiers*, dont le rapport $\frac{q}{p}$ est *proche* de un, apparaît dans un article de R. Sherman Lehman [11] datant de 1974. Dans cet article, Lehman fait lui-même référence à un article de F.W.Lawrence (1895), où ce dernier propose une méthode efficace pour factoriser un nombre tel que le rapport $\frac{q}{p}$ est *proche* d'un rapport $\frac{a}{b}$, a et b étant petits et relativement premiers. L'analyse de Lehman le conduit à démontrer un théorème existentiel, puis à élaborer un algorithme basé sur la méthode de Fermat, en ramenant le temps moyen d'exécution à $O(n^{1/3})$. Nous ne revenons pas sur la démonstration du théorème; elle est complexe, ingénieuse, technique, faisant appel à la théorie des nombres, au calcul differentiel, et à des suites, semblables aux suites de Farey, qui servent à subdiviser l'intervalle continu [0,1], et à coincer le rapport $\frac{q}{p}$ dans une suite d'intervalles emboités, et elle s'étend sur plusieurs pages. Nous en reproduisons toutefois l'énoncé (que nous avons traduit librement) pour fin de commentaires et de référence immédiate, puis nous présentons une version adaptée de l'algorithme de Lehman, version qui est en continuité avec *Fermat_Clef*.

Théorème de Lehman

Soit n un entier positif impair et soit r un entier tel que $1 \le r < n^{1/2}$.

Si $n = pq$, où p et q sont des nombres premiers, et si $\left(\frac{n}{r+1}\right)^{1/2} < p \le n^{1/2}$

Alors il existe des entiers non-négatifs x, y et k tels que

1. $x^2 - y^2 = 4kn$, où $1 \le k \le r$

$x \equiv k+1 \bmod 2$

$x \equiv k+n \bmod 4$ si k est impair

$0 \le x - \sqrt{4kn} \le \left(\frac{1}{4(r+1)}\right) \cdot \left(\frac{n}{k}\right)^{1/2}$ et

2. $p = \min(\text{pgcd}(x+y,n), \text{pgcd}(x-y,n))$

Si n est premier alors il n'existe pas de nombre satisfaisant **1.**.

Algorithme de Lehman

Dans son article, Lehman montre aussi comment, si on choisit r proportionnel à $n^{1/3}$ (il propose $r = \frac{1}{10} n^{1/3}$), on obtient un algorithme pouvant factoriser n en $O(n^{1/3})$ itérations. Cet algorithme, que nous avons légèrement adapté pour être en continuité avec le précédent, peut se résumer comme suit :

- On choisit r puis on s'assure, à l'aide de l'algorithme classique, que le plus petit des deux facteurs, disons p, soit tel que $\left(\frac{n}{r+1}\right)^{1/2} < p \le n^{1/2}$. Si r est tel

que suggéré, cette opération nécessite $O(n^{\frac{1}{3}})$ itérations de l'algorithme classique.

- Pour chaque valeur k considérée $(1 \le k \le r)$, appliquer l'algorithme *Fermat_Clef* en limitant le nombre d'itérations par une borne $B_k = \left(\frac{1}{4(r+1)}\right) \cdot \left(\frac{n}{k}\right)^{\frac{1}{2}}$.

L'exécution peut donc se faire selon un ou des appels à $Fermat\_Clef(n, 4k, B_k)$.

Quelques remarques et observations

Remarquons que, pour chaque k essayé, le théorème établit une relation entre la solution x recherchée, et la valeur k. Les deux relations $x \equiv k+1 \bmod 2$, et $x \equiv k+n \bmod 4$ si k est impair, permettent de réduire, pour chaque k, de moitié ou de trois quarts le nombre d'itérations de l'algorithme *Fermat_Clef*, s'il est modifié pour en tenir compte, puisqu'elles précisent un ensemble de valeurs possibles de x. Il suffit, pour réaliser cette modification, de vérifier en premier la parité de k. Si k est pair, alors le x recherché est impair et l'on essaie que des valeurs x $\left(x \ge \sqrt{4kn}\right)$ impaires; tandis que si k est impair, on détermine la plus petite valeur entière $x_0 \equiv k+n \bmod 4$, supérieure ou égale à $\sqrt{4kn}$, puis on effectue les itérations en incrémentant chaque fois x par quatre, à partir de cette valeur initiale x_0.

Dans sa stratégie d'implantation de l'algorithme, Lehman propose d'essayer en premier des valeurs k qui sont égales ou multiples de nombres composés de plusieurs petits facteurs comme: 30, 24, 18, 12 et 6. Il souligne, avec justesse, que cette approche permet d'effectuer plusieurs vérifications simultanées. Par exemple, si $n = pq$, et $k = 30$ alors $4kn = (2 \cdot 3 \cdot p)(2 \cdot 2 \cdot 5 \cdot q)$, ou $4kn = (2 \cdot 5 \cdot p)(2 \cdot 2 \cdot 3 \cdot q)$, ou

autres combinaisons possibles. Si une, quelconque, de ces combinaisons vérifie les conditions de succès de l'algorithme, on obtient alors une factorisation de n.

Remarquons le rôle du multiplicateur quatre (4) utilisé dans la construction du nombre $4kn$. Par hypothèse (du théorème), n est composé d'un produit de deux facteurs premiers (impairs), mais le multiplicateur k peut être pair ou impair. Si k est pair mais non multiple de quatre (4), alors il n'est pas possible de séparer kn en deux facteurs de même parité, en quel cas la factorisation est impossible par la méthode de Fermat, même s'il existe deux facteurs de kn proches l'un de l'autre. La multiplication de kn par quatre (4), permet d'obtenir deux facteurs de même parité, chacun multiple de deux. Notons toutefois que si k est impair, l'utilisation du multiplicateur quatre (4) a pour effet de doubler le nombre d'itérations à effectuer pour trouver une solution, sans contribuer à la parité des facteurs. En effet, supposons que $n = pq$, produit de deux nombres premiers impairs, supposons aussi que $k = ab$, où a et b sont impairs et tels que $\left|\sqrt{ap} - \sqrt{bq}\right| = c$, alors on a $\left|\sqrt{2ap} - \sqrt{2bq}\right| = \sqrt{2}\,c$, ce qui correspond à multiplier par deux le nombre d'itérations vers une solution puisque celui-ci se calcule, nous l'avons démontré, par $\frac{c^2}{2}$. Notons que ceci s'applique, en fait, à tout nombre n impair ou multiple de quatre, et fait contrepoids a l'avantage obtenu en utilisant les relations liant x et k, tel que discuté plus haut.

La méthode de Lehman ajoute, en quelque sorte, une dimension à celle de Fermat. La recherche des facteurs, d'un nombre n donné, s'effectue selon la méthode originale, mais elle se fait sur un ensemble de multiples ($4kn$) du nombre. Pour chacune des valeurs k essayées, le nombre d'itérations de l'algorithme est limité par une borne B_k qui diminue à mesure que k augmente, et par les conditions imposées sur les valeurs x possibles. Le théorème de Lehman garantit l'existence d'une solution dans les limites fixées et cette solution se trouve en au plus $O(n^{1/3})$ itérations, qui représente la somme des itérations effectuées pour chaque valeur k possible. L'algorithme de Lehman marque une étape importante dans la recherche d'une solution générale au problème de la factorisation. Selon H. Cohen ([5], p.418) qui

traite en détail du problème et d'algorithmes de factorisation, cette méthode est la première, dont le temps d'exécution a formellement été démontré être significativement inférieur à $O(n^{\frac{1}{2}})$. Mais, quoique relativement récent, l'algorithme de Lehman est rapidement supplanté par des algorithmes globalement plus performants et, comme celui de Fermat, déjà relegué aux oubliettes : « [...] is now only of historical interest. » ([5], p.419).

Une question de stratégie

Les limitations de l'algorithme de Lehman sont claires : si le nombre à factoriser est *dur*, composé de deux grands facteurs premiers choisis au hasard, il y a peu de chance de réussir une factorisation en temps raisonnable. Une stratégie, où le choix des multiplicateurs k se fait selon un ordre de priorité, tel que suggéré par Lehman, peut permettre une certaine optimisation du processus de recherche d'une solution; mais, sans la connaissance d'une *clef de factorisation*, l'exécution de l'algorithme risque d'être longue. Toutefois, à l'instar de l'algorithme de Fermat, celui de Lehman aussi permet la factorisation efficace d'une multitude de nombres *durs* et autres et, à ce titre, devrait possiblement avoir une place, si petite soit-elle, à l'intérieur d'une stratégie globale de factorisation. Pour appuyer ce point de vue, mentionnons que nous avons comparé, à travers divers exemples, l'efficacité de la fonction *ifactor* , native au logiciel Maple V et reconnue très performante, versus celles d'algorithmes adaptés de ceux de Fermat et Lehman. Nous avons pu construire à volonté, selon une méthode aléatoire, des nombres *durs* que la fonction *ifactor* ne réussissait pas à factoriser, même après plusieurs heures d'exécution, tandis qu'avec la méthode de Fermat, ou par extension celle de Lehman, on obtenait généralement une solution assez rapidement. Les nombres utilisés étaient construits selon la méthode suivante :

- Fixer (pseudo) aléatoirement la taille l d'un facteur premier. Dans nos essais, on a limité le nombre de chiffres l, à 250 au maximum.
- Générer (pseudo) aléatoirement un nombre p premier, de taille l. Cette étape est effectuée en générant d'abord un nombre aléatoire de taille l, puis en

cherchant un nombre premier proche, à l'aide d'une fonction comme *nextprime* .

- Générer (pseudo) aléatoirement deux nombres entiers a et b, chacun constitué d'un maximum de m chiffres. Dans nos esssais on a limité m à quelques (4-5) chiffres.
- Générer (pseudo) aléatoirement un nombre entier d, constitué de $\frac{l}{2}$ chiffres ou moins (ou choisir d tel que $bd < \sqrt{p}$) .
- Construire un deuxième nombre premier q comme suit : $q = nextprime\left(\left\lfloor p \cdot \frac{a}{b} \pm d \right\rfloor\right)$.
- Former $n = pq$.

Nous avons présenté, plus haut, des exemples de nombres construits par cette méthode. Toutefois, pour ces exemples, on a fixé d'avance le rapport $\frac{a}{b}$ (on a utilisé $\frac{5}{7}$). Nous avons pu observer que, pour ces nombres, la procédure *Fermat_Clef* réussit à trouver rapidement les facteurs p et q. La connaissance préalable de la clef (35) a, bien entendu, facilité, dans nos exemples, la détermination de la solution. Sans elle, il en dépend entièrement de la stratégie utilisée dans la recherche d'une clef $k = ab$, et du rapport véritable (inconnu) $\frac{q}{p}$. Mais, soulignons qu'il en dépend aussi d'une stratégie de recherche en profondeur versus en largeur, où chercher en profondeur consisterait ici, pour une valeur k donnée, à appliquer *Fermat_Clef* jusqu'à la borne B_k , tandis que chercher en largeur reviendrait à parcourir les valeurs k selon une stratégie préétablie. Pour clarifier, considérons l'algorithme suivant:

- En entrée un nombre $n = pq$ à factoriser

- Initialiser $r \cong n^{1/3}$, entier ou un multiple prédéterminé, et

  $$B_{\max} = \left(\frac{1}{4(r+1)}\right) \cdot n^{1/2} \cong n^{1/6}$$

- Pour B variant de 1 à $B_{\max}$ faire
  - Pour k variant de 1 à r faire

    (remarque : certaines valeurs k seront inutiles)

    - Si $B > B_k = \left(\frac{1}{4(r+1)}\right) \cdot \left(\frac{n}{k}\right)^{1/2}$ alors boucler sur B

      (retour à l'étape précédente en posant $k = r$)
    - Sinon calculer $x_k = \lfloor \sqrt{4kn} \rfloor$, puis $x = x_k + B$
    - Si $x^2 - 4kn$ est un carré (on a une solution), alors

      $p = \text{pgcd}(x - \sqrt{x^2 - 4kn}, n)$ et $q = \frac{n}{p}$. Poser alors

      $B = B_{\max}$ et $k = r$ pour terminer la recherche
  - Boucler sur k
- Boucler sur B

- Retourner $[p,q]$ et terminer.

Sauf pour son manque à filtrer certaines valeurs k inutiles, cet algorithme respecte les conditions du théorème de Lehman et ainsi retourne, en environ $O(n^{1/3})$ itérations, les facteurs d'un nombre, si celui-ci est tel que spécifié dans l'énoncé du théorème, i.e. $n = pq$ où p et q sont premiers et $\left(\frac{n}{r+1}\right)^{1/2} < p \leq n^{1/2}$. Remarquons toutefois que, pour cet algorithme, la recherche d'une solution se fait en parcourant, pour chaque valeur B considérée, l'ensemble des valeurs k possibles, dans cet ordre et non l'inverse, soit une recherche en largeur d'abord. Remarquons aussi que nous avons choisi ici, tout simplement, de parcourir successivement les entiers k sans chercher à optimiser l'ordre des valeurs. Si nous appliquons cet algorithme au nombre n (300 chiffres) vu plus haut, il retourne rapidement la factorisation puisque, pour ce

nombre, une solution se trouve à $k = 35$ et $B = 1$, laquelle est vite atteinte. Il est clair aussi, que pour cet exemple, si (dans l'algorithme) la stratégie de recherche d'une solution (k, B) est modifiée pour se faire en profondeur d'abord, il n'est pas possible de trouver une solution en temps raisonnable. En effet, si on respecte les contraintes sur B, la clef minimale est $k = 35$, alors pour chacune des valeurs $k = 1,2,3,...,34$ l'algorithme va effectuer $B_k \cong n^{1/6}$ itérations sans déceler une solution, avant d'arriver à ($k = 35$, $B = 1$); or, pour un nombre de 300 chiffres $n^{1/6} \cong 10^{50}$, ce qui nécessiterait une exécution bien longue.

L'algorithme présenté au paragraphe précédent peut, en fait, trouver efficacement la factorisation de tout nombre généré par la méthode décrite plus haut. Ceci s'explique par le fait que pour chaque nombre $n = pq$ construit par cette méthode, il existe une clef $k = ab$ (si a et b sont limités à cinq chiffres chacun, alors $k_{max} \cong 10^{10}$) et, pour cette clef k, on a nécessairement $B = 1$, car dans l'expression $q = nextprime\left(\left\lfloor p \cdot \frac{a}{b} \pm d \right\rfloor\right)$, la valeur aléatoire d est de l'ordre de $\sqrt{p}$ ou moins. Ainsi, à l'intérieur des limites fixées pour les paramètres a, b, et d, on obtient une factorisation en au plus 10^{10} itérations, le temps de trouver la clef k. Il est clair de ce qui précède que, si on augmente les limites fixées arbitrairement aux paramètres aléatoires a et b, le temps maximum d'éxécution de l'algorithme augmente alors proportionnellement. Regardons maintenant ce qui se produit, si nous permettons au paramètre aléatoire d d'avoir des valeurs supérieures à celles fixées précédemment. Dans ce cas, on peut constater que pour

$$4kn = 4abpq = (2ap)\cdot(2bq), \text{ et } \left|\sqrt{2ap} - \sqrt{2bq}\right| = c,$$

si on a $2bq \cong 2b(\frac{a}{b}p \pm d) = 2(ap \pm bd)$,

alors $\left|\sqrt{2ap} - \sqrt{2bq}\right| \cong \left|\sqrt{2ap} - \sqrt{2(ap \pm bd)}\right|$

qui augmente à mesure que d est augmenté. On peut observer expérimentalement, que si la taille de d est inférieure ou égale à environ la moitié de celle de p, et que a et b sont petits, alors le paramètre d n'a généralement aucune influence sur le temps d'exécution de l'algorithme, et dans ce cas, une solution se trouve toujours à $(k = ab,\ B = 1)$. Par contre, chaque chiffre de plus ajouté à d, augmente par un facteur d'environ 10^2 la valeur solution pour B. Par exemple, si dans la méthode de construction de nombres *durs* vue plus haut on génère un d aléatoire, avec disons deux chiffres de plus que la moitié de chiffres dans p, on a une solution à $(k = ab, B \cong 10^4)$. Il est clair que, dans ce cas, si le nombre à factoriser est grand (300 chiffres, par exemple), la stratégie de recherche utilisée dans l'algorithme, tel qu'il est défini, n'est plus adéquate puisqu'elle ne permet pas de trouver une solution en moins de $n^{1/6} \cong 10^{50}$ itérations, comme mentionné précédemment. Ces observations, qui sont conformes aux théories vues au chapitre I, mettent en évidence certains faits :

On conviendra que si un nombre est le produit de deux facteurs premiers, $n = pq$, il est toujours possible d'exprimer un des facteurs, disons q, en terme de l'autre à l'aide d'une relation telle $q = nextprime\left(\left\lfloor p \cdot \frac{a}{b} \pm d \right\rfloor\right)$, i.e. il existe toujours des entiers a, b, et d qui valident cette relation; en fait, il en existe une infinité puisque de ces trois paramètres, deux sont libres. Si l'on tente une factorisation de n à l'aide d'une méthode *à la Fermat-Lehman*, pour chaque valeur clef $k = ab$ essayée, correspond une valeur d qui valide la relation, et par suite une valeur clef $B = B_d$, c'est-à-dire qu'une solution se situe à $\left(k = ab, B = B_d\right)$. Plus le rapport $\frac{a}{b}$ est proche du véritable rapport $\frac{q}{p}$, plus la valeur de B_d sera petite, et cet effet est d'autant plus prononcé lorsque les valeurs a et b sont petites, ce qui correspond à raccourcir la recherche en profondeur. Les questions qui se posent alors naturellement ici sont des questions de stratégies :

- Dans quelle ordre faut-il choisir les valeurs k ?
- Est-il préférable d'effectuer d'abord une recherche en profondeur jusqu'à la borne maximale B_k, pour chaque valeur k essayée, ou vaudrait-il mieux parcourir toutes les valeurs k en fixant d'abord $B = 1$, puis en recommencant avec $B = 2$, etc., ou peut-être encore de combiner les deux approches en choisissant, par exemple, pour chaque k, de tenter de valider certaines valeurs B arbitrairement déterminées selon le nombre à factoriser ?

Dans son article, Lehman propose, comme nous l'avons déjà remarqué, de choisir d'abord des valeurs k qui sont des multiples de nombres composés de petits facteurs premiers et pour chaque valeur k, vérifier successivement toutes les valeurs B jusqu'à la borne maximale B_k. Cette stratégie a certes les avantages déjà soulignés et se révèle très appropriée quand peu est connu sur le nombre à factoriser; d'ailleurs, Lehman l'illustre en donnant la factorisation d'une dizaine de nombres, chacun d'un ordre de grandeur d'environ 10^{20} (des *durs* d'alors!). En fait, si l'on ne connait rien d'autre sur la nature d'un nombre à factoriser, sauf qu'il est le produit de deux facteurs premiers, la stratégie proposée par Lehman est, sans doute, la meilleure pour cette méthode de factorisation. Mais il y a aujourd'hui, pour factoriser de tels nombres, des algorithmes généralistes beaucoup plus performants.

Sur une association possible : classe – méthode

Ce qui parait toutefois ressortir de notre analyse, c'est la relation que l'on peut établir entre la structure d'un nombre, vue d'une certaine perspective, et la possibilité de le factoriser efficacement à l'aide d'une méthode telle que décrite. Prenons par exemple la classe des nombres générés par la méthode élaborée plus haut. En théorie cette classe, disons C, est infinie, contient des nombres de toutes tailles, et tous sont composés de deux nombres premiers. Formellement :

- $n \in C$ *si et seulement si* il existe p et q premiers tels que $n = pq$, et il existe a, b, et d, des entiers tels que $q = nextprime\left(\left\lfloor p \cdot \frac{a}{b} \pm d \right\rfloor\right)$, et tels que $bd < \sqrt{p}$.

Pour cette classe C, l'algorithme présenté plus haut sera efficace pour factoriser n dans la mesure ou $k = ab$ est atteignable en temps raisonnable. La classe C s'élargit si on permet au paramètre d des valeurs plus grandes mais, comme nous l'avons déjà remarqué, pour des nombres de 200-300 chiffres, l'algorithme discuté devient alors inadéquat. Toutefois, si nous le modifions pour permettre une recherche partielle en profondeur à chaque passage sur une valeur k, l'algorithme pourrait alors factoriser efficacement, en fonction de la profondeur permise, une classe élargie. En fait, il faudrait plutôt remarquer ici que quelle que soit la statrégie utilisée, il y a toujours une classe de nombres correspondante pour laquelle la factorisation peut se faire efficacement; celles que nous avons présentées en sont des exemples aisément formulables. La possibilité de pouvoir associer à une méthode donnée une classe de nombres pouvant être factorisés efficacement font, de l'algorithme décrit (ou ses variantes), un algorithme spécialisé, comme par exemple l'algorithme $p-1$ de Pollard, décrit dans ([17], p.133-134), et démontré efficace sur des nombres *durs* quand un des facteurs (disons p) est tel que $p-1$ n'a que des *petits* facteurs (*smooth*).

En résumé, nous avons associé un algorithme de factorisation et une classe de nombres, des *durs* de toutes tailles, et pour tout membre de cette classe, l'algorithme réussit efficacement la factorisation. Afin d'en saisir la portée, considérons la procédure suivante, adaptée des méthodes discutées :

Procédure *Fermat_Lehman* (N, K, B) (ici voir **Annexe C**)

Pour cet algorithme (paramétrisé!), dérivé des méthodes de Fermat et Lehman, nous n'avons pas cherché à filtrer les valeurs k inutiles, ni à tenir compte des valeurs maximales B_k définies précédemment, ceci afin d'en garder l'expression à son plus

simple et d'en faire ressortir les principales caractéristiques. Comme on peut le constater, la procédure accepte en entrée un nombre N à factoriser, et deux intervalles de valeurs entières définis par les paramètres $K = [K_{\min}, K_{\max}]$ et $B = [B_{\min}, B_{\max}]$. L'algorithme parcourt séquentiellement, avec la variable k, l'intervalle de $K_{\min}$ à $K_{\max}$ et, pour chaque valeur k, calcule $n = 4kN$ puis $x = \lfloor \sqrt{n} \rfloor$. Ensuite, pour chaque valeur b, allant de $B_{\min}$ à $B_{\max}$, l'algorithme vérifie si la valeur $z = (x+b)^2 - n$ est un carré, auquel cas une solution est trouvée, et l'algorithme retourne les facteurs p et q ainsi que les valeurs-solutions *ksol* et *bsol*. Cette approche permet d'explorer aisément diverses situations et de vérifier les relations discutées. Nous présentons ci-après trois exemples (et sous-exemples!) qui illustrent les concepts discutés; chacun apporte une nuance ou souligne une particularité :

Exemple I : (ici voir **Annexe D**, 2.2 Exemples 4a et 4b) - Ici, on choisit d'abord aléatoirement l'ordre de grandeur l (de un à 250 chiffres) de chacun des facteurs premiers. On génère ensuite aléatoirement un premier facteur premier p, puis trois nombres a, b et d, (pseudo) aléatoires eux aussi, tels que a et b soient compris entre 10 et 100, et d est tel que $d < \sqrt{p}$. Pour générer les nombres au hasard on se sert, comme précédemment, de la fonction *Rand* (N) qui retourne un nombre (pseudo) aléatoire compris entre 1 et N, et de la fonction *Rnd* (l) qui retourne un nombre (pseudo) aléatoire de l'ordre 10^l. Pour trouver un nombre premier proche d'un nombre donné on se sert encore de *nextprime* :

- **l:=Rand(250);p:=nextprime(Rnd(l));a:=Rand(100);b:=Rand(100); d:=Rand(isqrt(p));q:=nextprime(floor(evalf(a*p/b,length(p)))+d); n:=p*q;**
- $l := 212$, $p := 64630632\ldots$, $a := 71$, $b := 86$, $d := 68807201\ldots$, $q := 53357848\ldots$, $n := 344855150200\ldots$

La méthode a permis ici de générer un nombre *dur* de plus de 400 chiffres. Puisque a et b sont compris entre 1 et 100; on cherche alors la clef k dans l'intervalle $K = [1,$

10000], et puisque $d < \sqrt{p}$, la clef b est petite, et on la cherche dans l'intervalle B= [1,100] :

- **Fermat_Lehman(n,[1,10000],[1,100]);**
- $P = 64630632\ldots$, $Q = 53357848\ldots$, $ksol = 6106$, $bsol = 20$

On constate que la procédure a trouvé la solution à $ksol$ = 6106, qui correspond effectivement à $ab = 71 \times 86$, a et b étant ici relativement premiers, tandis que la valeur $bsol$ égale 20. Ainsi, la factorisation de n a pu être effectuée en $6105 \times 100 + 20 = 610520$ itérations de l'algorithme, soit un temps d'éxecution de l'ordre de 10^6. Sur un ordinateur portable, de très modeste capacité, cette opération a nécessité quelques minutes, dues surtout au traitement de nombres d'aussi grandes tailles, dans un réduit d'espace-mémoire.

Quand on exécute la procédure sur un autre nombre, généré par la même méthode, mais en limitant cette fois les facteurs premiers à moins de 100 chiffres, on obtient un résultat similaire : (ici voir **Annexe D**, 2.2 Exemple 4b) :

- **l:=Rand(100);p:=nextprime(Rnd(l));a:=Rand(100);b:=Rand(100); d:=Rand(isqrt(p));q:=nextprime(floor(evalf(a*p/b,length(p)))+d); n:=p*q;**
- $l := 61$, $p := 90517\ldots$, $a := 48$, $b := 79$, $d := 21001\ldots$, $q := 54998\ldots$,
  $n := 49783096569\ldots$.
- **Fermat_Lehman(n,[1,10000],[1,100]);**
- $P = 90517\ldots$, $Q = 54998\ldots$, $ksol = 3792$, $bsol = 16$

On observe ainsi, qu'à l'intérieur des mêmes intervalles de recherche, la procédure réussit (toujours!) à trouver les facteurs premiers des nombres présentés, dans un temps de l'ordre de 10^6.

Exemple II : (voir **Annexe D**, 2.2 Exemples 5a et 5b) - Nous constatons ici que la réussite de la procédure ne dépend pas du fait que le nombre soit *dur*. Mentionnons

que, dans la démonstration de son théorème, Lehman utilise cette propriété, et ceci contribue à prouver l'existence d'une solution à l'intérieur de certaines limites. Toutefois, comme nous l'avons déjà noté, à chaque valeur k correspond une valeur clef b, et cette propriété ne dépend pas du fait que le nombre à factoriser soit *dur*. Pour cet exemple nous prenons alors des nombres p et q générés selon la méthode précédente, mais cette fois sans utiliser la fonction *nextprime*. (ici voir **Annexe D**, 2.2 Exemple 5a)

- **l:=Rand(100);p:=Rnd(l);a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=floor(evalf(a*p/b,length(p)))+d;n:=p*q;**
- $l := 52$, $p := 778400\ldots$, $a := 67$, $b := 78$, $d := 745525\ldots$, $q := 668626\ldots$, $n := 52045940811\ldots$
- **Fermat_Lehman(n,[1,10000],[1,100]);**
- $P = 778400\ldots$, $Q = 668626\ldots$, $ksol = 5226$, $bsol = 17$

On constate qu'ici encore la procédure retourne les facteurs p et q dans les limites fixées, et ce, malgré le fait qu'aucun de ces deux facteurs ne soit premier. En effet :

- **isprime(p);isprime(q);**
- *false*, *false*

Si nous essayons avec un autre nombre, choisi avec les mêmes paramètres, on obtient alors (ici voir **Annexe D**, 2.2 Exemple 5b) :

- **l:=Rand(100);p:=Rnd(l);a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=floor(evalf(a*p/b,length(p)))+d;n:=p*q;**
- $l := 76$, $p := 7735887\ldots$, $a := 16$, $b := 88$, $d := 3372767\ldots$, $q := 14065250\ldots$, $n := 108807200111\ldots$.
- **Fermat_Lehman(n,[1,10000],[1,100]);**
- $P = 3867943900\ldots$, $Q = 2813050109\ldots$, $ksol = 22$, $bsol = 3$

On constate ici qu'une solution est trouvée à $ksol = 22$; ceci est dû au fait que a et b ont un facteur commun (8) alors $\frac{a}{b} = \frac{16}{88} = \frac{2}{11}$. On remarque aussi que les facteurs retournés diffèrent des valeurs originales par un facteur de deux, et si nous cherchons une autre solution avec $k = ab$ on obtient encore deux facteurs différents:

- **Fermat_Lehman(n,[a*b,a*b],[1,100]);**
- $P = 96698597505\ldots$, $Q = 11252200437\ldots$, $ksol = 1408$, $bsol = 18$

Exemple III : - Dans cet exemple, on choisit aléatoirement un nombre n, quelconque mais de taille plus modeste cette fois (une dizaine de chiffres au plus). On applique ensuite la procédure sur n avec différents intervalles de recherche et on constate l'existence de différentes valeurs solutions: (ici voir **Annexe D**, 2.2 Exemple 6).

Des intervalles d'exclusion

Nous terminons cette section, comme nous l'avons déjà fait, en soulevant la question d'exclusion de facteurs, advenant un échec de la procédure, à l'intérieur des limites fixées pour une exécution. Regardons ce qui se passe si l'on fixe une valeur k. Si pour cette valeur k la procédure parcourt les valeurs b de l'intervalle $\left[1, B_{\max}\right]$ sans déceler de solution, on peut alors conclure, à partir des propriétés démontrées dans la section précédente, que le nombre $N = 4kn$ n'a aucun facteur dans l'intervalle d'entiers $\left[(X_0 + B_{\max}) \pm \left\lfloor \sqrt{(X_0 + B_{\max})^2 - N} \right\rfloor\right]$, où $X_0 = \left\lfloor \sqrt{N} \right\rfloor$. Mais ceci implique que n n'a aucun facteur dans ce même intervalle, et de plus, si nous divisons n par chacune des bornes de cet intervalle, on obtient alors un autre intervalle d'exclusion de facteurs de n, soit $\left[\frac{n}{(X_0 + B_{\max}) \pm \left\lfloor \sqrt{(X_0 + B_{\max})^2 - N} \right\rfloor}\right]$.

L'argument que nous avons présenté ici s'applique, par extension, à chaque valeur k essayée par la procédure. Cela produit, pour l'ensemble de ces valeurs k et selon la

valeur choisie pour $B_{\max}$, une suite d'intervalles d'exclusion des facteurs de n, chacun s'élargissant à mesure que $B_{\max}$ est augmenté.

Terminons cette section par (encore!) un petit exemple : (voir **Annexe D**, 2.2 Exemple 7). Dans la section suivante, nous voyons comment généraliser la méthode de Fermat suivant d'autres axes.

2.3 Sur d'autres généralisations

Notre but, dans cette section, est de développer quelques uns des concepts déjà vus. Nous le faisons dans un contexte d'expressions algébriques un peu plus générales, et en deux temps. En premier, nous revenons sur la méthode de Fermat, pour la situer par rapport à une approche quadratique qui en est inclusive. Quoique sans conséquence, il en découle quand même quelques éclaircissements sur les propriétés et mécanismes qui sous-tendent la factorisation par cette méthode. La seconde partie vise à élargir la portée du critère de factorisation démontré au premier chapitre. Nous le faisons pour des expressions de la forme $x^k \pm y^k$ où $k \geq 2$. Il en découle ici, un algorithme (à la Fermat) capable de factoriser tout nombre pouvant s'exprimer sous cette forme, parfois même assez efficacement. En somme : un algorithme de factorisation efficace *un peu partout*!

- **D'un polynôme à un autre**

Rappelons d'abord que la méthode de Fermat s'applique à des nombres pouvant s'exprimer comme une différence de deux carrés, $n = x^2 - y^2 = (x-y)(x+y)$. Ici, nous considérons plutôt des expressions de la forme $n = (x-y)(x+z) = x^2 + (z-y)x - yz$ où x, y, et z sont des entiers non négatifs tels que $y < \lfloor \sqrt{n} \rfloor \leq x$ et $y \leq z$. Remarquons que tout entier positif peut s'écrire sous cette forme, ou plus précisément, pour tout entier positif n, il existe des entiers non négatifs x, y, et z vérifiant les conditions ci-dessus et tels que $n = (x-y)(x+z)$. En effet, pour tout n positif donné, il existe toujours des entiers p et q tels que $1 \leq p \leq \sqrt{n}$, $\sqrt{n} \leq q \leq n$ et $n = pq$. Il suffit alors de poser $x = \lfloor \sqrt{n} \rfloor$, et de prendre $y = (x-p)$ et $z = (q-x)$, pour avoir au moins une solution. Par exemple, si n = 7029871 = (1237)(5683), alors $x = \lfloor \sqrt{n} \rfloor = 2651$ et l'on peut écrire n = (2651-1414)(2651+3032). Il est clair que, pour cet exemple, il existe plusieurs autres valeurs possibles pour x, y et z. Remarquons toutefois que le nombre de triplets-solutions (x,y,z), respectant les conditions énoncées, dépend du nombre n qui lui est quelconque. Ainsi, si $n = p^2$, où p est un nombre premier, il n'y a que deux possibilités :

- $\left(x = \sqrt{n} = p,\ y = 0,\ z = 0\right)$

  donnant : $n = \left(\sqrt{n} - 0\right)\left(\sqrt{n} + 0\right)$, i.e. $n = (p)(p)$

- $\left(x = \sqrt{n} = p,\ y = p - 1,\ z = p^2 - p\right)$

  donnant : $n = \left(p - (p-1)\right)\left(p + (p^2 - p)\right) = (1)(p^2)$ (trivial)

Regardons maintenant le cas où le nombre n peut s'exprimer comme un produit de deux facteurs p et q consécutifs, i.e. avec $q = p + 1$ (par exemple $n = (41)(42) = 1722$), disons *presque* un carré. Pour un tel nombre, on a toujours $\lfloor \sqrt{n} \rfloor = p$, et la solution $\left(x = \lfloor \sqrt{n} \rfloor = p,\ y = 0,\ z = 1\right)$ donne la factorisation $n = (p - 0)(p + 1)$. À part la solution triviale et, dépendant du nombre de facteurs de n, il existe possiblement d'autres solutions avec des valeurs x supérieures, mais elles donnent d'autres factorisations. Ainsi, pour l'exemple précédent, avec $n = 1722$, le triplet-solution $(x = 45,\ y = 24,\ z = 37)$ mène à la factorisation (45-24)(45+37) = (21)(82).

Pour poursuivre dans le même sens, supposons qu'un nombre n puisse s'exprimer comme $n = pq$ où $q = p + 2$ (par exemple $n = (41)(43) = 1763$). Pour un tel nombre, on a aussi $\lfloor \sqrt{n} \rfloor = p$, et l'on a toujours une solution avec $\left(x = \lfloor \sqrt{n} \rfloor = p,\ y = 0,\ z = 2\right)$ donnant alors $n = (x - 0)(x + 2) = (p)(p + 2)$; mais on remarque que dans ce cas, on a aussi une solution avec $\left(x = \lceil \sqrt{n} \rceil = p + 1,\ y = 1,\ z = 1\right)$ donnant la factorisation $n = (x - 1)(x + 1) = (p)(p + 2)$, qui en fait est la même.

À ce stade, déterminons ce qui se passe en termes de solutions, si l'on augmente la distance entre p et q, une question que nous avons examinée sous un autre angle au premier chapitre. Ici toutefois, nous nous laissons guider par les différences $(z - y)$, qui sont toutes non négatives puisque le modèle proposé exige $y \leq z$. On remarque donc que, pour les cas ci-haut, si l'on prend $x = \lfloor \sqrt{n} \rfloor$, alors ces différences sont petites, (0, 1 ou 2). Maintenant, si l'on augmente l'écart entre les facteurs p et q, par exemple avec

$n = (41)(53) = 2173$, on a $\lfloor \sqrt{n} \rfloor = 46 = p + 5$, et n peut s'écrire comme $(46-5)(46+7)$; on constate ainsi que la différence $(z - y) = (7 - 5) = 2$ n'a toujours pas augmenté. En fait, on peut facilement vérifier que celle-ci va conserver cette même valeur ($z - y = 2$) tant que q restera à l'intérieur d'une certaine distance de p, ou encore, comme nous l'avons déjà vu, si p et q sont de même parité, tant que $\sqrt{q} - \sqrt{p} < \sqrt{2}$. Il est toutefois intéressant de constater (on ne le démontre toutefois pas!), que si p et q sont de parité contraire, on a alors une différence $z - y$ impaire, et celle-ci va égaler un (1) tant que $\sqrt{q} - \sqrt{p} < 1$. Ainsi, par exemple, si $p = 41$, $q = 54$, on a $n = pq = 2214$, $\sqrt{q} - \sqrt{p} \cong 0.945344992 < 1$ et l'on constate que pour le triplet solution $\left(x = \lfloor \sqrt{n} \rfloor = 47, y = 6, z = 7\right)$, on a $z - y = 7 - 6 = 1$. Maintenant, si $p = 41$, $q = 56$, on a $n = pq = 2296$, $1 \leq \sqrt{q} - \sqrt{p} \cong 1.080190537 < \sqrt{2}$, et pour le triplet solution $\left(x = \lfloor \sqrt{n} \rfloor = 47, y = 6, z = 9\right)$ on a $z - y = 9 - 6 = 3$.

L'ensemble de ces observations suggère un algorithme de factorisation basé sur des expressions de la forme $x^2 + bx - c$ où x, b, et c sont des entiers non négatifs et où pour un nombre $n = x^2 + bx - c$, on chercherait des valeurs x et b (cela suffit!) qui permettent la factorisation de n. Si nous voulons un algorithme capable de séparer tout entier positif composé, sans exception aucune, et que nous insistons pour préserver les conditions sur x, b, et c, nous ne pouvons alors considérer des valeurs x autres que $\lfloor \sqrt{n} \rfloor$ car, comme nous l'avons remarqué, pour des entiers de la forme $n = p^2$, où p est premier, la seule solution (non triviale) possible nécessite $x = \lfloor \sqrt{n} \rfloor$. Il suffit alors de parcourir l'ensemble des valeurs b ($b = 0, 1, 2, \ldots$) et de tester, à l'aide d'un test approprié, si l'on a une solution. Rappelons ici que les racines d'une équation quadratique générale $ax^2 + bx + c = 0$ sont données par la formule usuelle $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, qui dans le contexte se traduit par $x = \dfrac{-b \pm \sqrt{b^2 + 4c}}{2}$, puisque notre expression a la forme $x^2 + bx - c$. Notre test consiste

alors à vérifier, pour chaque b essayé, et chaque c calculé en fonction de b, x et n, si $b^2 + 4c$ est un carré; auquel cas on peut calculer les facteurs p et q. Voici l'algorithme :

Algorithme *split_quad* (n, B)

Entrée : un nombre n à factoriser, et une borne B (arbitraire), qui limite le nombre d'itérations. Sauf n et B, toutes les variables sont locales.

- Calculer $x = \lfloor \sqrt{n} \rfloor$.
- Pour b allant de 0 à B faire
  - Calculer $c := (x^2 + bx) - n$
  - Calculer $d := b^2 + 4c$
  - Vérifier si d est un carré
  - Si la condition est vérifiée alors
    - calculer $p := x - \left(\frac{-b + \sqrt{d}}{2} \right)$
    - calculer $q := \frac{n}{p}$ puis terminer la boucle
  - sinon boucler sur b
- Retourner $[[b], p, q]$ et terminer

Sortie : le nombre d'itérations effectuées $[b]$, et les facteurs p, q, quand trouvés

Procédure *split_quad* (n, B) (voir **Annexe C**)

Comparons maintenant cette procédure à *Fermat*(n, B), définie dans la section 2.1; à cette fin, regardons en premier quelques applications des deux procédures sur des nombres typiques : (voir ici **Annexe D**, 2.3 Exemples 1 à 4).

Il ressort de ces exemples et de nos observations, que la procédure *split_quad* est, en fait, une extension de la procédure *Fermat*, en ce sens qu'elle en élargit la portée, puisqu'elle parvient à factoriser tout entier positif composé alors que la procédure *Fermat* nécessite l'existence de deux facteurs de même parité. Cet élargissement se fait toutefois à un coût, puisque, comme nous l'avons noté (dans nos exemples), quand les solutions coïncident, la procédure *Fermat* y parvient deux fois plus vite. Il est simple de vérifier algébriquement l'équivalence totale des deux algorithmes, si l'on ne considère que les valeurs paires de la variable b dans la procédure *split_quad*. Il est aussi intéressant de rappeler ici (voir section 2.2), que si un nombre n ne peut s'exprimer comme produit de deux facteurs de même parité, il est toujours possible de le factoriser par la méthode de Fermat si, au préalable, on le multiplie par quatre (4), puis que l'on utilise la fonction pgcd sur les facteurs trouvés; mais nous avons déjà remarqué que cette opération avait pour conséquence de doubler le nombre d'itérations nécessaires pour arriver à une solution, si le nombre à factoriser peut déjà s'exprimer comme produit de deux facteurs de même parité.

Mentionnons que l'approche que nous avons présentée ici peut s'étendre à des expressions quadratiques de forme générale, i.e. $n = ax^2 + bx + c$, en commençant par diviser le nombre donné par une valeur entière arbitraire (on essaie!) pour a (inconnu!), puis de procéder de manière (presque) similaire. On obtient ainsi une méthode quasi-équivalente à celle de Lehman. Son développement étant fastidieux et sans réel apport nouveau, nous ne le présentons donc pas. Notre élargissement de concepts se tourne plutôt vers des expressions de la forme $x^k \pm y^k$, ce qui nous mène à une généralisation du critère de factorisation démontrée dans la section 1.3. Il en découle un algorithme de factorisation pour des nombres de cette forme.

- **Généralisation du critère de factorisation**

Le critère de factorisation énoncé en 1.3 précise, pour des nombres exprimables comme différence de carrés, ceux qui sont *séparables* par l'application de la fonction *split* définie en 1.1. Rappelons que nous avons démontré que, quand $n = x^2 - y^2$, alors *split* retourne

les facteurs $(x+y)(x-y)$ si et seulement si $y < \sqrt{2x-1}$. Nous voyons ici comment ce critère se généralise à des formes $n = x^k \pm y^k$. Nous présentons aussi un algorithme (et une procédure) qui prolonge, dans une nouvelle direction encore, celui de Fermat.

Racine carrée, racine k-ième, à plus ou moins près

2.3.1 Lemme

Soit *n* un entier positif tel que $n = x^k - y^k$, pour des entiers $0 \le y < x$, et $k \ge 2$.

Alors, les relations $\left\lceil \sqrt[k]{n} \right\rceil = x$ et $y < \sqrt[k]{x^k - (x-1)^k}$ sont équivalentes.

Démonstration

$$\left\lceil \sqrt[k]{n} \right\rceil = x$$

$$\Leftrightarrow \quad x - 1 < \sqrt[k]{n} \le x$$

$$\Leftrightarrow \quad x - 1 < \sqrt[k]{x^k - y^k} \le x$$

$$\Leftrightarrow \quad (x-1)^k < x^k - y^k \le x^k$$

$$\Leftrightarrow \quad y^k < x^k - (x-1)^k \le x^k$$

$$\Leftrightarrow \quad y < \sqrt[k]{x^k - (x-1)^k}$$

C.Q.F.D.

On remarque ici que si $k = 2$, alors l'équivalence se traduit par $\left\lceil \sqrt{n} \right\rceil = x \Leftrightarrow y < \sqrt{x^2 - (x-1)^2} = \sqrt{x^2 - (x^2 - 2x + 1)} = \sqrt{2x-1}$, ce qui correspond au résultat antérieur. L'équivalence établie permet aussi de constater que, pour des entiers

exprimables sous la forme $n = x^k - y^k$, la valeur y peut, tout en conservant sa relation $y < \sqrt[k]{x^k - (x-1)^k}$, être de plus en plus proche de x à mesure que k est grand.

Ce lemme nous amène à énoncer un critère de factorisation pour ce type d' expressions. Rappelons d'abord que la forme polynomiale $x^k - y^k$ peut toujours se factoriser comme $(x-y)\cdot\frac{x^k - y^k}{(x-y)}$. On vise ici, à trouver le premier des deux facteurs, l'autre s'obtenant par division.

Un critère de factorisation pour des entiers de la forme : $n = x^k - y^k$

2.3.2 Proposition

Soit n un entier positif tel que $n = x^k - y^k$ pour des entiers $0 \le y < x$, et $k \ge 2$.

Soit $X = \lceil \sqrt[k]{n} \rceil$, et $Y = \sqrt[k]{X^k - n}$

Si $y < \sqrt[k]{x^k - (x-1)^k}$ alors $X - Y$ est un facteur de n.

De plus, si k est un nombre pair, alors $\sqrt{X^k} - \sqrt{X^k - n}$ est aussi un facteur de n.

Démonstration

Si $y < \sqrt[k]{x^k - (x-1)^k}$ alors, par le lemme, $X = \lceil \sqrt[k]{n} \rceil = x$. Le résultat suit par le fait que $y = \sqrt[k]{x^k - n}$.

De plus, si k est pair, alors $n = x^k - y^k = (x^{k/2} - y^{k/2})(x^{k/2} + y^{k/2})$.

C.Q.F.D.

Nous remarquons ici que, comme pour les critères précédents, ce dernier est lui aussi indépendant de la taille des nombres. Nous notons également que si l'on réussit à trouver le facteur $(x-y)$, celui-ci est nécessairement de l'ordre de $\sqrt[k]{n}$. En termes de nombre de chiffres, si le nombre n s'écrit avec l chiffres, alors $(x-y)$ en a approximativement l/k. Ceci implique, par exemple $(k=3)$, que si un nombre $n = x^3 - y^3 = pq$, où p et q sont premiers et que $y < \sqrt[3]{x^3-(x-1)^3} = \sqrt[3]{3x^2-3x+1}$, il faut alors qu'un des facteurs ait (environ) deux fois plus de chiffres que l'autre. Inversement, si p et q sont premiers et de même taille, alors $n = pq$ ne peut s'écrire (par exemple) comme différence de deux cubes $n = x^3 - y^3$ avec $y < \sqrt[3]{3x^2-3x+1}$. Mais alors, si un nombre est le produit de deux nombres premiers de même taille, il ne peut en aucun cas s'écrire comme différence de deux cubes. En effet, supposons p et q premiers et de *même taille*, si $n = x^3 - y^3 = pq$ et $y < \sqrt[3]{3x^2-3x+1}$, il faut alors que le facteur $(x-y)$ ait environ un tiers des chiffres de n, et si $y \geq \sqrt[3]{3x^2-3x+1}$ alors le facteur $(x-y)$ est encore plus petit, en contradiction de l'hypothèse. Ces remarques se transportent naturellement à toute valeur $k > 2$; sauf pour $k = 2$, cette propriété empêche l'utilisation de cette méthode, pour les situations où les nombres considérés sont produits de deux nombres premiers de même taille, comme en cryptologie actuellement. Le critère énoncé conserve néanmoins sa valeur, dans le contexte général du problème de la factorisation. Nous en élargissons d'ailleurs, un peu encore, la portée.

Regardons le cas où $n = x^k + y^k$. Il est accessoire, mais il nous permet de souligner que la relation d'équivalence, établie dans le lemme 2.3.1, est en fait valide à *plus ou moins près*. Notons d'abord que si $k > 1$ est impair, on a toujours une factorisation avec $x^k + y^k = (x+y)(x^{k-1} - x^{k-2}y + \ \ldots\ - xy^{k-2} + y^{k-1})$. Si k est pair et contient au moins un facteur impair, on peut alors écrire $k = 2^i l$, où $l > 1$ est impair, et $i \geq 1$. Posons $a = 2^i$; dans ce cas, $x^k + y^k = (x^a)^l + (y^a)^l$ avec l impair, qui se ramène au cas précédent. Alors, $(x^a + y^a)$ est un facteur de $x^k + y^k$, le second facteur pouvant se calculer par division. Nous exclurons (du critère de factorisation) le cas où les valeurs k

sont des puissances de 2, puisque $x^2 + y^2$ ne peut pas être factorisé avec des coefficients entiers (ni même réels), et que si $k = 2^i$, avec $i \geq 1$, on a alors $x^k + y^k = \left(x^{k/2}\right)^2 + \left(y^{k/2}\right)^2$, se ramenant à une somme de carrés, qui ne peut se factoriser sous cette forme. Similairement au lemme précédent, nous démontrons d'abord:

2.3.3 Lemme

Soit n un entier positif tel que $n = x^k + y^k$ pour des entiers $0 \leq y < x$, et $k = 2^i l$ où $l > 1$ est impair et $i \geq 0$ est un entier quelconque.

Alors les relations $\left\lfloor \sqrt[k]{n} \right\rfloor = x$ et $y < \sqrt[k]{(x+1)^k - x^k}$ sont équivalentes.

Démonstration

$$\left\lfloor \sqrt[k]{n} \right\rfloor = x$$

$$\Leftrightarrow \quad x \leq \sqrt[k]{n} < x+1$$

$$\Leftrightarrow \quad x \leq \sqrt[k]{x^k + y^k} < x+1$$

$$\Leftrightarrow \quad x^k \leq x^k + y^k < (x+1)^k$$

$$\Leftrightarrow \quad 0 \leq y^k < (x+1)^k - x^k$$

$$\Leftrightarrow \quad y < \sqrt[k]{(x+1)^k - x^k}$$

C.Q.F.D.

Un critère de factorisation pour des entiers de la forme : $n = x^k + y^k$

2.3.4 Proposition

Soit n un entier positif tel que $n = x^k + y^k$, pour des entiers $0 \le y < x$ et $k = 2^i l$, où $l > 1$ est impair et $i \ge 0$ est un entier quelconque.

Soit $X = \left\lfloor \sqrt[k]{n} \right\rfloor$, $Y = \sqrt[k]{n - X^k}$, et $a = 2^i$.

Si $y < \sqrt[k]{(x+1)^k - x^k}$ alors $X^a + Y^a$ est un facteur de n,

Démonstration

Si $y < \sqrt[k]{(x+1)^k - x^k}$ alors, par le lemme, $X = \left\lfloor \sqrt[k]{n} \right\rfloor = x$. Le résultat suit par le fait que $y = \sqrt[k]{n - x^k}$.

C.Q.F.D.

Nous complétons cette section en présentant ci-après un algorithme et une procédure (voir **Annexe C**) *Fermat_kieme* qui met en pratique les résultats démontrés, et en fait les prolonge par une approche similaire à celle utilisée pour l'algorithme *Fermat_Lehman*. Ici toutefois, la recherche en *largeur* signifie de parcourir un ensemble de valeurs k qui sont des *exposants* plutôt que des multiplicateurs; la recherche en *profondeur* a le même sens que précédemment, i.e., parcourir, à l'aide d'une variable, un ensemble de valeurs x partant d'une valeur initiale $x_0 = \left\lfloor \sqrt[k]{n} \right\rfloor$, et en effectuant à chaque itération un test de validité. Ce test consiste ici à vérifier si une certaine valeur numérique, obtenue d'une expression, est une valeur entière. Nous avons opté, ici aussi, de paramétriser l'algorithme, en passant en entrée, en plus du nombre n à factoriser, deux paramètres K et B, qui délimitent chacun un intervalle de recherche. Cette approche permet, comme nous l'avons déjà remarqué, d'explorer diverses situations et ainsi de mieux apprécier les capacités et limitations de l'algorithme. Nous incluons dans celui-ci une composante qui traite la recherche d'une solution pour les nombres de la forme $n = x^k + y^k$, au coût d'un d'un code alourdi et d'une exécution ralentie; elle peut certes en être exclue, mais au coût évident de ne pas pouvoir reconnaître ces solutions quand elles existent.

Remarquons aussi, que la stratégie de recherche utilisée dans l'algorithme consiste à parcourir en profondeur, pour chaque exposant. Cet ordre peut être inversé et produire, selon le contexte, une meilleure ou une pire performance. Toutefois, nous notons que comme pour l'algorithme *Fermat_Lehman*, quelle que soit la stratégie utilisée et en fonction des paramètres donnés pour une exécution, il existe une classe (identifiable) de nombres, sans limite de taille, que l'algorithme peut factoriser efficacement; d'où notre appellation : *efficace, un peu partout*!

Algorithme *Fermat_kième* (n, K, B)

Entrée : un nombre n à factoriser, et
un intervalle de valeurs précisé par le paramètre $K = [K_{\min}, K_{\max}]$ pour les exposants, et un intervalle $B = [B_{\min}, B_{\max}]$ qui délimite la recherche des valeurs x.

- Pour k variant de $K_{\min}$ à $K_{\max}$ faire
  - Calculer $x = \left\lfloor \sqrt[k]{n} \right\rfloor$
  - Pour b variant de $B_{\min}$ à $B_{\max}$
    - Calculer $y = \sqrt[k]{(x+b)^k - n}$
    - Vérifier si y et $\dfrac{n}{x+b-y}$ sont des entiers
    - Si la condition est vérifiée alors
      - poser $p = x+b-y$, $q = \dfrac{n}{p}$,
      - retourner $[[k,b],[p,q]]$ et terminer
    - Sinon calculer $y = \sqrt[k]{n-(x-b)^k}$
    - Vérifier si y et $\dfrac{n}{x-b+y}$ sont des entiers
    - Si la condition est vérifiée alors
      - Poser $p = x-b+y$, $q = \dfrac{n}{p}$

- retourner $[[k,b],[p,q]]$ et terminer
  - Boucler sur b
- Boucler sur k

- Terminer : retourne $[[k,b],[p,q]]$

Sortie : les valeurs (solutions) $[k,b]$ à titre d'information et les facteurs p et q, si trouvés.

Nous mettons en **Annexe C**, une procédure *Fermat_kieme*(n, K, B) qui met en pratique cet algorithme. Nous mettons aussi en **Annexe D** une série d'exemples (2.3 -exemple(s) 5), petits et grands, indicateurs de la performance de l'algorithme dans diverses situations. Nous voyons, au chapitre suivant, comment appliquer certaines de ces idées à la cryptologie.

Chapitre III

Applications en cryptologie

Notre but, dans ce chapitre, est de montrer comment certains des concepts et résultats établis dans les sections précédentes pourraient être utilisés dans des contextes de nature cryptologique. Dans un premier temps nous élaborons, à titre illustratif, un cryptosystème probabiliste à *clef privée* basé sur les concepts discutés, que nous décrivons et analysons sommairement. Dans la seconde partie du chapitre nous visons à mettre en évidence des situations (hypothétiques) où l'utilisation de ces concepts, ou des algorithmes de factorisation associés, pourrait mettre à risque des cryptosystèmes dont la sécurité (calculatoire) est basée sur le problème de factorisation. À cet effet, notre attention porte essentiellement sur le système à *clef publique* RSA [14], qui, par son lien étroit avec le problème de factorisation et par son utilisation aujourd'hui très courante, s'avère être un exemple sur mesure. Nous rappelons, pour fin de référence immédiate, les principales caractéristiques de ce système. Pour toute précision additionnelle, nous en référons à [12] ou [17].

3.1 Application à un cryptosystème probabiliste

L'objectif de cette section est d'illustrer l'application d'idées déjà vues, à l'élaboration d'un cryptosystème probabiliste à *clef privée*, sans prétention aucune. Celui-ci présente toutefois quelques caractéristiques intéressantes dont nous discutons brièvement. Nous n'avons pas cherché à analyser la complexité des algorithmes de chiffrement et de déchiffrement, nous contentant de constater, par l'exécution des procédures associées, qu'elles se réalisaient efficacement et sans difficulté sur une machine de très modeste capacité. La sécurité du système, nous le verrons, est liée au problème de factorisation, mais augmentée par un facteur d'incertitude. Toutefois, ici non plus, nous n'avons pas cherché à pousser notre analyse au delà de quelques observations, et nous ne présentons donc pas de mesure formelle de complexité.

Avant de décrire les algorithmes en question, rappelons brièvement que *cryptosystème à clef privée* sous-entend que ses utilisateurs doivent pouvoir échanger, au préalable, des clefs de chiffrement et de déchiffrement, à travers un canal de communication entièrement sécurisé. Il est habituel, dans le contexte, d'utiliser des clefs symétriques, la même clef servant à la fois au processus de chiffrement et à celui de déchiffrement. Nous distinguons, comme de coutume, les fonctions d'*encodage* et de *chiffrement*, la première servant à *encoder* un texte clair, en l'associant à un nombre, qui est ensuite *chiffré* par la seconde. Il en est de même, dans le sens inverse, pour les fonctions de *déchiffrement* et de *décodage*.

Description du cryptosystème

1. Encodage des messages

Pour nos essais, nous avons developpé une fonction effectuant un encodage probabiliste, en ce sens que plusieurs nombres peuvent correspondre à un message donné, selon un tableau de correspondance pré-établi pour l'encodage de chaque caractère. Dans notre implantation, chaque caractère est remplacé par un nombre de deux chiffres choisi au hasard (uniformément) dans une liste; il est, bien sûr, possible d'utiliser plus de chiffres pour la substitution de chaque caractère et ainsi d'augmenter l'incertitude, mais au coût évident d'un gonflement proportionnel du message de sortie. Nous ne présentons pas cette procédure, celle-ci étant basée sur des principes simples et connus. Remarquons que si le processus d'encryption s'arrêtait ici, la sécurité du système serait à risque, puisque ce type de chiffrement peut être cassé par cryptanalyse statistique ([17], sect. 1.2). Soulignons toutefois que l'incertitude conséquente à cette étape est, par la suite, amplifiée par la fonction de chiffrement.

2. Le processus de chiffrement

Il requiert en entrée un nombre, le *message* original maintenant encodé en n, et une *clef* secrète. Dans notre cas, celle-ci se compose d'une liste contenant plusieurs clefs (nombres) que nous nous appelons $k, a, b, \ldots, g$. Les clefs $a, b, \ldots, f$ sont constituées de nombres premiers, de tailles quelconques, préférablement grands. La clef k est un

paramètre, entier aussi, qui sert à définir la taille de nombres utilisés et pourrait être inscrit directement dans le code, au coût d'une diminution de souplesse. Son rôle dans la sécurité du système est toutefois négligeable si l'on ne considère que sa complexité, soit celle d'un entier de petite taille. La clef g est indépendante des autres composantes et pourrait être entièrement omise. Notons par contre que sa présence va permettre d'ajouter une couche significative de sécurité à l'ensemble. Avec des ajustements mineurs elle peut prendre diverses formes, soit comme nombre entier, rationnel ou réel, ou même comme instance d'une fonction quelconque. Dans notre exemple, nous l'utilisons comme nombre entier.

Les fonctions utilisées

- Nous nous servons d'une fonction génératrice de nombres pseudo-aléatoires que nous appelons *Rnd* et qui, pour k donné, retourne un entier compris dans l'intervalle $10^{k-1},\ldots, 10^{k}$. Dans notre implantation, nous avons utilisé une fonction dérivée de celle native au logiciel Maple, sans autre considération.
- Nous nous servons d'une fonction qui retourne, pour un entier donné, un nombre premier qui lui est proche. À cette fin, nous utilisons la fonction *nextprime*, déjà mentionnée.
- La fonction *length* retourne le nombre de chiffres (ici en décimal) contenus dans un nombre donné.
- La fonction $\lfloor x \rfloor$, a son sens habituel d'arrondir par le bas un nombre x donné.

L'algorithme de chiffrement

- Étant donné le message codé n et la clef k calculer $M = Rnd(k)\cdot 10^{k+\,length(n)} + 10^{k}\cdot n + Rnd(k)$

  Remarque: Cette étape consiste à *saler* le message n en lui adjoignant, au début et à la fin, k chiffres aléatoires. Ainsi, n est enrobé et transformé en M. Le lien entre les deux se situe au niveau des chiffres centraux de M, et ne concerne donc pas les chiffres significatifs de ce dernier.

- Avec les clefs a, b, e, f et k former un nombre premier p tel que $p = nextprime\left(\left\lfloor \frac{a}{b} \cdot \frac{e}{f} \cdot M \right\rfloor + Rnd(k-3)\right)$.

  Remarque: p est dérivé de M à l'aide des clefs a, b, e, f, mais l'ajout de la quantité aléatoire $Rnd(k-3)$, dans le calcul de p, en augmente l'incertitude. La soustraction de la quantité 3 (dans l'expression $k-3$) est accessoire, mais elle a pour effet de simplifier et ainsi d'accélérer le processus de déchiffrement.

- Similairement, avec les clefs c, d, e, f et k former un autre nombre premier q tel que $q = nextprime\left(\left\lfloor \frac{c}{d} \cdot \frac{e}{f} \cdot M \right\rfloor + Rnd(k-3)\right)$.
- Le message chiffré est $N = pq + g$.

3. Le processus de déchiffrement

Les fonction utilisées

- Une fonction $CutL(n,k)$ qui tronque les k premiers chiffres d'un nombre n donné.
- Une fonction $CutR(n,k)$ qui tronque les k derniers chiffres d'un nombre n donné.
- Les fonctions pgcd(n, m) et $\lceil x \rceil$ avec leur sens habituel.

L'algorithme de déchiffrement

- Étant donné le message chiffré N et la clef g calculer $pq = N - g$.
- Calculer $m = pq \cdot a \cdot b \cdot c \cdot d$
- Calculer $x = \lceil \sqrt{m} \rceil$
- Retrouver $p = \text{pgcd}(x + \sqrt{x^2 - m},\ pq)$ puis $q = \frac{n}{p}$

  Remarque: Ici, on applique en fait d'abord sur m la fonction *split*, définie au premier chapitre; celle-ci va toujours résulter en succès car les clefs a, b, c et d

s'associent avec p et q pour former un nombre m composé de deux facteurs relativement proche l'un de l'autre. La quantité $Rnd(k-3)$ n'étant ici pas assez significative, une recherche en profondeur n'est pas nécessaire.

- Calculer $M_1 = \left\lceil p \cdot \frac{b}{a} \cdot \frac{f}{e} \right\rceil$, $M_2 = \left\lceil p \cdot \frac{d}{c} \cdot \frac{f}{e} \right\rceil$, $M_3 = \left\lceil q \cdot \frac{b}{a} \cdot \frac{f}{e} \right\rceil$, et $M_4 = \left\lceil q \cdot \frac{d}{c} \cdot \frac{f}{e} \right\rceil$

Remarque: Des quatre valeurs M_i ($i = 1,...,4$) deux seront proches de M. On en retrouve une parmi M_1 et M_2, l'autre parmi M_3 et M_4.

- Calculer $m_i = CutL(CutR(M_i, k), k)$ pour $i = 1,...,4$

Remarque: Ici on tronque les k chiffres à droite et à gauche de chaque M_i. Le message encodé n se retrouve alors deux fois parmi les quatre m_i, les deux autres quantités étant différentes. Il suffit alors d'effectuer un test pour l'identifier.

- Si $m_1 = m_3$ ou $m_1 = m_4$ alors $n = m_1$, sinon $n = m_2$

4. Décodage des messages

Il s'effectue, comme pour l'encodage, à l'aide d'un tableau de correspondance, mais ici sans la dimension aléatoire. Il suffit de parcourir les chiffres de n, de gauche à droite et deux à deux, puis d'effectuer la substitution inverse.

Analyse sommaire du cryptosystème et de sa sécurité

Il est aisé de constater, à partir des procédures d'encodage et de chiffrement, que le système a quatre couches de sécurité, dans l'ordre suivant:

1. La clef g qui barre l'accès à pq.

2. La factorisation de pq, un nombre *dur*, mais factorisable efficacement avec les clefs a, b, c et d.
3. Les clefs e et f qui permettent de retrouver M (ou un proche), et la clef k pour déterminer ensuite n.
4. Le décodage de n, qui offre un minimum de sécurité comme nous l'avons déjà remarqué.

Notons que la sécurité globale du système va dépendre, dans une très large mesure, de la complexité des clefs utilisées, celles-ci étant adaptables à divers contextes. Si nous partons du message chiffré pour essayer de remonter au *message* original, en connaissance de tout le processus, mais non des clefs évidemment, celui-ci se présente comme un nombre aléatoire $N = pq + g$ quelconque. N est de taille plus grande que le message encodé n, le degré de gonflement pouvant se contrôler avec la clef k. Cette clef, comme nous l'avons remarqué, détermine le nombre de chiffres aléatoires qu'on adjoint au début et à la fin de n pour le changer en M. Il faut donc nécessairement commencer le processus de déchiffrement par le retranchement de la clef g, n étant inaccessible autrement. Cette clef g pouvant avoir la complexité voulue, nous avons utilisé, dans notre implantation, un g suffisamment grand pour ensuite le tronquer, au besoin et dynamiquement, afin qu'il soit de la taille de pq (avec deux ou trois chiffres de moins pour éviter les débordements). Si l'on retire cette première couche de sécurité, le problème devient, en première ligne, celui de la factorisation d'un nombre *dur*. Le degré de *dureté* dépend alors de la taille du message encodé n, et du nombre de chiffres ajoutés, i.e. $4k$, puisque la taille de chacun des deux nombres premiers égale celle de n plus $2k$. Notons ici que, de par leur construction, ces paires de nombres premiers sont dans une très grande mesure aléatoires, et leur rapport est indécelable sans la connaissance effective de leur valeur individuelle, i.e. il faut donc résoudre le problème de factorisation. On note ici que le fait d'avoir, pour un *message* donné, un pq résultant qui est essentiellement aléatoire dans un vaste (et contrôlable) espace, ajoute à la complexité de retracer la clef g. Si l'on parvient à franchir la deuxième couche de sécurité, en retrouvant les facteurs p et q, il faut encore arriver à calculer n, qui nécéssite la connaissance des clefs e et f (et k), qui ne peuvent être retracées à partir de p et q. En

effet, en effectuant le rapport $\frac{p}{q}$ on obtient un nombre relativement proche du rapport véritable $\frac{a \cdot d}{b \cdot c}$. Ce nombre, notons le, pourrait permettre la factorisation de toute autre paire générée par les même clefs, à l'aide d'approximations successives pour retrouver $a \cdot b \cdot c \cdot d$. Mais remarquons maintenant, qu'en effectuant le rapport $\frac{p}{q}$, la quantité $\frac{e}{f}$ est éliminée, étant présente dans les deux facteurs. À ce niveau, le lien demeurant entre p et q se situe donc par rapport à tierce quantité M, inconnue. Or, $\frac{e}{f}$ est essentiel pour remonter à M, puis à n par tronquation, en connaissance de la clef k. Si l'on parvient à retrouver n, alors la confidentialité du message est essentiellement brisée.

Supposons maintenant qu'un adversaire ait accès (illimité) à une machine à la fois chiffrante et déchiffrante, mais que les clefs (qui lui sont inconnues) soient intégrées au code, qui lui est entièrement sécurisé. Cet adversaire pourrait-il retrouver les clefs ? Supposons d'abord que celui-ci utilise plusieurs fois, un même *message*, composé d'au moins un caractère. Si la clef k est suffisamment grande (par exemple 100 ou plus), alors le message chiffré N résultant de chaque exécution aura une taille d'au moins 200 chiffres, et sera (quasi-sûrement) différent d'une instance à l'autre. Sans la connaissance de la clef g, on ne peut retrouver pq, et ici une approximation ne suffit pas. Si l'on devait essayer systématiquement toutes les valeurs possibles pour g, encore faudrait-il savoir où arrêter; à chaque étape le problème devient de déterminer d'abord si le nombre obtenu est un *dur*, et de plus, s'il en était, rien n'indique que c'est celui que l'on cherche. Il faut pouvoir se rendre au bout du processus de déchiffrement et de décodage pour vérifier si le g essayé est effectivement une des clefs. À ce niveau, l'incertitude nous paraît plus grande que la complexité effective de g. Vient s'ajouter ensuite le problème de factorisation. En supposant qu'une valeur g nous mène à un *dur* reconnaissable (comment le savoir sinon en cherchant à le factoriser), on se retrouve alors en fait confronté à une multitude d'instances du problème de factorisation, une seule débouchant sur la solution. Notre analyse, certes non-exhaustive, n'a pas permis de voir comment contourner cet obstacle. Advenant la factorisation (difficile) d'un de ces *durs*, il est

possible de retrouver, comme on l'a déjà noté, la clef de factorisation $a \cdot b \cdot c \cdot d$, mais à condition d'avoir le bon *dur*, et là aussi, pour le savoir, il faut pouvoir aller au bout du processus. Supposons maintenant que l'adversaire arrive, par un quelconque moyen, a obtenir les clefs a,b,c,d et g. La clef k, elle, peut facilement se trouver en observant la taille de N. Reste alors les clefs e et f, indépendantes des autres. Ici, même si l'on dispose de multiples paires p, q toutes associées au même message, elles ne permettent pas de retracer les clefs e et f, puisque, dans chaque cas, le lien est par rapport à une tierce quantité qui est changeante. Si nous supposons maintenant que l'adversaire varie ses *messages*, il nous semble plus improbable encore qu'il puisse retracer un lien quelconque entre les instances, l'espace de textes chiffrés étant très vaste. Les caractéristiques observées de ce cryptosystème suggèrent une réutilisation possible d'un même ensemble de clefs, sans diminution significative de sécurité, au besoin au coût d'une légère modification de l'algorithme et d'une permutation des clefs pour en augmenter l'incertitude. Une réutilisation de clefs, même partielle ou temporaire, peut constituer un avantage important dans certaines situations. Le but de l'exercice était toutefois ici, rappelons-le, simplement de démontrer une application possible des concepts étudiés. Les procédures développées et utilisées pour nos expériences se trouvent en **Annexe C** (procédures *Chiffre* et *Dechiffre*). Un petit exemple d'application du procédé, sur un court message, est également fourni à l'**Annexe D** (3.1 – Exemple 1).

3.2 La cryptographie à clef publique: quelques critères à préciser

La cryptographie dite à *clef publique* est largement pratiquée dans le contexte des communications électroniques d'aujourd'hui. Ses origines sont pourtant récentes; l'idée fut introduite par Diffie et Hellman en 1976 [8]. Contrairement aux systèmes classiques, elle permet à deux individus de pouvoir communiquer secrètement, sans échange préalable de clef(s). Le premier système de cryptographie à clef publique (RSA) fut inventé en 1978 par Rivest, Shamir et Adleman [14]. Sa sécurité est essentiellement basée sur la difficulté de factoriser de grands nombres. Afin de décrire le fonctionnement de RSA, nous introduisons le, maintenant traditionnel, trio de personnages: *Bob* et *Alice*, deux correspondants désireux d'échanger confidentiellement des messages, et l'intrus *Oscar*. Pour un court rappel de quelques définitions et théorèmes pertinents de la théorie des nombres voir l'**Annexe A**; pour le processus de chiffrement et de déchiffrement RSA, voir l'**Annexe E**.

Le problème RSA

On remarque que les fonctions de chiffrement et de déchiffrement, respectivement $y = x^b \bmod n$ et $x = y^a \bmod n$, sont réciproques l'une de l'autre. En effet, si $n = pq$ où p et q sont deux nombres premiers distincts, et si a et b sont tels que $ab \equiv 1 \mod \phi(n)$, alors on a $ab = k\phi(n) + 1$ pour un entier k. Si $x \in \mathbf{Z}_n$ avec $\text{pgcd}(x, n) = 1$ alors $y^a \equiv (x^b)^a \equiv x^{k\phi(n)+1} \bmod n \equiv (x^{\phi(n)})^k x \bmod n \equiv 1^k x \bmod n$, car $x^{\phi(n)} \equiv 1$ par le théorème d'Euler (voir **Annexe A**), d'où $y^a \equiv x \bmod n$. Mentionnons aussi que la fonction de chiffrement $x \to x^b \bmod n$ est une fonction injective dite à *sens unique*, c'est-à-dire calculatoirement difficile à inverser sans la connaissance d'une *trappe* cachée, ici la factorisation de n, tenue secrète par *Bob*. Voyons maintenant quelques situations illustrant comment *Oscar* pourrait réussir à briser la confidentialité d'un message x, chiffré en y selon le protocole RSA, et transmis par *Alice* à l'intention de *Bob* sur un canal peu sécuritaire, si certains critères ne sont pas précisés.

Oscar peut, on le suppose, obtenir facilement une copie du message chiffré y puisque celui-ci est transmis sur un canal peu sûr; pour lui, la difficulté n'est pas là. Il est familier avec le processus (RSA) de chiffrement et de déchiffrement, c'est le *principe de Kerckhoff*, utilisé en cryptanalyse ([17], p.22). Il peut également se renseigner sans difficulté sur la valeur des clefs n et b, celles-ci étant diffusées dans un répertoire public. Il ne sait toutefois pas comment retrouver x efficacement sans connaître a, la clef secrète de *Bob*, et pour calculer a il lui faudrait connaître $\phi(n)$ qui est aussi difficile à calculer que de factoriser n, ce qu'il ne sait pas faire en temps efficace si n est *dur*, à moins qu'il ne dispose d'un oracle ou d'un ordinateur quantique [16]. La connaissance de $\phi(n)$ est, bien sûr, suffisante pour casser le système puisque, si *Oscar* connaît $\phi(n)$, il peut, tout comme *Bob*, se servir de l'algorithme étendu d'Euclide pour calculer a, l'inverse de $b \bmod \phi(n)$. En fait, comme nous l'avons déjà mentionné, la connaissance de $\phi(n)$ permet de factoriser n, puisque dans le cas présent $n = pq$, le produit de deux nombres premiers, alors $\phi(n) = (p-1)(q-1)$, qui est équivalent à $p^2 - (n - \phi(n) + 1)p + n = 0$, une équation du second degré en p, qui se résout algébriquement. Formellement, le problème RSA consiste ([12], def. 3.28) à trouver un entier x (le message d'Alice encodé) tel que $x^b \equiv y \bmod n$, connaissant n, y, et b et sachant que $\text{pgcd}(b, \phi(n)) = 1$, un problème aujourd'hui encore considéré calculatoirement difficile. Il est clair toutefois qu'Oscar peut essayer diverses tactiques pour tenter d'atteindre son objectif. Il pourrait, par exemple, chercher à retrouver a, la clef secrète de Bob, sans nécessairement passer par la factorisation de n. Ce calcul est possible, en temps efficace, dans certaines situations spécifiques, par exemple ([12], 8.2.2 ii et iv), mais, de manière générale, il a été établi ([12], fact 8.6) que le problème de trouver l'exposant secret a est calculatoirement équivalent au problème de factorisation. Il est simple de constater que le problème RSA se réduit au problème de factorisation puisque la solution de ce dernier permet de retrouver la clef secrète a. La conjecture actuelle est que le problème RSA est, en fait, calculatoirement équivalent au problème de factorisation ([12], 3.20), mais cela reste à être démontré.

Attaques contre RSA

Si *Oscar* choisit de s'attaquer à RSA par le biais de la factorisation, il est clair que la sécurité va dépendre entièrement de la difficulté qu'il aura à factoriser le modulus n, avec la technologie (limitée) dont il dispose et de sa stratégie de recherche. La sécurité de RSA va donc dépendre grandement du choix des deux facteurs premiers de n, soit p et q ainsi que de la taille de n. Un modulus n de 512 bits, soit environ 154 chiffres en décimal, ne procure pas aujourd'hui suffisamment de sécurité devant une attaque concertée (*Oscar* a des alliés qui unissent leurs ressources pour l'aider à factoriser n) avec des algorithmes de factorisation tels le *crible quadratique* ou le *crible algébrique* ou encore l'algorithme sur les *courbes elliptiques*, qui sont de développement récent. Actuellement, il est recommandé d'utiliser un modulus n avec 768 bits (environ 231 chiffres en décimal), ou, mieux encore, 1024 bits (308 chiffres en décimal), pour contrer une attaque à l'aide de ces algorithmes. Mais, comme nous l'avons vu dans les chapitres précédents, il pourrait en dépendre également de la composition de n et de la stratégie utilisée pour essayer de le factoriser. Ainsi, par exemple, si $n = pq$ est tel que $\left|\sqrt{p} - \sqrt{q}\right| < \sqrt{2}$ et qu'*Oscar* choisit de commencer sa recherche des facteurs de n à l'aide de la fonction *split* définie en 1.1, il trouverait alors immédiatement la solution, indépendamment de la taille du modulus n, comme nous l'avons établi. Notons ici que ([12], note 8.8 (ii)) recommande de choisir p et q de sorte que la différence $p - q$ ne soit pas trop petite, puis mentionne que si $p \approx \sqrt{n}$, il est alors possible de factoriser efficacement n, à l'aide de l'algorithme de division (*classique*), en essayant tous les nombres impairs proches de $\sqrt{n}$. Remarquons que si n est tel que $\left|\sqrt{p} - \sqrt{q}\right| \cong \sqrt{2}$, par exemple, ce calcul nécessiterait un nombre de divisions de l'ordre de $n^{1/4}$. Si n (en décimal) contient plus de 300 chiffres, cela se traduit par plus de 10^{75} divisions, qui peut aujourd'hui difficilement s'effectuer en temps raisonnable. La fonction *split*, ou par extension l'algorithme de Fermat, permet une factorisation efficace pour tout nombre de cette classe. Notons par contre qu'il est mentionné dans la même référence, avec plus de justesse cette fois, que si p et q sont choisis uniformément au hasard, alors avec une très forte probabilité la différence $|p - q|$ sera suffisamment grande. Ainsi, si *Bob* choisit ses nombres premiers, p et q, selon un

processus (pseudo) aléatoire valable, son modulus n sera, fort probablement, calculatoirement sécure face à une attaque par la fonction *split* ou par l'algorithme de Fermat. Toutefois, le hasard permet l'exceptionnel; nous voulons alors souligner ici que, face à une telle attaque, le degré de sécurité va, en fait, dépendre entièrement de la distance effective $\left|\sqrt{p}-\sqrt{q}\right| = c$. Pour contrer une telle attaque, *Bob* devrait-il calculer c puis estimer si $\left\lceil \frac{c^2}{2} \right\rceil$ représente une valeur suffisamment grande, en nombre d'opérations, pour ne pas être à la portée de la technologie dont dispose *Oscar*?

L'importance de vérifier si la distance entre les facteurs p et q est suffisamment grande pour que RSA soit calculatoirement sécure est signalée dans la norme ANSI [ANSI X9.31] qui recommande une distance $|p-q| > n^{\frac{1}{2}-\frac{100}{\log_2 n}}$. Remarquons que si le modulus n a 1024 bits, cela se traduit par $|p-q| > n^{\frac{1}{2}-\frac{100}{1024}}$ soit environ $n^{2/5} \cong 10^{123}$, qui est certes suffisant pour contrer une attaque par l'algorithme de Fermat, surtout si celle-ci se fait à partir de $\sqrt{n}$. Mais notons ici que rien n'empêche *Oscar* de tenir compte de cette information, et de commencer sa recherche de $\frac{p+q}{2}$ à partir d'une valeur x calculée en fonction de cette distance minimale. Si la distance effective $|p-q|$ est suffisamment proche de la valeur minimale recommandée, ici $n^{\frac{1}{2}-\frac{100}{\log_2 n}}$, il devient alors possible de factoriser n efficacement. Pour effectuer le calcul de cette valeur de départ x, supposons que $q > p$ et remarquons que si $q - p > c$, où c est un seuil critique quelconque, alors $\frac{n}{p} - p > c$ puisque $n = pq$. Il en découle, comme nous l'avons déjà vu, une inéquation du second degré en p, soit $n - cp - p^2 > 0$ qui mène à une valeur maximale pour p, soit $p < \frac{-c+\sqrt{c^2+4n}}{2}$. Si nous considérons que $q = \frac{n}{p}$ il s'en suit que $p+q \geq \frac{-c+\sqrt{c^2+4n}}{2} + \frac{2n}{-c+\sqrt{c^2+4n}}$. En effet, comme nous l'avons noté dans la

section 2.1., si l'on observe les valeurs possibles pour p et q par rapport au pivot $\sqrt{n}$, n étant fixe, les valeurs q croissent plus rapidement que ne diminuent les valeurs p. Il suffit alors de commencer la recherche des facteurs, à l'aide de l'algorithme de Fermat paramétrisé, à partir de $x = \left\lfloor \frac{1}{2}\left(\frac{-c+\sqrt{c^2+4n}}{2} + \frac{2n}{-c+\sqrt{c^2+4n}} \right) \right\rfloor$. Il est clair que si la valeur effective de $\frac{p+q}{2}$ est calculatoirement proche de cette valeur initiale x (exceptionnellement sans doute), alors *Oscar* réussira à briser la sécurité de RSA, et ce, même si les facteurs sont conformes aux normes minimales recommandées par ANSI. Remarquons que si l'on prend $|p-q| > n^{2/5}$, pour un n d'environ 300 chiffres en décimal, la formule ci-dessus va produire une valeur $x > \sqrt{n}$ telle que $x - \sqrt{n} \cong 10^{140}$. L'importance de ce détail est précisée plus loin. Mais la question demeure : Faut-il que *Bob* s'assure que la distance $|p-q|$ soit significativement supérieure au seuil recommandé ? Les remarques qui précèdent suggèrent, certes, une réponse affirmative. Mais, il est clair aussi que l'imposition d'un seuil réduit de fait l'espace de recherche, tout en laissant à risque les cas frontières. Faudrait-il alors préciser qu'il vaudrait peut-être mieux ne pas être trop précis quant au seuil minimum recommandé ? Ces remarques peuvent tout autant s'appliquer au rapport effectif $\frac{q}{p}$. Comme nous l'avons noté dans la section 2.1, si celui-ci est trop proche d'une fraction $\frac{q'}{p'}$ telle que le produit $p'q'$ est relativement petit, il devient alors possible de factoriser efficacement n à l'aide d'un algorithme paramétrisé à la Fermat-Lehman, comme celui que nous avons présenté. Il est clair, dans ce cas aussi, que toute information précise, sur une valeur minimale ou sur un intervalle de valeurs possibles pour $p'q'$, pourrait être utilisée par ce même algorithme pour le diriger dans sa recherche des facteurs. Le cas échéant, la sécurité du cryptosystème est encore à risque pour les cas frontières. Rappelons maintenant, qu'advenant un échec de la procédure dans un temps d'exécution donné, il est possible d'en déduire des intervalles d'exclusion pour les facteurs recherchés. Cette information

peut être utile pour limiter une recherche ultérieure, à l'aide d'autres algorithmes de factorisation, ou par d'autres méthodes comme nous le constatons dans ce qui suit.

Attaques sur l'exposant secret, en fonction de la distance entre les facteurs

L'attaque que nous examinons maintenant porte sur l'exposant secret a, mais en relation à la distance effective $|p-q|$ entre les facteurs premiers de n. Cette question est analysée dans un article [7] récent. Dans cet article, l'auteur démontre que si cette distance est significativement inférieure à $n^{1/2}$, il devient alors possible d'améliorer l'efficacité des attaques connues de Wiener [18] et de Boneh-Durfee [1]. Ces attaques portent sur l'exposant a, quand celui-ci est relativement petit; plus précisément, quand $a < n^{1/4}$ dans le cas Weiner, et $a < n^{1-1/\sqrt{2}}$ dans le cas Boneh-Durfee Notons ici que la norme ANSI recommande d'avoir $a > n^{1/2}$; la situation examinée est donc hypothétique, mais *Bob* pourrait, sait-on jamais, être ignorant des subtilités du protocole, ou tout simplement négligent. Remarquons ici que si *Oscar* réussit à trouver l'exposant secret de *Bob*, il peut alors factoriser n efficacement à l'aide d'un algorithme tel que décrit dans ([12], 8.2.2 (i)).

Voyons d'abord comment de Weger réussit à améliorer le résultat de Wiener, sous l'hypothèse d'une certaine proximité entre les facteurs premiers de n. Nous verrons ensuite comment obtenir un résultat équivalent ou mieux, sous l'hypothèse d'avoir certaines informations concernant les facteurs de n, et dépendant de la qualité de cette information. Rappelons ici que $a \equiv b^{-1} \bmod \phi(n)$ entraîne l'existence d'une valeur k positive telle que $ab - k\phi(n) = 1$. En divisant les deux membres par $a\phi(n)$ on obtient une équation équivalente $\frac{b}{\phi(n)} - \frac{k}{a} = \frac{1}{a\phi(n)}$, qui indique une forte proximité entre les rapports $\frac{b}{\phi(n)}$ et $\frac{k}{a}$, étant donné que $\phi(n) \approx n$ (supposé grand). Cette proximité se traduit dans les faits, si l'on examine les listes des fractions convergentes vers chacun de

ces rapports, par la présence de $\frac{k}{a}$ dans celle de $\frac{b}{\phi(n)}$. Remarquons que ces convergentes se calculent efficacement en utilisant un développement en fraction continue ([13], chap. 7) des rapports en question. Pour son attaque, Wiener utilise le fait que $\phi(n) \approx n$ pour obtenir une approximation de $\frac{b}{\phi(n)}$ par $\frac{b}{n}$. Il démontre que si l'exposant secret de *Bob* est tel que $a < n^{1/4}$, alors cette approximation est suffisante pour que $\frac{k}{a}$ apparaisse dans la liste des fractions convergentes vers $\frac{b}{n}$. Remarquons ici que la connaissance du rapport exact $\frac{k}{a}$, donc de k et de a, permet de calculer $\phi(n)$ directement, et par suite de factoriser n. L'idée de de Weger est d'utiliser une meilleure approximation pour $\phi(n)$. Il suggère de prendre $\phi(n) \cong n+1-2\sqrt{n}$, et cette modification s'avère en fait être très significative lorsque la distance $|p-q|$ est relativement petite. Rappelons ici que $\phi(n) = (p-1)(q-1) = n+1-(p+q)$ et que l'on a toujours $p+q \geq 2\sqrt{n}$. Si p et q sont relativement proches, nous avons déjà vu que l'algorithme de Fermat permet de trouver efficacement les facteurs de n, ce qui revient à dire que $\frac{p+q}{2}$ est suffisamment proche de $\sqrt{n}$. Ainsi, plus la distance $|p-q|$ est petite, d'autant meilleure est l'approximation de $\phi(n)$ par $n+1-2\sqrt{n}$. En fait, comme nous l'avons démontré, si $|\sqrt{p}-\sqrt{q}| < \sqrt{2}$ alors $\lceil \sqrt{n} \rceil = \frac{p+q}{2}$, et dans ce cas on a exactement $\phi(n) = n+1-2\lceil \sqrt{n} \rceil$. Avec l'approximation $n+1-2\sqrt{n}$, de Weger démontre qu'il est possible, en fonction du degré de proximité des facteurs p et q, de retrouver $\frac{k}{a}$ dans la liste des convergentes de $\frac{b}{n+1-2\sqrt{n}}$, pour des valeurs $n^{1/4} < a < n^{1/2}$, celles-ci pouvant être assez proches de $n^{1/2}$ si p et q sont suffisamment proches l'un de l'autre. Plus formellement, il démontre que si $|p-q| = n^{\beta}$ et que

$a = n^{\delta}$, pour des valeurs $\beta \in \left(\frac{1}{4}, \frac{1}{2}\right)$ et $\delta \in \left(0, \frac{1}{2}\right)$, et si $\delta < \frac{3}{4} - \beta$, alors $\frac{k}{a}$ est une convergente de $\frac{b}{n+1-2\sqrt{n}}$. Pour contrer une telle attaque, il suggère de s'assurer que la relation $\delta + 2\beta > \frac{7}{4}$ se vérifie. Il est intéressant de remarquer ici que dans sa démonstration, de Weger utilise une inégalité (démontrée en lemme) portant sur la complexité de l'algorithme de Fermat. Celle-ci s'exprime par la relation $0 < p+q-2n^{1/2} < \frac{\Delta^2}{4n^{1/2}}$ où $\Delta = |p-q|$. Si l'on tient compte du fait que le nombre effectif d'itérations de l'algorithme de Fermat est donné par $\frac{p+q}{2} - \left\lceil n^{1/2} \right\rceil$, cette inégalité traduit le nombre maximum d'itérations de cet algorithme en fonction du carré de la distance entre p et q, pondéré par $8n^{1/2}$. Ce résultat peut se comparer à celui que nous avons exprimé en terme de distance entre $\sqrt{p}$ et $\sqrt{q}$, et où nous avons établi que si $\left|\sqrt{p} - \sqrt{q}\right| = c$ alors $\frac{p+q}{2} - \left\lceil n^{1/2} \right\rceil = \left\lceil \frac{c^2}{2} \right\rceil$.

Voyons maintenant comment il est possible de modifier le modèle discuté, pour en étendre la portée, si nous supposons la connaissance d'une information partielle sur les facteurs recherchés.

Attaques sur l'exposant secret, en fonction d'une information partielle

Cette information pourrait prendre différentes formes, comme par exemple la distance minimale entre p et q, ou encore la connaissance d'un certain nombre de chiffres significatifs d'un des deux facteurs; avec son approximation, de Weger utilise en fait une information ($p+q \geq 2\sqrt{n}$) qui est commune à tout nombre. Mentionnons ici que, dans son exposé, de Weger fait référence à un article de Coppersmith [6] et suggère d'essayer d'utiliser les idées qui y sont présentées pour améliorer les attaques de Boneh-Durfee.

Dans l'article en question, Coppersmith démontre qu'il est possible de factoriser efficacement un nombre n, composé de deux facteurs premiers de même taille, si l'on connait la moitié des chiffres (en bits) significatifs d'un des facteurs; un résultat à date jamais dépassé ni même égalé. Durant le cours de notre recherche sur le problème de factorisation, nous nous sommes quelque peu arrêtés sur l'article en question, ainsi que sur un résultat antérieur de Rivet et de Shamir [15], qui eux démontrent qu'il est possible de factoriser efficacement, si l'on a la connaissance des deux tiers premiers bits d'un facteur du nombre. Nous avons alors essayé, par divers moyens, d'intégrer la connaissance des premiers chiffres d'un des facteurs, aux méthodes de Fermat et de Lehman. Nos observations nous avaient permis de constater que pour les situations considérées (par exemple avec la méthode de Lehman en multipliant le nombre à factoriser par $p'q'$, composés des chiffres connus), l'incertitude résultante restait généralement du même ordre de grandeur que celle sur les chiffres inconnus du facteur en question, sauf exceptions dépendantes de la structure effective des nombres considérés. Dans le cas présent toutefois, cette information partielle peut devenir plus significative. Notre argument est, en fait, en continuité avec celui de de Weger, car nous cherchons à montrer qu'il est possible, si l'on dispose d'une information partielle sur les facteurs, d'utiliser une meilleure approximation pour $\phi(n)$. Dans certains cas, celle-ci est suffisante pour permettre de retrouver, pour n et b donnés, des valeurs secrètes a plus grandes que celles atteignables par l'approximation $n+1-2\sqrt{n}$. Remarquons d'abord que sous l'hypothèse $|p-q| > c = n^{\beta}$, il est possible, comme nous l'avons vu, de calculer une valeur minimale pour $p+q$, celle-ci étant donnée par la formule $p+q \geq \frac{-c+\sqrt{c^2+4n}}{2} + \frac{2n}{-c+\sqrt{c^2+4n}}$. Si nous appelons cette valeur x et que nous utilisons $n-x$ comme approximation de $\phi(n)$, il est clair que celle-ci sera meilleure que $n+1-2\sqrt{n}$ quand $\beta > \frac{1}{4}$, puisque dans ce cas cela équivaut à dire que $\left|\sqrt{p}-\sqrt{q}\right| > \sqrt{2}$ et donc que $p+q > 2\sqrt{n}$. Nous avons mentionné, dans un paragraphe précédent, que sur des nombres de 300 chiffres en décimal, avec le seuil recommandé par ANSI, soit $\beta \cong \frac{2}{5}$,

la différence entre x (en fait $\frac{x}{2}$) et $\sqrt{n}$ était généralement de l'ordre de 10^{140}, qui est non négligeable dans le contexte. Dans son article, de Weger souligne l'importance d'une estimation aussi précise que possible pour $\phi(n)$. Dans le cas décrit, nous avons pu observer que l'augmentation de précision obtenue en utilisant l'estimation $n-x$ pour $\phi(n)$ était suffisante pour permettre de retrouver efficacement un exposant secret a, plus grand qu'il n'est possible de le faire avec l'estimation $n+1-2\sqrt{n}$. Remarquons aussi qui si nous exécutons l'algorithme de Fermat, débutant la recherche à partir de la valeur minimale (identifiée) possible pour $\frac{p+q}{2}$, ici $\frac{x}{2}$, et que l'algorithme effectue un certain nombre i d'itérations sans succès, on peut alors utiliser $n-x-2i$ comme estimation de $\phi(n)$. On obtient ainsi, si i est suffisamment important dans le contexte, une approximation de $\phi(n)$ qui est significativement meilleure. Nous pouvons appliquer le même argument si l'on considère la connaissance de quelques chiffres significatifs (les premiers) d'un des facteurs p de n. Si l'on suppose que les deux facteurs premiers sont de même taille (en chiffres), il suffit de prendre les chiffres connus de p, compléter avec suffisamment de zéros pour avoir une valeur minimale pour p, que nous appelons p_0, puis obtenir une valeur minimale q_0 pour q en utilisant $\frac{n}{p_0}$ et en arrondissant de manière appropriée, et finalement d'utiliser $x = p_0 + q_0$ puis $n-x$ comme approximation de $\phi(n)$. Remarquons ici que l'effet discuté est d'autant plus prononcé quand la distance effective entre p et q est grande. En contrepartie, si p et q sont très proches, on aura possiblement $x = p_0 + q_0 \leq 2\sqrt{n}$, auquel cas l'approximation de $\phi(n)$ par $n+1-2\sqrt{n}$ est meilleure. Il suffit alors de prendre $x = \max\left(p_0 + q_0,\ 2\left\lceil \sqrt{n} \right\rceil - 1\right)$.

Précisons, maintenant plus formellement, ces observations :

3.2.1 Proposition

Soit $n = pq$ un nombre composé de deux facteurs premiers et soit $\phi = \phi(n) = (p-1)(q-1)$.

Soit x un entier tel que $0 \le x \le n - \phi$ et soit $\gamma \ge 0$ tel que $n - x - \phi = n^{\gamma}$ (quand $x < n - \phi$).

Soit $a = n^{\delta}$ où $\delta \in \left(0, \frac{1}{2}\right)$ et où $\text{pgcd}(a, \phi) = 1$, et soit $b = a^{-1} \bmod \phi$.

Si n et b sont connus et si $\delta < \frac{1}{2} - \frac{\gamma}{2} - \varepsilon$ (où ε est négligeable),

alors on peut retrouver a, et par suite factoriser n, efficacement.

Démonstration

Selon l'approche utilisée dans [7].

Par définition, il existe un entier k tel que $ab - k\phi = 1$. On désire montrer que si les conditions sont respectées, alors on aura $\left|\frac{b}{n-x} - \frac{k}{a}\right| < \frac{1}{2a^2}$, impliquant que $\frac{k}{a}$ est une convergente de $\frac{b}{n-x}$ ([13], p.161).

On a $\left|\frac{b}{n-x} - \frac{k}{a}\right| \le \left|\frac{b}{n-x} - \frac{b}{\phi}\right| + \left|\frac{b}{\phi} - \frac{k}{a}\right| = b\left|\frac{1}{n-x} - \frac{1}{\phi}\right| + \frac{1}{\phi a}$

$< \phi\left|\frac{1}{n-x} - \frac{1}{\phi}\right| + \frac{1}{\phi a}$ car $b < \phi$

$$= \phi \left| \frac{\phi - (n-x)}{\phi (n-x)} \right| + \frac{1}{\phi a} \leq \frac{n^\gamma}{n-x} + \frac{1}{\phi a}$$

$$\leq \frac{n^\gamma}{\phi} + \frac{1}{\phi a} \quad \text{car } n - x \geq \phi .$$

Il s'en suit alors que $\left| \frac{b}{n-x} - \frac{k}{a} \right| < \frac{1}{\phi}\left(n^\gamma + \frac{1}{a} \right)$.

Il suffit alors de montrer que $\frac{1}{\phi}\left(n^\gamma + \frac{1}{a} \right) < \frac{1}{2a^2}$, qui est équivalent à $n^{2\delta}\left(n^\gamma + \frac{1}{n^\delta} \right) < \frac{\phi}{2}$ si l'on substitue $a = n^\delta$ et que l'on transpose quelques termes. Maintenant, on a sûrement $n^{2\delta}\left(n^\gamma + \frac{1}{n^\delta} \right) = n^{\gamma+2\delta} + n^\delta < 2n^{\gamma+2\delta}$. Alors, si $n^{\gamma+2\delta} < \frac{\phi}{4}$ on obtiendra le résultat désiré. Posons $\frac{\phi}{4} = n^{1-\varepsilon}$ (ε est négligeable car on a toujours $n \approx \phi$, à une petite constante près. Dans sa preuve, de Weger suppose $\phi > \frac{3}{4}n$). On a donc ainsi $n^{\gamma+2\delta} < n^{1-\varepsilon} \Rightarrow \left| \frac{b}{n-x} - \frac{k}{a} \right| < \frac{1}{2a^2}$, d'où l'on tire la condition $\gamma + 2\delta < 1 - \varepsilon$, et par équivalence $\delta < \frac{1}{2} - \frac{\gamma}{2} - \varepsilon$.

Nous avons donc établi que si $\delta < \frac{1}{2} - \frac{\gamma}{2} - \varepsilon$ alors $\frac{k}{a}$ est une convergente de $\frac{b}{n-x}$ et peut donc se calculer efficacement. La factorisation de n s'obtient, comme nous l'avons déjà mentionné, en calculant d'abord ϕ (on résout l'équation $ab - k\phi = 1$ en connaissance de a, b, et k) puis en résolvant $p^2 - (n - \phi + 1)p + n = 0$ pour p.

C.Q.F.D.

Voyons maintenant comment interpréter ce résultat en le comparant à ceux de Wiener et de Weger. Pour simplifier nos explications, nous omettons la quantité ε (négligeable)

de l'inéquation. Nous considérons donc la situation où $\delta < \frac{1}{2} - \frac{\gamma}{2}$ par rapport à $\delta < \frac{1}{4}$, pour Wiener, et par rapport à $\delta < \frac{3}{4} - \beta$ pour de Weger. Rappelons qu'ici $|p-q| = n^{\beta}$ où $\beta \in \left(\frac{1}{4}, \frac{1}{2}\right)$. Tout d'abord, si nous prenons $x = 0$, on utilise alors n comme approximation de ϕ; ce qui correspond à l'attaque de Wiener. Dans ce cas, on a $n - \phi \approx n^{1/2}$, i.e. $\gamma = \frac{1}{2}$, ce qui donne $\delta < \frac{1}{4}$ comme condition pour retrouver a efficacement. Notre résultat est donc en accord avec celui de Wiener. Maintenant, si nous prenons systématiquement $x = 2\lceil \sqrt{n} \rceil - 1$, comme le fait de Weger, il va alors en dépendre, comme nous l'avons déjà remarqué, de la distance effective entre les facteurs. Si celle-ci est telle que $\beta \approx \frac{1}{4}$, cela veut dire que $\lceil \sqrt{n} \rceil$ est proche de $\frac{p+q}{2}$. Ainsi, si $n - x - \phi = n^{\gamma}$, on aura γ proche de zéro, et à la limite $\delta < \frac{1}{2} - \frac{\gamma}{2}$ donnera $\delta < \frac{1}{2}$, en accord ici avec le résultat de de Weger. À l'autre extrême, si nous supposons maintenant que la distance entre les facteurs est de l'ordre $n^{1/2}$, i.e. $\beta \approx \frac{1}{2}$, il est clair que l'on aura aussi $\gamma \approx \frac{1}{2}$. Dans ce cas, $\delta < \frac{1}{2} - \frac{\gamma}{2}$ donnera $\delta < \frac{1}{4}$, en accord ici encore avec le résultat de de Weger $\delta < \frac{3}{4} - \beta$, et qui lui ici correspond à celui de Wiener. Si toutefois on dispose d'une information permettant d'identifier une valeur x telle que $n - x - \phi = n^{\gamma}$ avec $\gamma < \frac{1}{2}$, et que l'on se sert de cette approximation de ϕ en lieu de $n + 1 - 2\sqrt{n}$, il est clair que cela va permettre de retrouver des valeurs $a = n^{\delta}$ supérieures. Par exemple, avec $\gamma \approx \frac{3}{8}$, cela donne $\delta < \frac{5}{16} = 0.3125$ qui est supérieure à

$1-\frac{1}{\sqrt{2}} \cong 0.292$, la borne obtenue par Boneh-Durfee; à la limite, si γ se rapproche de zéro, cela permet d'aller jusqu'à $\delta < \frac{1}{2}$.

Notre modèle se différencie de celui de de Weger en ce sens qu'il permet d'établir une relation entre les valeurs maximales possibles pour δ (sous lesquelles RSA n'est pas sécure) en fonction d'approximations de ϕ, calculées à partir d'informations partielles. Il est intrinsèquement inclusif de celui proposé par de Weger, comme nous avons pu le constater dans nos interprétations. Si l'on prend $\max\left(n-x,\ n-2\lceil\sqrt{n}\rceil+1\right)$ comme estimation de ϕ, où x est une valeur calculée à partir d'une information quelconque, alors cette estimation sera, au pire des cas, égale à celle proposée par de Weger (par exemple si on a aucune information sur les facteurs). Toutefois, si l'information obtenue conduit à une estimation de ϕ significativement meilleure, il devient alors possible de retrouver des valeurs secrètes a de plus en plus grandes, à mesure de la qualité de l'information obtenue et utilisée. Nous donnons en **Annexe D**, un exemple (3.2 – Exemple 1) qui illustre les concepts discutés ici.

Oscar, grand frère

L'importance d'assurer un degré élevé de confidentialité en matière de communications d'informations personnelles ou organisationnelles est acquise. Cette confidentialité peut toutefois être sujette à attaque par des individus ou organisations et ce, pour diverses motivations. Sans entrer dans ces détails, il est clair qu'un bris dans la sécurité d'un système peut s'avérer très lourd de conséquences, comme en font foi nombreux évènements historiques [16]. Nous complétons alors notre mémoire en présentant un petit scénario (évidemment hypothétique), qui, sous des apparences légères dans le contexte, est tout de même porteur d'une question sérieuse et inquiétante. Dans celui-ci *Oscar* joue le rôle de *Big Brother*, ce personnage intangible à l'affût de tous nos secrets, à tous. Nous nous contentons de soulever une question du type « et si ...? » , sans chercher à y répondre, mais en relevant toutefois que la situation imaginée serait possiblement

réalisable, en considération d'observations faites dans notre exposé. Nous situons, ici aussi, notre scénario dans le contexte RSA, donc sécuritairement dépendant du problème de factorisation. Nous imaginons qu'*Oscar* est vraiment diabolique et nous lui attribuons les pires intentions, et, même s'il ne dispose que de ressources limitées, nous supposons que celles-ci pourraient être considérables par rapport à un individu moyen, comme *Bob* ou *Alice*. En fait, *Oscar* pourrait même être une entité organisée subversive, ou criminelle, ou...même (para) gouvernementale, ou quoi encore; le scénario ne le précise pas et le choix est laissé au lecteur. Il est toutefois sous-entendu, qu'en fait, les rôles pourraient tout autant être inversés; le cas échéant, *Bob* et *Alice* pourraient bien être les *vilains*, et les intentions d'*Oscar* pourraient être très louables, mais le point n'est pas là.

Nous supposons que *Bob* et *Alice* sont des utilisateurs avertis des nouvelles technologies et qu'ils utilisent, pour communiquer secrètement, un logiciel des plus récents mis en marché par une firme (réputée) *Oscar Cryptosoft Corporation*. Ce logiciel utilise une technique d'encryption tel RSA, avec un *très grand* modulus *n* pour une sécurité accrue, la documentation sur le produit en faisant foi. Ce produit, des plus performants et d'une grande facilité d'utilisation, a de multiples fonctions qui permettent des calculs avec des grands nombres, des opérations en arithmétique modulaire, ainsi que de générer aléatoirement des paires de grands nombres premiers afin de former les nombres requis par RSA. En fait tout ce qu'il faut afin d'effectuer aisément tout le processus de chiffrement et de déchiffrement. Malgré sa confiance en la réputation de la firme en question, *Bob*, étant de nature douteuse, veut tout de même s'assurer que le produit n'a pas de faille, étant donné l'importance pour lui de préserver la confidentialité de sa correspondance avec *Alice*. Il effectue alors une série de tests destinés à valider le degré de sécurité du logiciel, en fonction de la difficulté à factoriser les nombres obtenus en effectuant le produit des paires de nombres premiers générés par le programme. Il soumet ainsi toute une série de nombres à la factorisation, à l'aide des algorithmes les plus performants qu'il connaisse et dont il dispose, et il constate avec satisfaction que ces nombres résistent à toutes ces attaques, dans les temps alloués. Notons aussi que *Bob* est informé, par la documentation disponible sur le produit, de tous les algorithmes utilisés dans le programme, et s'y connaissant quelque peu en la matière, il a l'assurance de la

validité de ceux-ci. En fait, la seule chose dont *Bob* ne dispose pas, c'est le code source du programme, celui-ci étant de nature *secret industriel*, étant donné la concurrence très active dans ce marché, et les enjeux économiques importants qui en dépendent. Qu'à cela ne tienne, tout paraît bien conforme, et *Bob* ne va tout de même pas tenter de retracer le code-source du logiciel à partir du code-machine du programme exécutable. Cette tâche s'avérerait certes très longue et presque sûrement futile, même si *Bob* a une certaine expertise en informatique. Sur ce point, il va donc accorder le bénéfice du doute à la firme en question; il a tout de même pris toutes les précautions raisonnables, et de toute manière, a-t-il vraiment le choix ? Ce produit ou un autre, le problème risque fort de demeurer. Et c'est justement cette minuscule faille qu'*Oscar* choisit d'exploiter. Il a compris qu'il était possible de générer aléatoirement une multitude de paires de nombres premiers de toutes tailles, liés entre eux par une relation quasi-indécelable, étant donné le nombre effarant de possibilités. Il réalise aussi que ce lien pourrait, dans les faits, se traduire par un ensemble limité de *clefs de factorisation*, capables de permettre à l'aide d'algorithmes à la Fermat-Lehman (ou variantes), la factorisation de tout nombre qui est le produit d'une paire liée selon une procédure (secrète) pré-établie; en somme, une ou quelques clefs universelles pour factoriser efficacement tous les nombres générés par le programme, tout en s'assurant que ces nombres paraissent entièrement aléatoires et conformes aux normes établies.

Nous avons montré, à travers divers exemples, qu'il était possible de faire des associations entre de vastes classes de nombres, générés pseudo-aléatoirement, selon des protocoles définis, et des clefs permettant à l'aide d'algorithmes de factorisation paramétrisés (tels ceux présentés) d'effectuer une factorisation efficace, et ainsi de contourner la sécurité d'un cryptosystème comme RSA. Alors,

La question est : Et si *Oscar* cachait une petite *trappe* dans les méandres de son code ?

Mais terminons ce scénario sur une note d'optimisme, en rappelant que tous les personnages et organisations mentionnés dans celui-ci sont purement fictifs, et que la situation décrite est, bien entendu, entièrement imaginée.

Conclusion

La première partie de notre exposé a permis de caractériser le problème de factorisation dans un cadre mathématique spécifique. Notre analyse nous a conduit à établir l'équivalence d'un ensemble de relations impliquant les facteurs d'un nombre donné. Celles-ci se traduisent, dans les faits, par des classes d'entiers. L'une de ces relations, $\left|\sqrt{p}-\sqrt{q}\right|<\sqrt{2}$, permet d'énoncer un critère de factorisation, selon une mesure de distance entre les facteurs. Nous avons défini une fonction (*split*) capable de trouver efficacement deux facteurs non triviaux d'un nombre donné, si et seulement si celui-ci appartient à cette classe. C'est également une fonction qui, en cas d'échec, permet d'exclure un grand nombre de candidats facteurs. Il découle du premier chapitre, un algorithme de factorisation, plutôt simple et limité, mais un tant soit peu efficace.

Le deuxième chapitre nous a d'abord conduit à travers des sentiers oubliés, suivant quelques traces laissées par Fermat, *mathémagicien* par excellence. On constate qu'une de ses méthodes, considérée *vieillotte*, masque une relation significative concernant le problème de la factorisation. Notre analyse a permis de caractériser cette relation, en continuité des résultats précédents. Les idées qui en découlent nous ont mené à étendre la méthode, à l'aide des notions de *clefs de factorisation* et d' *algorithme de factorisation paramétrisé*. Cette approche a servi, par la suite, de trait d'union avec une méthode proposée par Lehman (désuète elle aussi!) qui généralise, dans une direction spécifique, celle de Fermat. Notre analyse de la méthode de Lehman nous a amené à relever l'importance (contextuelle) des *stratégies* de factorisation, puisqu'elles permettent, dans une certaine mesure, d'établir des relations entre des classes de nombres et des algorithmes paramétrisés capables de les traiter efficacement. Nous avons situé la méthode originale de Fermat, dans un contexte d'expressions quadratiques un peu plus générales, et l'avons prolongée dans ce sens. Nous avons également élargi la portée du critère de factorisation établie en première partie et, par suite, élaboré un algorithme de factorisation généralisant celui de Fermat, cette fois selon l'axe d'exposants.

Dans le dernier chapitre, nous avons discuté d'applications possibles à la cryptologie. Nous avons d'abord utilisé le critère de factorisation, démontré au premier chapitre, en combinaison avec les concepts de clef de factorisation et d'algorithme paramétrisé, pour élaborer un cryptosystème à clef privée. Notre analyse de ce système, quoique succincte, a permis d'en relever quelques caractéristiques intéressantes, notamment une sécurité à première vue élevée (avec des clefs suffisamment complexes, bien entendu), qui découle de propriétés établies.

Nous avons ensuite montré comment appliquer quelques uns de ces concepts, dans des situations hypothétiques, pour mettre à risque la sécurité du cryptosystème RSA. Nous avons noté que le fait de préciser un seuil minimal de distance, entre les paramètres (facteurs premiers) du système, réduisait de facto l'espace de recherche, et ne contribuait pas à régler la question des cas frontières, face à une attaque à l'aide d'un algorithme (paramétrisé) à la Fermat. Notre analyse de l'attaque (de Weger) contre l'exposant secret RSA, en fonction de la distance entre les facteurs premiers, a mis en évidence certains liens qui s'imposaient avec notre traitement du problème de la factorisation. Cela nous a aussi permis de développer et de présenter un modèle un peu plus généraliste, tenant compte d'informations partielles sur les paramètres du système. Le petit scénario, élaboré à la fin, nous a permis de formuler une question qui, à notre avis, méritait d'être posée, particulièrement dans le contexte actuel des technologies de l'information, de leur utilisation maintenant courante, et des considérations de sécurité qui en découlent.

Notre recherche a effleuré, à peine, ces deux vastes mondes que sont la factorisation d'entiers et la cryptologie. Chacun est dense de ramifications, riche d'une longue et fascinante histoire. Le problème de la factorisation demeure, bien entendu, toujours ouvert ! Quant à la cryptologie, l'histoire ne s'arrête sûrement pas ici.

Sources documentaires

[1] Boneh, D., Durfee, G.. *Cryptanalysis of RSA with Private Key d less than* $N^{0.292}$. Advances in Cryptology – EUROCRYPT '99, Lecture Notes in Computer Science 1592, pages 1-11, Springer-Verlag, Berlin, 1999.

[2] Brassard, G. and Bratley, P.. *Fundamentals of Algorithmics.* Prentice Hall, 1996.

[3] Bressoud, D.M.. *Factorization and Primality Testing.* Springer-Verlag, 1989.

[4] Chabert, J.L., Barbin, E., Guillemot, M., Michel-Pajus, A., Borowczyk, J., Djebbar, A., Martzloff J.C.. *Histoires d'Algorithmes - du caillou à la puce.* Éditions Belin, 1994.

[5] Cohen, H.. *A Course in Computational Number Theory.* Springer-Verlag, Berlin, 1993.

[6] Coppersmith, D.. *Finding a Small Root of a Bivariate Integer Equation : Factoring with High Bits Known.* Advances in Cryptology – EUROCRYPT '96, Lecture Notes in Computer Science 1070, pages 178-189, Springer-Verlag, Berlin, 1996.

[7] de Weger, B.. *Cryptanalysis of RSA with small prime difference* (May 2, 2000). site internet : http:\\ www.iacr.org

[8] Diffie, W., Hellman M. E.. *New directions in cryptography.* IEEE Transactions on Information Theory, 22, 1976, pages 644-654.

[9] Dixon, J.D.. *Asymptotically fast factorization of integers*, pages 255-260, vol. 36, no. 153, Mathematics of Computation, 1981.

[10] Knuth, D. E.. *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

[11] Lehman, R. Sherman. *Factoring Large Integers*. Math. Comp., **28** (1974), pages 637-646.

[12] Menezez, A.J., van Oorschot, P.C. , Vanstone, S.A.. *Handbook of Applied Cryptography*. CRC Press, 1997.

[13] Niven, I. and Zuckerman, H.S.. *An Introduction to the Theory of Numbers*, third edition, John Wiley & Sons Inc., 1972

[14] Rivest, R.L., Shamir, A., Adleman, L.. *A method for obtaining digital signatures and public key cryptosystems*. Communications of the ACM, 21 (1978), pages 120-126.

[15] Rivest, R.L., Shamir, A.. *Efficient factoring based on partial information.* Advances in Cryptology-EUROCRYPT '85 (LNCS 219), pages 31-34, 1986.

[16] Singh, S.. *Histoire des codes secrets* (traduit de l'anglais). Éditions Jean-Claude Lattès, 1999.

[17] Stinson, D.. *Cryptographie théorie et pratique* (traduction de l'anglais). International Thomson Publishing France, 1996

[18] Wiener, M.J.. *Cryptanalysis of short RSA secret exponents*. IEEE Transactions on Information Theory, 32 (1990), pages 553-558

[19] site internet :
http:\\www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html

Annexe A

Notation et rappel mathématique

Notation et rappel mathématique

$\lceil x \rceil$	le nombre réel x arrondi par le haut
$\lfloor x \rfloor$	le nombre réel x arrondi par le bas
$\sqrt{n}$ ou $n^{1/2}$	la racine carrée d'un nombre n
$\sqrt[k]{n}$ ou $n^{1/k}$	la racine k-ième d'un nombre n
pgcd (m, n)	le plus grand diviseur commun à m et n

Définitions - Basées sur l'arithmétique modulaire

- $\mathbf{Z}_n = \{0, 1, 2, \ldots, n-1\}$ l'ensemble des entiers modulo n
- Soit $a \in \mathbf{Z}_n$. L'*inverse multiplicatif* de a est un élément $a^{-1} \in \mathbf{Z}_n$ tel que $aa^{-1} \equiv a^{-1}a \equiv 1 \bmod n$
- Deux entiers m et n sont *relativement premiers* si et seulement si pgcd $(m, n) = 1$.
- $\phi(n)$ représente le nombre d'entiers de $\mathbf{Z}_n$ qui sont relativement premiers avec n. ϕ est appelée la *fonction indicatrice d'Euler*

Quelques théorèmes importants

- L'équation $ax \equiv b \bmod n$ admet une solution unique $x \in \mathbf{Z}_n$, pour tout $b \in \mathbf{Z}_n$, si et seulement si pgcd$(a, n) = 1$
- Le *théorème fondamental de l'arithmétique* (voir section 1.1)
- Supposons $n = \prod_{i=1}^{m} p_i^{e_i}$ où les p_i sont des nombres premiers distincts et les exposants $e_i > 0$ pour $1 \le i \le m$ (une représentation unique d'après le théorème mentionné ci-dessus). Alors $\phi(n) = \prod_{i=1}^{m} (p_i^{e_i} - p_i^{e_i - 1})$

  Cas particuliers:

  Si p est premier alors $\phi(p) = p - 1$ Si $n = pq$ avec p et q premiers distincts, alors $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$
- *Théorème d'Euler*: Soit $a \in \mathbf{Z}_n$. Si pgcd$(a, n) = 1$ alors $a^{\phi(n)} \equiv 1 \bmod n$.

Annexe B

Fonctions Maple V© utilisées

Annexe B

Principales fonctions utilisées

a^(-1) **mod** n	retourne l'inverse multiplicatif de a modulo n, quand il existe
ceil (x)	équivaut à la fonction mathématique $\lceil x \rceil$
cfrac(a, k, x)	retourne une liste x contenant les k premières fractions convergentes vers a, selon la méthode de fractions continues
convert (x, y)	effectue une conversion de x en type y
evalf(x, k)	retourne une évaluation numérique de x avec une précision de k chiffres
floor (x)	équivaut à la fonction mathématique $\lfloor x \rfloor$
ifactor (n)	retourne les facteurs de l'entier n
igcd (m, n)	retourne le pgcd de m et n
isprime (p)	vérifie si un nombre donné p est premier ou composé, et retourne *vrai* ou *faux* selon le cas. **Remarque**: Cette fonction est basée sur un test probabiliste qui s'effectue efficacement, à un coût polynomial en $\log p$. Si la fonction retourne *vrai*, alors p est fort *probablement premier* ([12], def. 4.5). Quand **isprime** (p) retourne *faux*, p est sûrement composé (voir [12], 4.2).
isqrt (x)	retourne l'entier le plus proche de $\sqrt{x}$
length (x)	retourne le nombre de chiffres d'un x donné
nextprime (n)	retourne le premier nombre premier plus grand que l'entier n. **Remarque**: Utilise la fonction **isprime** pour déterminer si les successeurs impairs de n sont composés ou probablement premiers. Le calcul s'effectue alors efficacement, en temps polynomial en $\log p$.
numtheory	librairie de fonctions spécifiques à la théorie des nombres, dont *nextprime* et *cfrac*

rand(r)	retourne un entier pseudo-aléatoire compris dans l'intervalle $0 .. r-1$, selon une distribution uniforme U(0, r -1). Avec un intervalle de valeurs comme paramètre d'entrée, **rand(r..s)** retourne, uniformément, un nombre de cet intervalle.
Rnd(*k*)	retourne, uniformément, un entier pseudo-aléatoire compris dans l'intervalle $10^{k-1} .. 10^{k}$. Correspond à **rand(**$10^{k-1} .. 10^{k}$**)**.
sqrt (x)	retourne $\sqrt{x}$
type (x, y)	retourne vrai si x est de type y

Annexe C

Procédures en Maple V

Procédure *split*

Entrée : un nombre n à factoriser

```
proc (n)
local x;
      x := ceil(evalf(sqrt(n),length(n)));
      if type(sqrt(x^2-n),posint) then [x + sqrt(x^2-n), x-sqrt(x^2-n)] else [1, n] fi
end
```

Sortie: une paire de facteurs de n, ou la paire $[1, n]$

Procédure *Fermat*

Entrée : un nombre n à factoriser, et une borne B, qui limite le nombre d'itérations

```
proc (n, B)
local b, i, x, r, z, p, q;
      x := floor(evalf(sqrt(n), length(n)));
      for i to B do
        r := n-x^2;
        z := 2*x+1-r;
        if type(z,square) then  p := x+1+sqrt(z); q := x+1-sqrt(z); b := i; i := B  fi;
        x := x+1
      od;
     [[b], p, q]

end
```

Sortie: un triplet $[[b], p, q]$ contenant le nombre $[b]$ d'itérations effectuées, et deux facteurs de n (ou la paire 1 et n)

Procédure *Fermat_B*

Entrée : un nombre n à factoriser, et une borne B, qui limite le nombre d'itérations

```
proc (n, B)
local b, i, x, r, z, p, q;
      x := floor(evalf(sqrt(n), length(n)));
      for i to B do
       r := n-x^2;
       z := 2*x+1-r;
       if type(z,square) then  p := x+1+sqrt(z); q := x+1-sqrt(z); b := i; i := B
          print ([[b], p, q])
       fi;
       x := x+1
    od;
end
```

Sortie: un ou plusieurs triplets $[[b], p, q]$ contenant le nombre $[b]$ d'itérations effectuées, et deux facteurs de n (ou la paire 1 et n)

Procédure *Fermat_Classique*

Entrée : un nombre n à factoriser, et une borne B, qui limite le nombre d'itérations

```
proc(N, B)
local n, b, i, k, x, r, z, p, q, sol;
    n := N;
    if N mod 2 = 0 and N mod 4 ≠ 0 then n := 1 / 2*N fi;
    if isprime(n) then q := n; p := N / n; b := 0; print([[b], p, q]) fi;
    if b ≠ 0 then
        x := floor(evalf(sqrt(n), 2*length(n)));
        for i to B do
            k := 2*i + 1;
            r := n - x^2;
            z := 2*x + 1 - r;
            if type(n / k, posint) then
              p:= n / k; q:= N / p; b:= i; print([[b], p, q],'_Classique')
            fi
            if type(z, square) then
              p:= p:= x + 1 - sqrt(z); q:= N / p; b:= i; print([[b], p, q],'_Fermat')
            fi
            x := x + 1;
            if b = i then i := B fi
        od
    fi;
end
```

Sortie: un triplet $[[b], p, q]$ contenant le nombre $[b]$ d'itérations effectuées, et deux facteurs de n (ou la paire 1 et n), et identifie la composante *Classique* ou *Fermat* qui trouve la factorisation.

Procédure *Fermat_Clef*

Entrée : un nombre N à factoriser, un nombre *clef* de factorisation, et une borne B, qui limite le nombre d'itérations

```
proc(N, clef, B)
local b, c, i, x, n, r, z, p, q;
    p := 1;
    n := N*clef;
    x := floor(evalf(sqrt(n), 2*length(n)));
    for i to B do
        r := n - x^2;
        z := 2*x + 1 - r;
        if type(z, square) then
          p:= p:= x + 1 + sqrt(z); q:= x + 1 - sqrt(z); b:= i; i:= b;
        fi
        x := x + 1
    od;
    if 1 < p then p := igcd(p, N); q := N / p fi;
    [[b], p, q]
end
```

Sortie: un triplet [[b], p, q] contenant le nombre [b] d'itérations effectuées, et deux facteurs de n (ou la paire 1 et n)

Procédure *Fermat_Lehman*

Entrée : un nombre N à factoriser, et deux intervalles de recherche $K = [K_{\min}, K_{\max}]$ et $B = [B_{\min}, B_{\max}]$, qui limitent le nombre d'itérations

```
proc(N, K, B)
local b, x, n, z, p, q, k, kmin, kmax, ksol, bmin, bmax, bsol;
    kmin := K[1];
    kmax := K[2];
    bmin := B[1];
    bmax := B[2];
    for k from kmin to kmax do
        n := 4*k*N;
        x := floor(evalf(sqrt(n), length(n)));
        for b from bmin to bmax do
            z := (x + b)^2 - n;
            if type(z, square) then
              q:= x + b + sqrt(z); bsol:=b; ksol:=k ; b:=bmax ; k:=kmax ;
            fi
        od
    od;
    if q ≠ 'q' then q := igcd(q, N); p := N / q fi;
    print('P' = p);
    print('Q' = q);
    print('ksol' = ksol);
    print('bsol' = bsol)
end
```

Sortie: Les facteurs p et q (si) trouvés, et les valeurs *ksol* et *bsol* associées à la solution.

Procédure *split_quad*

Entrée : un nombre n à factoriser, et une borne B, qui limite le nombre d'itérations

```
proc(n, B)
local x, b, c, d, p, q, bsol;
    x := floor( evalf( sqrt(n), length(n) ));
    for b from 0 to B do
        c:= x^2 + b * c - n; d:= b^2 + 4 * c;
        if type(d, square) then
            p:= x + 1 / 2 * b - 1 / 2 * sqrt(d); q:= n / p;bsol:= b;b:= B;
        fi
    od;
    [ [ bsol ], p, q ]
end
```

Sortie: un triplet [[*bsol*], p, q] contenant le nombre [*bsol*] d'itérations effectuées, et deux facteurs de n (ou la paire 1 et n)

Procédure *Fermat_kieme*

Entrée : un nombre N à factoriser, et deux intervalles de recherche $K = [K_{min}, K_{max}]$ et $B = [B_{min}, B_{max}]$, qui limitent le nombre d'itérations

```
proc (n, K, B)
local b, c, j, k, x, y, p, q;
    for k from K[1] to K[2] do
        x := floor(evalf(n^(1/k),length(n)));
        for b from B[1] to B[2] do
            y := convert(evalf(((x+b)^k-n)^(1/k),length(n)),fraction);
            if type(y,integer) and x+b-y <> 0 and type(n/(x+b-y),integer) then
              print([k, b]); p := x+b-y; q := n/p; k := K[2]; b := B[2]; [p, q]
            else
              y := convert(evalf((n-(x-b)^k)^(1/k),length(n)),fraction);
              if type(y,integer) and x-b+y <> 0 and type(n/(x-b+y),integer) then
                print([k, b]); p := x-b+y; q := n/p; k := K[2]; b := B[2]; [p, q]
              fi
          fi
        od
    od
end
```

Sortie: la paire de facteurs $[p, q]$, si trouvés, et la paire de valeurs $[k, b]$ associées à la solution

Procédure *Chiffre*

Entrée : un nombre n à chiffrer, et une *clef* de chiffrement composée, en fait, de 8 clefs distinctes.

```
proc(n, clef)
local k, a, b, c, d, e, f, g, M, N, p, q;
    k := clef[1];
    a := clef[2];
    b := clef[3];
    c := clef[4];
    d := clef[5];
    e := clef[6];
    f := clef[7];
    g := clef[8];
    M := Rnd(k)*10^(k + length(n)) + 10^k*n + Rnd(k);
    p := nextprime(floor(evalf(a*e*M / (b*f), length(M))) + Rnd(k - 3));
    q := nextprime(floor(evalf(c*e*M / (d*f), length(M))) + Rnd(k - 3));
    N := p*q + g
end
```

Sortie: un nombre N, chiffré.

Procédure *Dechiffre*

Entrée : un nombre *NN* à déchiffrer, et une *clef* de déchiffrement composée de huit clefs distinctes.

```
proc(NN, clef)
local k, a, b, c, d, e, f, g, i, l, C, M, x, P, Q, N, msg, n,
    k := clef[1];
    a := clef[2];
    b := clef[3];
    c := clef[4];
    d := clef[5];
    e := clef[6];
    f := clef[7];
    g := clef[8];
    N := NN - g;
    C := N*a*c*b*d;
    x := ceil(evalf(sqrt(C), length(C)));
    P := igcd(x + sqrt(x^2 - C), N);
    Q := N / P;
    M[1] := ceil(evalf(P*b*f / (a*e), length(P)));
    M[2] := ceil(evalf(P*d*f / (c*e), length(P)));
    M[3] := ceil(evalf(Q*b*f / (a*e), length(Q)));
    M[4] := ceil(evalf(Q*d*f / (c*e), length(Q)));
    msg[1] := CutR(M[1], k);
    msg[1] := CutL(msg[1], k);
    msg[2] := CutR(M[2], k);
    msg[2] := CutL(msg[2], k);
    msg[3] := CutR(M[3], k);
    msg[3] := CutL(msg[3], k);
    msg[4] := CutR(M[4], k);
    msg[4] := CutL(msg[4], k);
   if msg[1] = msg[1] or msg[1] = msg[4] then n := msg[1] else n := msg[2] fi
end
```

Sortie: un nombre *n*, le message déchiffré.

Annexe D

Exemples

1.1 Exemple 1

x:=Rnd(40);n:=x^2-1;

x := 279150298977484802372202899739136804732

n:=77924889419219152700819631656494844737035230167072485098868395656102306975918 23

split(n);

[279150298977484802372202899739136804733,

279150298977484802372202899739136804731]

1.1 Exemple 2a

```
> l:=Rnd(2);x:=Rnd(l);y:=Rnd(floor(l/3));n:=x^2-y^2;
l := 24
x := 729479870548736450984003
y := 24701461
n := 532140881535801290157255453677738516824786369488

> split(n);
[729479870548736475685464, 729479870548736426282542]
```

1.1 Exemple 2b

```
> l:=Rnd(2);x:=Rnd(l);y:=Rnd(floor(3*l/4));n:=x^2-y^2;
l := 33
x := 412272244885805483826208735338321
y := 963489794196134912196910
n :=
169968403903181564985412334317136732801543367521864160899717550941

> split(n);
 [1,
169968403903181564985412334317136732801543367521864160899717550941]
```

1.1 Exemple 3

> **p:=nextprime(Rnd(40));q:=nextprime(p+Rnd(20));n:=p*q;split(n);**

p := 715747810811411522216610945716861254678 7

q := 715747810811411522224841156702144204623 1

n:=512294928681328140734890673918854666235915636105707048726192351898193335045097 97

> **split(n);**

[715747810811411522224841156702144204623 1,

715747810811411522216610945716861254678 7]

2.1 Exemple 1a

p := 23178314906195914825136973281314862289449638581889

q := 23178314906195914825137020362017973589046220869705

deux nombres premiers, chacun de 50 chiffres, on a alors

n:=53723428189078373985959685646611917740349561708409407295985762040249814962581762744166195390122591 1

Ici, $c = \left|\sqrt{p} - \sqrt{q}\right| \cong 4.88958516469244398451677 15$ et $\left\lceil \frac{c^2}{2} \right\rceil = 12$

> **Fermat(n,20);**

[[12],
23178314906195914825137020362017973589046220869705,
23178314906195914825136973281314862289449638581889]

2.1 Exemple 1b

> p:=nextprime(p+Rnd(25));q:=nextprime(q+Rnd(25));n:=p*q;

p := 23178314906195914825136974728849642481264232238041

q := 23178314906195914825137026355060302087070763760491

n := 53723428189078373985959702892615848644929556960791930071470647597008197794145671417654473422303813 1

> Fermat(n,20);

[[15],
23178314906195914825137026355060302087070763760491,
23178314906195914825136974728849642481264232238041]

2.1 Exemple(s) 2

> B:=100;n:=7*29; Fermat_Classique(n,B);

$B := 100$

$n := 203$

[[3], 29, 7], _Classique

> n:=23*29; Fermat_Classique(n,B);

$n := 667$

[[1], 23, 29], _Fermat

> p:=nextprime(Rnd(2));q:=nextprime(Rnd(20));n:=p*q;

$p := 59$

$q := 12673709525428506221$

$n := 747748862000281867039$

> Fermat_Classique(n,B);

[[29], 12673709525428506221, 59], _Classique

> p:=nextprime(Rnd(30));q:=nextprime(p+Rnd(16));n:=p*q;

$p := 268026856069438450681407469457$

$q := 268026856069445521555467425029$

$n := 71838395574469370432263054657988260143737569601806754839253$

> Fermat_Classique(n,B);

[[24], 268026856069438450681407469457,

268026856069445521555467425029], _Fermat

2.1 Exemple 3

>f:=(x,n)->x-sqrt(x^2-n);g:=(x,n)->x+sqrt(x^2-n);

$$f := (x,n) \to x - \sqrt{x^2 - n}$$

$$g = (x,n) \to x + \sqrt{x^2 - n}$$

> p:=13;q:=47;n:=p*q;r:=ceil(sqrt(n));

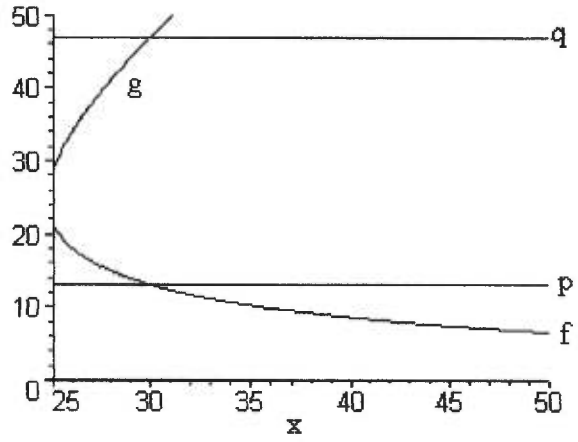
$p := 13$

$q := 47$

$n := 611$

$r := 25$

> plot({p,q,f(x,n),g(x,n)},x=r..2*r,colour=black);



2.2 Exemple 1

$p :=$

805708070311129960512711459455292120992889151524251562032482805591285422750752571798135144738812217 1

$q :=$

575505764507949971794793899610922943566349393945893972880345140877977401700865421462997890565282966 7

> n:=p*q;

$n :=$

463689638974631957217534123755610517278129116742518392095665238745025774669963043563225066665029352573754095742324730079787393342347267330356372590493311454021391522388483070354144869011585716492470 57

> Fermat_Clef(n,35,100);

[[1],

575505764507949971794793899610922943566349393945893972880345140877977401700865421462997890565282966 7,

805708070311129960512711459455292120992889151524251562032482805591285422750752571798135144738812217 1]

2.2 Exemple 2

> p:=nextprime(Rnd(150));

p :=

876798339062890227065912603127119521589474574157513825150650905007553408748182082815984929359632269852681585809504709739738485231104248045693803729709

q :=

626284527902064447904223287947942515421053267255367017964750646433966720534415773439989235256880192751915418438898517775822536933670965351037689577937

> n:=p*q;

n:=

549125233845316438761003720387465179227319483854900694441825956383484386091947330228408849206108676001176293024983076625084398768215139068495957893497092572170912347199382093415446014611806376192720948043508762591727566593537599225138057470449876030599798608223671787099835913816264865767848637830333

> Fermat_Clef(n,35,10);

[[1],

626284527902064447904223287947942515421053267255367017964750646433966720534415773439989235256880192751915418438898517775822536933670965351037689577937,

876798339062890227065912603127119521589474574157513825150650905007553408748182082815984929359632269852681585809504709739738485231104248045693803729709]

2.2 Exemple 3

> Fermat_Clef(n,(p-1)*(q-1),100);

[[1],
62628452790206444790422328794794251542105326725536701796475064643396
67205344157734399892352568801927519154184388985177758225369336709653
51037689577937,
87679833906289022706591260312711952158947457415751382515065090500755
34087481820828159849293596322698526815858095047097397384852311042480
45693803729709]

> Fermat_Clef(n,(p+1)*(q+3),100);

[[1],
87679833906289022706591260312711952158947457415751382515065090500755
34087481820828159849293596322698526815858095047097397384852311042480
45693803729709,
62628452790206444790422328794794251542105326725536701796475064643396
67205344157734399892352568801927519154184388985177758225369336709653
51037689577937]

> Fermat_Clef(n,35*12345^2,100);

[[1],
62628452790206444790422328794794251542105326725536701796475064643396
67205344157734399892352568801927519154184388985177758225369336709653
51037689577937,
87679833906289022706591260312711952158947457415751382515065090500755
34087481820828159849293596322698526815858095047097397384852311042480
45693803729709]

2.2 Exemple 4a

>l:=Rand(250);p:=nextprime(Rnd(l));a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=nextprime(floor(evalf(a*p/b,length(p)))+d);n:=p*q;

$l := 212$

$p :=$

64630632990365572644369865198827953573637755090805168833872126019816784770471786002056737615799415173025434401195901438461406403621591847184755288115823429734930482183428606633928192552090120660217372949065516371

$a := 71$

$b := 86$

$d :=$

68807201851302480764856705727979965038867148497422117111078967825806626564261139459929762307205492682420 53

$q :=$

53357848166464600671514656152520752368933495481943802176801406365197578124459265187744515938625098573079144584033080503861563167730259090439732089694192341178921157141241081488341329267415743753852040076985122089

$n :=$

34485515020024242014796715165206251856080475083840381396240957331386602752334985289571458718271558922801669283483959600496450929984515865781974083986097783821772388147569315737246174853857647903418212571481917594785548193558308061821539183840931874090434096674167592818966588877481644810315359537905888874675835423686421049360055329737144215662736146140212169690274743162796164442809092006251472810600046944015962812632190 19

>Fermat_Lehman(n,[1,10000],[1,100]);

P =

64630632990365572644369865198827953573637755090805168833872126019816
78477047178600205673761579941517302543440119590143846140640362159184
71847552881158234297349304821834286066339281925520901206602173729490
65516371

Q = 53357848166464600671514656152520752368933495481943802176801 4
06365197578124459265187744515938625098573079144584033080 50
38615631677302590904397320896941923411789211571412410814 88
34132926741574375385204007698512208 9

ksol = 6106

bsol = 20

2.2 Exemple 4b

>l:=Rand(100);p:=nextprime(Rnd(l));a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=nextprime(floor(evalf(a*p/b,length(p)))+d);n:=p*q;

$l := 61$

$p :=$ 905177771327273258068682489009023088485502888894411492553563

$a := 48$

$b := 79$

$d :=$ 2100121961819618967847669970449

$q :=$ 54998143067986223275059189205632530646585460325035124348528 71

$n :=$

49783096569418292512413772925349408873612789749215844797466008852053

46479745341774447898789268821282821081232034678315137 3

> Fermat_Lehman(n,[1,10000],[1,100]);

$P =$ 905177771327273258068682489009023088485502888894411492553563

$Q =$ 54998143067986223275059189205632530646585460325035124348528 71

$ksol = 3792$

$bsol = 16$

2.2 Exemple 5a

l:=Rand(100);p:=Rnd(l);a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=floo
r(evalf(a*p/b,length(p)))+d;n:=p*q;

l := 52

p := 778400904201459506298432094118984663897572715325733 9

a := 67

b := 78

d := 74552503936135042302187661

q := 66862641771151008874352501137796798952552181902420 42

n :=

52045940811962221229333014801642821149087661195477746449100054915639
23608705407937767855861542512284623 8

> Fermat_Lehman(n,[1,10000],[1,100]);

P = 778400904201459506298432094118984663897572715325733 9

Q = 66862641771151008874352501137796798952552181902420 42

ksol = 5226

bsol = 17

On constate qu'ici encore la procédure a retourné les facteurs *p* et *q* dans les limites fixées, et ce, malgré le fait qu'aucun de ces deux facteurs ne soit premier.En effet:

> isprime(p);isprime(q);

false

false

2.2 Exemple 5b

>l:=Rand(100);p:=Rnd(l);a:=Rand(100);b:=Rand(100);d:=Rand(isqrt(p));q:=floor(evalf(a*p/b,length(p)))+d;n:=p*q;

l := 76

p :=
7735887800457862510421921865626146414372078842657662729297077739372579317480

a := 16

b := 88

d := 337276738288519791558277770619448677 88

q :=
1406525054628702274622167611932026620828651099766608839028023729675141107330

n :=
1088072001114050654856550445530017369570207296553304040642632277939628976581618550009983671697508642446546510793281045175616409603716082761702825128400

> Fermat_Lehman(n,[1,10000],[1,100]);

P =
3867943900228931255210960932813073207186039421328831364648538869686289658740

Q =
2813050109257404549244335223864053241657302199533217678056047459350282214660

ksol = 22

bsol = 3

On constate ici qu'une solution est trouvée à *ksol* = 22 ; ceci est dû au fait que *a* et *b* ont un facteur commun (8) alors $\frac{a}{b} = \frac{16}{88} = \frac{2}{11}$. On remarque aussi que les facteurs retournés diffèrent des valeurs originales par un facteur de deux et si nous cherchons une autre solution avec $k = ab$ on obtient encore deux facteurs différents:

> Fermat_Lehman(n,[a*b,a*b],[1,100]);

P =

9669859750572328138027402332032683017965098553322078411621347174215724146 85

Q =

1125220043702961819697734089545621296662920879813287071222418983740112885 8640

ksol = 1408

bsol = 18

2.2 Exemple 6

Dans cet exemple, on choisit aléatoirement un nombre *n* quelconque mais de taille plus modeste cette fois (une dizaine de chiffres au plus) puis on applique la procédure sur *n* avec différents intervalle de recherche pour constater l'existence de différentes valeurs solutions:

n:=Rnd(10);

n := 7325037837

On se sert ici de la fonction ifactor de Maple pour obtenir la factorisation complète de n, pour fin de comparaison:

> **ifactor(n);**

$(3)^2$ (13) (71) (593) (1487)

Fermat_Lehman(n,[1,1],[1,2*10^4]);

P = 69381

Q = 105577

ksol = 1

bsol = 3786

> **Fermat_Lehman(n,[1,10],[1,1000]);**

P = 57993

Q = 126309

ksol = 2

bsol = 221

> **Fermat_Lehman(n,[3,10],[1,2*10^4]);**

P = 42103

Q = 173979

ksol = 3

bsol = 3808

> **Fermat_Lehman(n,[4,10],[1,5000]);**

$P = 42103$

$Q = 173979$

$ksol = 4$

$bsol = 46$

> **Fermat_Lehman(n,[5,20],[1,100]);**

$P = 69381$

$Q = 105577$

$ksol = 6$

$bsol = 11$

> **Fermat_Lehman(n,[5,5],[1,5000]);**

$P = 173979$

$Q = 42103$

$ksol = 5$

$bsol = 1740$

Et ainsi de suite...

> **Fermat_Lehman(n,[Rand(100),100],[1,500]);**

$P = 173979$

$Q = 42103$

$ksol = 68$

$bsol = 140$

2.2 Exemple 7

p:=nextprime(1234);q:=nextprime(5678);n:=p*q;

$p := 1237$

$q := 5683$

$n := 7029871$

Fermat_Lehman(n,[3,3],[1,10]);

$P = p$

$Q = q$

$ksol = ksol$

$bsol = bsol$

Ici, nous avons effectué sans succès une recherche des facteurs de n, avec $k = 3$ et b variant de 1 à 10. On peut donc éliminer comme facteur possible de n tout nombre dans l'intervalle [8781, 9607], calculé selon la formule:

$$\left[(X_0 + B_{\max}) \pm \left\lfloor \sqrt{(X_0 + B_{\max})^2 - N} \right\rfloor\right] \text{ où } X_0 = \left\lfloor \sqrt{N} \right\rfloor$$

et on peut alors aussi éliminer comme facteur de n tout nombre de l'intervalle:

$$\left[\frac{7029871}{9607}, \frac{7029871}{8781}\right] \cong [731, 800].$$

2.3 Exemple 1

$n = p^2$ où p est premier. Prenons, par exemple $n = 19^2$.

> **B:=200: n:=19^2;split_quad(n,B);Fermat(n,B);**

$n := 361$

[[0], 19, 19]

[[162], 1, 361]

On constate ici que *split_quad* trouve immédiatement la factorisation 19×19, tandis que la procédure Fermat échoue, et en fait, ce sera toujours ainsi pour ce cas spécifique, i.e. quand $n = p^2$ avec p premier. Ceci est dû au fait que dans la procédure Fermat, le premier test s'effectue sur une valeur $x > \lfloor \sqrt{n} \rfloor$ et, comme nous l'avons déjà remarqué, quand n est le carré d'un nombre premier, il faut avoir $x = \lfloor \sqrt{n} \rfloor$.

2.3 Exemple 2

$n = 2pq$ où p et q sont des nombres premiers (excluant 2) et où $2p \cong q$ ou $2q \cong p$. Dans l'exemple, on génère d'abord p premier aléatoirement, d'une taille quelconque, puis on trouve q premier tel que $2q \cong p$. On constatera que la procédure *split_quad* trouve systématiquement la factorisation $(2q)(p)$ en une seule itération ($b = 1$), tandis que, ici encore, la procédure Fermat échoue. Le succès immédiat de *split_quad* s'explique par le fait que les facteurs trouvés sont *relativement proches* l'un de l'autre, alors que l'échec de la procédure Fermat est dû au fait que, pour des nombres n de cette forme, il est impossible de trouver deux facteurs de même parité dont le produit donne n.

>p:=nextprime(Rnd(30));q:=nextprime(floor(p/2+Rnd(10)));n:=2*p*q;
B:=100;split_quad(n,B);Fermat(n,B);

$p := 501261629479870548736450100477$

$q := 250630814739935274374819756167$

$n := 251263221188815027174400827708572070547794484607903580783318$

$B := 100$

[[1], 501261629479870548736450100477, 501261629479870548749639512334]

[[b], p, q]

2.3 Exemple 3

Nous voyons ici que si $n = pq$, avec p et q de parité contraire, *relativement proches* , mais où il existe p' et q' de même parité tels que $n = p'q'$, alors *split_quad* trouve immédiatement la factorisation pq, tandis que la procédure Fermat va trouver la factorisation $p'q'$ mais nécessitera plus d'itérations. Prenons $p = 101$, $q = 120$, donnant $n = 12120$.

> **split_quad(n,100);Fermat(n,100);**

[[1], 101, 120]

[[21], 60, 202]

Dans cet exemple, la procédure Fermat ne peut pas trouver la factorisation pq puisque ces facteurs sont de parité contraire; elle retourne toutefois la paire de facteurs de même parité qui sont, de toutes les combinaisons possibles, les plus rapprochés l'un de l'autre, et qu'elle trouve plus loin dans son exécution.

2.3 Exemple 4

Nous prenons ici $n = pq$ où p et q sont premiers et *relativement proches* .

```
> p:=nextprime(Rnd(30));q:=nextprime(p+Rnd(16));n:=p*q;
```

$p := 211299605127114594552921120421$

$q := 211299605127117110173146891317$

$n := 44647523126875083803253244003773335263738656752408756284457$

```
> split_quad(n,100);Fermat(n,100);
```

[[8], 211299605127114594552921120421, 211299605127117110173146891317]

[[4], 211299605127114594552921120421, 211299605127117110173146891317]

Nous constatons ici que les deux procédures trouvent rapidement la solution, mais que la procédure Fermat le fait en exactement deux fois moins d'itérations, et l'on peut facilement vérifier que c'est toujours ainsi pour ce cas. Rappelons que, pour la procédure *split_quad* , les valeurs b impaires correspondent à des solutions où les facteurs sont de parité contraire; elles sont donc inutiles à essayer quand le nombre à factoriser est le produit de deux nombres premiers. Remarquons aussi que les deux procédures ont toujours ce même comportement, quand le nombre à factoriser peut s'exprimer comme produit de deux facteurs de même parité, premiers ou pas. Par exemple, si $p = 46$, $q = 70$ alors:

```
> n:=46*70;split_quad(n,100);Fermat(n,100);
```

$n := 3220$

[[4], 46, 70]

[[2], 46, 70]

2.3 Exemple(s) 5

> n:=5^3+2^3; Fermat_kieme(n,[3,3],[0,10]);

n := 133

[[3, 0],[7, 19]]

> n:=7^3-4^3;

n := 279

> Fermat_kieme (n,[3,3],[0,10]);

[[3, 1], [3, 93]]

> n:=3111^6-143^6;

n := 906567326320430891312

Fermat_kieme (n,[6,6],[0,100]);

[[6, 1], [2968, 3054472123721128 34]]

> n:=3111231312311132 13^15+443542771459^15;

n :=

247733372744687120887590875789886038842801262257260597239291462828747446347843192389421678505264764249480512230782152908485446831305647473276248292333164946562886041634038461496398974610367177144303391172560753748892658347369607460122141876213763605212006334466 56

Fermat_kieme (n,[5,5],[0,100]);

[[5, 0],

[301159732747733032545013592766338797580368514626101 76,

822597929956995300273398095262923561986578382377412660434516579630144344095842981866782838111960076534561955940983727045602863271207656665664050982903708680794470631332107286019584480405508516634740590669164981]]

3.1 Exemple 1

>k:=15;a:=nextprime(Rnd(k)):b:=nextprime(Rnd(k)):c:=nextprime(Rnd(k)):d:=nextprime(Rnd(k)):e:=nextprime(Rnd(k)):f:=nextprime(Rnd(k)):g:=Rnd(5*k):clef:=[k,a,b,c,d,e,f,g];

$k := 15$

clef := [15, 481320635782561, 397473874329041, 776746133320907, 285722938497107, 221744908586753, 455309786807237, 4570391695941600884305716749604988340858129204579164537470194616431456403 18]

> msg:="petit message secret "; n:=encode(msg);

msg := "petit message secret "

n := 676541098185309042892568828589206059286485

> N:=Chiffre(n,clef);

N :=

2773533029605891512279089604916520956979940834982997109231645064185971812429352448138954947283729566598047445725128427400905460539332709 28096137

> M:=Dechiffre(N,clef);MSG:=decode(M);

M := 676541098185309042892568828589206059286485

MSG := "petit message secret "

3.2 Exemple 1

Cet exemple souligne le fait que, si on a une information partielle sur les facteurs p et q du modulus RSA, il est alors possible de retrouver un exposant secret plus grand qu'il n'est possible de le faire avec les attaques de Wiener et celle de de Weger. Ici, nos facteurs premiers ont une dizaine de chiffres chacun, et on va supposer la connaissance des deux premiers chiffres.

On définit d'abord deux fonctions, P et Q, qui retournent des bornes minimales pour p et q, selon n et selon la distance minimale entre ces facteurs, appelée Δ et estimée à partir de l'information connue.

```
> P:=(n,Delta)->-1/2*Delta+1/2*sqrt(Delta^2+4*n);
```

$$P := (n,\Delta) \rightarrow -\frac{1}{2}\Delta + \frac{1}{2}\sqrt{\Delta^2 + 4n}$$

```
> Q:=(n,Delta)->n/P(n,Delta);
```

$$Q := (n,\Delta) \rightarrow \frac{n}{P(n,\Delta)}$$

La fonction **sumPQ** retourne alors une valeur minimum pour $p+q$.

```
> sumPQ:=(n,Delta)->P(n,Delta)+Q(n,Delta);
```

$$sumPQ := P(n,\Delta) + Q(n,\Delta)$$

On définit maintenant les facteurs premiers, puis leur produit n. La valeur r sert dans l'approche de de Weger.

```
p:=nextprime(7134545511);q:=nextprime(9678912322);n:=p*q;r:=isqrt(n);
```

$p := 7134545521$

$q := 9678912329$

$n := 69054640605018628409$

$r := 8309912190$

Ensuite, on calcule Δ en fonction de l'information connue, soit les deux premiers chiffres de p et q.

> P0:=7100000000; Q0:=9600000000; Delta:=Q0-P0;

$P0 := 7100000000$

$Q0 := 9600000000$

$\Delta := 2500000000$

Puis on obtient une valeur minimum pour $p+q$, que l'on appelle s.

> s:=floor(evalf(sumPQ(n,Delta)));

$s := 16806801080$

Ici on calcule $\phi(n)$, puis son estimation W selon l'approche de Weger, puis on calcule X qui est l'estimation en fonction des informations connues.

> phi:=(p-1)*(q-1);W:=n+1-2*r;X:=n-s;

$\phi := 69054640588205170560$

$W := 69054640588398804030$

$X := 69054640588211827333$

On choisit maintenant un exposant secret d puis on calcule e son inverse modulo $\phi(n)$.On calcule aussi la quantité k telle que $ed - k\phi = 1$, et on affiche $k\_d = \frac{k}{d}$ pour fin de comparaison.

> d:=1512131;e:=d^(-1)mod(phi);k:=(e*d-1)/phi;k_d:=k/d;

$d := 1512131$

$e := 21591223149013578731$

$k := 472796$

$k\_d := \frac{472796}{1512131}$

Voici maintenant la liste des convergentes vers $\frac{e}{\phi}$. Le rapport $\frac{k}{d}$ s'y trouve toujours.

> x:='x':cfrac(e/phi,10,x):x;

$$\left[0, \frac{1}{3}, \frac{5}{16}, \frac{111}{355}, \frac{116}{371}, \frac{3939}{12598}, \frac{472796}{1512131}, \frac{21591223149013578731}{69054640588205170560}\right]$$

Voici maintenant la liste des convergentes vers $\frac{e}{n}$, selon l'approximation de Wiener. Le rapport $\frac{k}{d}$ ne s'y trouve pas.

> x:='x':Wiener:=cfrac(e/n,10,x):x;

$$\left[0, \frac{1}{3}, \frac{5}{16}, \frac{111}{355}, \frac{116}{371}, \frac{3939}{12598}, \frac{189188}{605075}, \frac{193127}{617673}, \frac{4437982}{14193881}, \ldots\right]$$

Regardons maintenant la liste des convergentes vers $\frac{e}{W}$, selon l'approximation de Weger. Le rapport $\frac{k}{d}$ ne s'y trouve pas non plus.

> x:='x':deWeger:=cfrac(e/W,10,x):x;

$$\left[0, \frac{1}{3}, \frac{5}{16}, \frac{111}{355}, \frac{116}{371}, \frac{3939}{12598}, \frac{464918}{1486935}, \frac{16276069}{52055323}, \frac{33017056}{105597581}, \ldots\right]$$

Toutefois, si nous utilisons notre approximation X, obtenue à partir d'une information, on constate que le rapport cherché $\frac{k}{d} = \frac{472796}{1512131}$ apparaît alors dans la liste des convergentes.

> **x:='x':S:=cfrac(e/X,10,x):x;**

$$\left[0, \frac{1}{3}, \frac{5}{16}, \frac{111}{355}, \frac{116}{371}, \frac{3939}{12598}, \frac{468857}{1499533}, \frac{472796}{1512131}, \frac{6615205}{21157236}, \ldots\right]$$

Annexe E

Le processus RSA

Le processus RSA

Bob

- Génère deux grands nombres premiers de même taille, p et q, à l'aide d'un générateur de nombres aléatoires et d'un test de *primalité* (par exemple le test de Miller-Rabin ([12], 4.24))
-
- Calcule $n = pq$ et $\phi(n) = (p-1)(q-1)$
- Choisit un nombre aléatoire b ($1 < b < \phi(n)$) et s'assure que pgcd($b, \phi(n)$) = 1 à l'aide de l'algorithme d'Euclide.
- Calcule $a = b^{-1} \bmod \phi(n)$ à l'aide de l'algorithme étendu d'Euclide ([12], 2.107)
- Publie n et b dans un répertoire

Alice

- Consulte le répertoire public pour obtenir les clefs n et b publiées par Bob
- Compose un message $x \in \mathbf{Z}_n$, un entier positif, à l'intention de Bob. Si $x \geq n$, alors elle scinde son message en blocs $x = x_1 x_2 \ldots x_m$ où $x_i < n$ et traite chaque bloc séparément
- Calcule $y = x^b \bmod n$ (fonction de chiffrement) et l'envoie à Bob sur un canal peu sûr

Bob

- Reçoit y puis, avec la clef secrète a connue de lui seul, calcule $y^a \bmod n = x$ (fonction de déchiffrement)