Analyse des algorithmes finalistes concourant pour le futur standard AES

Par

Jean-Philippe Gaulier
jean-philippe.gaulier@unilim.fr

Conservatoire National des Arts et Métiers
Centre Régional Associé de Limoges

Mémoire de synthèse
soumis dans le cadre d'un probatoire
en vue de l'acquisition d'un diplôme en

Ingénierie et Intégration Informatique
Systèmes d'Information

Jury composé de
Madame M.-C. Costa, Présidente
Monsieur M. Giry
Monsieur P. Jeulin
Monsieur M.Rybowicz

« **Cryptographie** *n.f.* Procédé (signes conventionnels, modification de l'ordre, de la disposition des signes, remplacement des signes...) permettant de rendre un message inintelligible, de protéger des données. »

Le Petit Robert, éd. 1993

« Vous rigolez ? Qu'est ce qui vous fait penser qu'un utilisateur dont la vie privée est concernée voudra bien croire que les communications sont sécurisées ? Aucun organisme gouvernemental ne peut faire foi lorsque l'on parle de ces choses. »

David Zimmerman, *commentaire sur AES adressé au NIST*

« L'organisme de sécurité des systèmes d'information de l'Agence Nationale de Sécurité encourage fortement votre proposition de développement d'un FIPS pour un algorithme de chiffrement avancé utilisant un processus public et accueille la possibilité d'apporter son avis. »

NSA, *commentaire sur AES adressé au NIST*

« Ce sont des choses sérieuses dont nous parlons. Tout algorithme proposé en 1997 ne sera pas approuvé avant l'an 2000. Ce standard devra être maintenu pendant une période de vingt à trente ans, ce à quoi il faut ajouter au moins dix années supplémentaires pour les administrations et vingt de mieux pour les données devant être sécurisées. Ces faits établis, nous sommes donc en train d'essayer d'établir quelle sera la sécurité en 2060. Je ne suis pas capable d'estimer la sécurité pour les dix années à venir, alors que puis-je dire pour 60 ans ? La seule option satisfaisante consiste à rester prudent. »

Bruce Schneier, *session de travail de l'appel à candidature pour l'AES du 15/04/1997*

Résumé

La Cryptographie, tout comme le savoir, a commencé avec l'écriture. En 1998, le National Institute of Standards and Technology (NIST) des états-Unis d'Amérique a lancé un appel pour la mise en place d'un nouveau modèle de chiffrement, afin de remplacer le Data Encryption Standard, qui se faisait vieillissant et moins sûr. Pendant des mois, la communauté cryptographique mondiale a partagé ses expériences et son savoir à ce sujet. De cet effort résulte le *Advanced Encryption Standard*, issu du projet belge nommé Rijndael. Ce travail vous invite dans une courte histoire de la cryptographie à travers les âges, avant de décrire les cinq algorithmes finalistes de cette compétition du NIST qui a finalement sélectionné Rijndael. Nous terminerons par un court état des lieux sur la sécurité d'AES dans son implémentation actuelle.

Abstract

Cryptography, just like knowledge, began with writing. In 1998, the U.S. National Institute of Standards and Technology called for a new encryption method, the Data Encryption Algorithm, then in use, being deemed too weak. For months, members of the cryptography community all around the world shared their experience and thoughts on the subject. This effort resulted in the Advanced Encryption Standard, based on a Belgian project named Rijndael. This work invites you to a brief history of cryptography throughout the ages, before describing the five algorithms selected for the final stage of the NIST competition that lead to AES. Finally, more details on the security of AES in its current implementation are provided.

Remerciements

Je tiens avant tout à remercier ma compagne pour sa formidable patience durant les innombrables heures que j'ai passé à la recherche et la lecture d'articles, l'échange de courriers électroniques, ainsi que la rédaction de ce document.

Un merci tout particulier à Stéphane Dodeller, pour son soutien constant et sans condition, pour comprendre ce que seuls des gens qui passent plus de douze heures par jour derrière un micro peuvent comprendre, ainsi que pour son aide à mon démarrage dans LaTeX.

Un grand merci à tous mes relecteurs, je pense entre autre ici à Régis Foinet et Muriel Baluteau.

Un chaleureux merci à éric Filiol, Chef du Laboratoire de virologie et de cryptologie de l'école Supérieure et d'Application des Transmissions, pour sa considération et son écoute.

Enfin, un merci aux contributeurs anonymes qui archivent et classent les documents sur Internet, que ce soit sur les répertoires officiels comme au NIST, ou sur des sites plus anonymes, car sans tout cela, ce travail n'aurait pu aboutir.

Table des matières

Table des figures

Liste des tableaux

Introduction

Si, tout comme le précise ce mémoire, la cryptographie concerne l'humanité depuis longtemps, elle reste une technique d'initié et l'on pourrait la restreindre à une certaine caste. En effet, si l'on s'attarde à faire un parallèle, par exemple, avec l'automobile, beaucoup de personnes emploient aujourd'hui une voiture, sans avoir la moindre idée du fonctionnement interne du moteur, de ce qu'il y a sous le capot. Il en va de même pour le chiffrement et les algorithmes associés. Dans le monde très fermé de l'informatique, l'usage de protocoles chiffrés, comme IPSEC, TLS ou autres, rassure, car aucune personne normalement constituée ne pourrait espionner votre trafic, les données transportées étant chiffrées, contrairement au *plaintext data*, flot de texte en clair. D'un coup de baguette féérique, on solutionne tous les problèmes des PME/PMI, avec l'option magique, les mots secrets, la formule. Quid de la sécurité véritable ? Ce mémoire se veut avant tout une étude personnelle, posée, de la réalité des instances mises en jeu, de l'importance du choix des algorithmes, de leur valeur dans notre société.

Au travers de cette étude, la mise en place d'un nouvel algorithme de chiffrement, les débats qui en sont issus, la proposition d'adoption puis l'instauration, on constate une communauté active, vivante, scientifique. On ne peut cependant que faire confiance à ce cercle restreint, qui se compose tout au plus de quelques milliers d'individus. étonnant, également, la possible acceptation d'un organisme national, dépendant du gouvernement des états-Unis d'Amérique, comme centre mondial et moteur en la matière. Plus intrigant, l'absence de « nouvelles démocraties » parmi les candidats. Qu'en est il de la Russie, entre autres, avec son algorithme *NUSH* ?

Nous sommes donc conviés à une introduction au monde de la cryptographie au travers d'une frise historique relatant les grands moments de sa vie. épopée nous amenant jusqu'en 1997, année où le *NIST* décide de renouveller son algorithme vieillissant, le DES.

Des experts vont dès lors s'affronter pendant près de trois années afin de savoir lequel des quinze puis des cinq algorithmes de chiffrement présentés il faut adopter pour arriver à l'*AES*. Cette sélection doit-elle amener un choix unique ou fournir plusieurs alternatives, c'est ce que nous verrons. Au travers de l'étude de critères de sélection comme la sécurité, l'implémentabilité, la manipulation de la clef, la souplesse et la flexibilité, nous tenterons de discerner pourquoi le NIST a finalement choisi Rijndael comme unique réponse. Réponse aujourd'hui critiquée par des travaux comme ceux de monsieur éric Filiol, mettant en péril l'invulnérabilité espérée d'AES pour les soixantes années à venir. Ces résultats n'étant encore qu'à un stade de discussion, nous en resterons aux faits. AES est aujourd'hui la solution en phase d'être adoptée par une majorité d'organismes, et nous allons tenter de comprendre pourquoi.

En d'autres termes, aucune révélation ne sera apportée par ce travail, qui reste, somme toute, la modeste compilation de mes efforts. Il ne peut en aucun cas avoir la prétention de se comparer à des travaux de personnes qui dédient leur carrière à cette science. Quoi qu'il en soit, et bien qu'il nécessite

des fondements solides en mathématiques, l'univers de la cryptographie et de ses secrets est passionnant. J'espère que ces quelques pages seront à même de retranscrire l'intérêt que j'ai découvert pour cette matière.

Chapitre 1

Petite histoire de la cryptographie

L'histoire est une suite d'événements mis bout à bout. Sans les acteurs qui créent cette dynamique, il ne serait possible de voir une progression. Les personnes suivantes ont toutes eu une incidence dans la cryptographie que l'on connaît aujourd'hui. Voici donc un rapide descriptif des faits marquants et des hommes qui ont permis cette évolution.

Vers 1900 av. J.-C., un scribe égyptien a employé des hiéroglyphes non conformes à la langue correcte dans une inscription. Kahn le qualifie de premier exemple documenté de cryptographie écrite.

Quatre siècles plus tard, vers 1500 av. J.-C., une tablette mésopotamienne contient une formule chiffrée pour la fabrication de vernis pour les poteries.

Cinq siècles avant notre ère, des scribes hébreux mettant par écrit le livre de Jérémie ont employé un simple chiffre de substitution connu sous le nom d'Atbash. C'était un des quelques chiffres hébreux de cette époque, avec Albam et Atbah.

En 487 av. J.-C., les grecs emploient un dispositif appelé la scytale, un bâton autour duquel une bande longue et mince de cuir était enveloppée et sur laquelle on écrivait le message. Le cuir était ensuite porté comme une ceinture par le messager. Le destinataire avait un bâton identique permettant d'enrouler le cuir afin de déchiffrer le message.

L'historien grec Polybe (env. 200-125 av. J.-C.) invente le carré de Polybe, dont s'inspireront plus tard bien des cryptosystèmes.

Jules César employait une substitution simple avec l'alphabet normal (il s'agissait simplement de décaler les lettres de l'alphabet d'une quantité fixe) dans les communications du gouvernement. Ce chiffre était moins robuste qu'Atbash, mais à une époque où très peu de personnes savaient lire, cela suffisait. César écrivait aussi parfois en remplaçant les lettres latines par les lettres grecques.

Le *Kama-sutra* est un texte écrit au 5e siècle par le brahman Vatsayayana, mais fondé sur des manuscrits du 4e siècle avant J.-C. Le Kama-sutra recommande que les femmes apprennent 64 arts, entre autres cuisiner, s'habiller, masser et élaborer des parfums. La liste comprend aussi des domaines

moins évidents, comme la prestidigitation, les échecs, la reliure et la tapisserie. Le numéro 45 de la liste est le mlecchita-vikalpa, l'art de l'écriture secrète, qui doit leur permettre de dissimuler leurs liaisons.

Abu Bakr ben Wahshiyya publie, en 855, plusieurs alphabets secrets utilisés à des fins de magie, dans son livre *Kitab shauk almustaham fi ma'arifat rumuz al aklam*, le livre de la connaissance longuement désirée des alphabets occultes enfin dévoilée.

Au 9è siècle, Abu Yusuf Ya'qub ibn Is-haq ibn as-Sabbah Oòmran ibn Ismaïl al-Kindi rédige le plus ancien texte connu décrivant la technique de décryptement appelée analyse des fréquences.

à partir de 1226 une timide cryptographie politique apparaît dans les archives de Venise, où des points ou des croix remplacent les voyelles dans quelques mots épars.

En 1250, Roger Bacon a non seulement décrit plusieurs chiffres, mais il a aussi écrit : « Il est fou celui qui écrit un secret de toute autre manière que celle qui le soustrait à la connaissance du vulgaire ».

Gabriel de Lavinde compose, en 1379, un recueil de clefs, dont plusieurs combinent code et substitution simple. En plus d'un alphabet de chiffrement, souvent avec des nulles, on trouve un petit répertoire d'une douzaine de noms communs et de noms propres avec leurs équivalents en bigrammes. C'est le premier exemple d'un procédé qui devait prévaloir pendant 450 ans en Europe et en Amérique : le nomenclateur.

En 1392, dans un ouvrage intitulé *L'équatorial des Planètes*, qui décrit le fonctionnement d'un instrument astronomique, Geoffrey Chaucer, a incorporé six courts cryptogrammes écrits de sa propre main.

La science arabe en matière de cryptologie est exposée dans la subh al-a sha, une énorme encyclopédie en 14 volumes datant de 1412, écrite pour fournir à la bureaucratie une connaissance exhaustive de toutes les principales branches du savoir. Son auteur, qui vivait en Egypte, était Abd Allah al-Qalqashandi. La section intitulée *De la dissimulation des informations secrètes dans les lettres* comporte deux parties, l'une traitant des représentations symboliques et du langage convenu, l'autre des encres invisibles et de la cryptologie.

Léon Battista Alberti invente et publie le premier chiffre polyalphabétique en 1466. Il conçoit un cadran chiffrant pour simplifier le processus. Cette classe de chiffres n'a pas été apparemment cassée jusqu'aux années 1800. Alberti a aussi écrit largement sur l'état contemporain des connaissances sur les chiffres, en plus de sa propre invention. Ces chiffres polyalphabétiques étaient beaucoup plus robustes que le nomenclateur qu'utilisaient les diplomates de l'époque. Alberti inventa aussi le surchiffrement codique. Mais le génie d'Alberti était trop en avance sur son temps et ce n'est que 400 ans plus tard, vers la fin du dix-neuvième siècle, que les principales puissances mondiales commencèrent à surchiffrer leurs codes par des procédés bien plus simples.

Le premier grand cryptanalyste européen fut peut-être Giovanni Soro, nommé secrétaire chiffreur en 1506. Il devint secrétaire du chiffre de Venise. Le Vatican lui-même testa ses chiffres sur Soro, qui les perça à jour une première fois. Le Pape envoya d'autres textes chiffrés à Soro afin de savoir si le meilleur cryptanalyste pouvait battre son chiffre. Soro renvoya les textes en écrivant qu'il n'avait pas réussi à les

déchiffrer mais on ne sut jamais s'il avait dit la vérité, ou s'il avait menti pour pouvoir décrypter sans difficulté tout message émanant des autorités pontificales...

Jean Trithème a écrit en 1518 le premier livre imprimé sur la cryptologie. Il a inventé un chiffre stéganographique dans lequel chaque lettre est représentée par un mot. La série résultante de mots ressemble à une prière. Il a aussi décrit des chiffres polyalphabétiques sous la forme désormais standard de tables de substitution rectangulaires.

Jérôme Cardan invente autour de 1550 le premier procédé autoclave, mais ce système est imparfait et c'est finalement un autre procédé qui porte son nom. La grille de Cardan consiste en une feuille de matériau rigide dans laquelle ont été découpées, à des intervalles irréguliers, des fenêtres rectangulaires de la hauteur d'une ligne d'écriture et de longueur variable. Le chiffreur écrit le texte dans les fenêtres, puis retire le cache et comble les espaces vides avec un texte anodin. Le destinataire pose la même grille sur le texte crypté pour lire le message caché.

Giovan Batista Belaso fait paraître, en 1553, un petit livre intitulé *La cifra del Sig. Giovan Batista Belaso*. Il y proposait, pour le chiffrement en substitution polyalphabétique, l'emploi de clefs littérales, faciles à garder en mémoire et à changer. Il les appelait « mot de passe ». Les clefs littérales furent immédiatement adoptées et l'innovation de Belaso est à l'origine de certains systèmes actuels très complexes où plusieurs clefs - et non pas une seule - sont utilisées et changées de façon irrégulière. Il a également écrit « De Futivis Literarum Notis ». Ces quatre livres, traitant respectivement des chiffres anciens, des chiffres modernes, de la cryptanalyse, des caractéristiques linguistiques qui favorisent le déchiffrement, représentent la somme des connaissances cryptologiques de l'époque. Parmi les procédés modernes, dont beaucoup sont de son invention, apparaît la première substitution bigrammatique : deux lettres sont représentées par un seul symbole. Il inventa aussi le premier chiffre polyalphabétique. Il fut le premier à classer les deux principes cryptographiques majeurs : la substitution et la transposition.

Blaise de Vigenère écrit son *Traicté des chiffres ou secrètes manières d'escrire* en 1585. Il présente entre autres un tableau du type Trithême, que l'on dénomme aujourd'hui à tort carré de Vigenère. On considéra longtemps ce chiffre comme indécryptable, légende si tenace que même en 1917, plus de cinquante ans après avoir été cassé, le Vigenère était donné pour « impossible à déchiffrer » par la très sérieuse revue *Scientific American*.

Sir Francis Bacon, qui est peut-être William Shakespeare, est l'inventeur d'un système stéganographique qu'il exposa dans *De dignitate et augmentis scientiarum* en 1623. Il appelait son alphabet bilitère, car il utilisait un arrangement des deux lettres A et B en groupes de cinq.

Antoine Rossignol et son fils Bonaventure élaborent le Grand Chiffre de Louis XIV en 1691. Il tomba en désuétude après la mort de ses inventeurs et ses règles précises furent rapidement perdues. Le grand Chiffre était si robuste qu'on était encore incapable de le lire à la fin du dix-neuvième siècle, jusqu'à Bazeries.

Dans les années 1790, Thomas Jefferson invente son cylindre chiffrant, si bien conçu qu'après plus d'un siècle et demi de rapide progrès technique, il était encore utilisé. C'était certainement le moyen de chiffrement le plus sûr de l'époque, et pourtant il fut classé et oublié. Il fut réinventé en 1891 par Etienne Bazeries, qui ne parvint pas à le faire adopter par l'armée française. L'armée américaine mit en service

un système presque identique en 1922.

Charles Wheatstone, un des pionniers du télégraphe électrique, invente en 1854 le chiffre Playfair, du nom de son ami Lyon Playfair qui a popularisé ce chiffre.

La même année, soit 269 ans après sa publication, Charles Babbage casse le chiffre de Vigenère, mais sa découverte resta ignorée, car il ne la publia pas. Ce travail ne fut mis en lumière qu'au vingtième siècle, lors de recherches effectuées sur l'ensemble des papiers de Babbage.

Friedrich W. Kasiski publie, en 1861, *Die Geheimschriften und die Dechiffrierkunst*, les chiffres et l'art du déchiffrement, qui donne la première solution générale pour le déchiffrement d'un chiffre polyalphabétique à clefs périodiques marquant ainsi la fin de plusieurs siècles d'invulnérabilité du chiffre de Vigenère.

1891. Le commandant étienne Bazeries produit son cryptographe cylindrique. Il était composé de vingt disques portant chacun vingt-cinq lettres. Il ne sera jamais employé par l'armée française. Bazeries fut aussi le premier à déchiffrer le Grand chiffre de Louis XIV.

Gilbert S. Vernam, travaillant pour AT&T, invente en 1917 une machine de chiffre polyalphabétique pratique capable d'employer une clef qui est totalement aléatoire et ne se répète jamais - un masque jetable. C'est le seul chiffre, dans nos connaissances actuelles, dont on a prouvé qu'il était indécryptable en pratique et en théorie. Ce procédé ne fut cependant jamais utilisé par l'armée car il exigeait de devoir produire des millions de clefs différentes (une par message), ce qui est impraticable. Par contre, il fut utilisé par les diplomates allemands dès 1921.

Le système ADFGVX a été mis en service par les Allemands à la fin de la première guerre mondiale. Il a été cassé par le lieutenant français Georges Painvin en 1918.

Cette même année, Arthur Scherbius fait breveter sa machine à chiffrer Enigma. Le prix d'un exemplaire s'élevait à 20 000 livres sterling actuelle. Ce prix sembla décourager les acheteurs potentiels.

Boris Caesar Wilhelm Hagelin (1892-1983) propose à l'armée suédoise la machine B-21, qui fut pendant une décennie la machine la plus compacte capable d'imprimer des messages chiffrés. Pendant la seconde guerre mondiale, les alliés fabriquèrent une autre machine de Hagelin, la Hagelin C-36 (appelée M-209 aux états-Unis), à 140 000 exemplaires. Après la guerre, Boris Hagelin créa à Zoug, en Suisse, Crypto AG, qui est aujourd'hui encore l'un des principaux fabricants d'équipements cryptographiques.

Lester S. Hill publie son article « Cryptography in an Algebraic Alphabet » [1]. Il y décrit le chiffre qui porte son nom. C'est un chiffre polygraphique où l'on utilise des matrices et des vecteurs.

Herbert O. Yardley publie *The American Black Chamber*, en 1931, un livre célèbre sur la cryptologie. Il déchiffre entre autres les codes japonais (avant leur machine PURPLE).

La machine Enigma ne fut pas un succès commercial mais elle fut reprise et améliorée pour devenir la machine cryptographique de l'Allemagne nazie pendant la seconde guerre mondiale. Elle a été cassée

par le mathématicien polonais Marian Rejewski, qui s'est fondé seulement sur un texte chiffré et une liste des clefs quotidiennes obtenues par un espion. Pendant la guerre, les messages furent régulièrement décryptés par Alan Turing, Gordon Welchman entre autres à Bletchley Parc, en Angleterre, à l'aide des premiers ordinateurs.

William Frederick Friedman, plus tard honoré comme le père de de la cryptanalyse américaine, à la tête de son équipe du Signal Intelligence Service (S.I.S.), réussit, en 1940, le déchiffrement de la machine japonaise PURPLE. Avec sa femme, il s'intéressa beaucoup aux chiffres shakespeariens, et, pendant la prohibition, ils déchiffrèrent les codes des trafiquants.

Au début des années 1970, Horst Feistel a mené un projet de recherche à l'IBM Watson Research Lab qui a développé le chiffre Lucifer, qui inspira plus tard le chiffre DES et d'autres chiffres.

Whitfield Diffie et Martin Hellman publient, en 1976, *New Directions in Cryptography*, introduisant l'idée de cryptographie à clef publique. Ils donnent une solution entièrement nouvelle au problème de l'échange de clefs. Ils avancent aussi l'idée d'authentification à l'aide d'une fonction à sens unique. Ils terminent leur papier avec l'observation suivante : « L'habileté dans la cryptanalyse a toujours été lourdement du côté des professionnels, mais l'innovation, en particulier dans la conception des nouveaux types de systèmes cryptographiques, est venue principalement d'amateurs. »

Novembre 1976, DES, pour Data Encryption Standard (« standard de cryptage de données »), est un algorithme très répandu à clef privée dérivé du chiffre Lucifer de Feistel (de chez IBM) dans sa version à 64 bits. Il sert à la cryptographie et l'authentification de données. Il a été jugé si difficile à percer par le gouvernement des états-Unis qu'il a été adopté par le département de la défense des états-Unis qui a contrôlé depuis lors son exportation. Cet algorithme a été étudié intensivement et est devenu l'algorithme le mieux connu et le plus utilisé dans le monde à ce jour. Bien que DES soit solide, une alternative nommée le « triple-DES » a été implémentée, qui n'est rien d'autre que l'algorithme DES appliqué trois fois, avec trois clés privées différentes.

RSA signifie Rivest-Shamir-Adleman, en l'honneur de ses trois inventeurs : Ron Rivest, Adi Shamir et Leonard Adleman qui l'ont inventé en 1977. Le brevet de cet algorithme appartient à la société américaine RSA Data Security, qui fait maintenant partie de Security Dynamics, aux Public Key Parteners, (PKP à Sunnyvale, Californie, états-Unis) qui possèdent les droits en général sur les algorithmes à clé publique. RSA est un algorithme à clé publique qui sert aussi bien à la cryptographie de documents qu'à l'authentification. Grâce à sa structure asymétrique et au fait qu'il était très solide, l'algorithme RSA est devenu un standard de facto dans le monde.

1990. Xuejia Lai et James Massey publient *A Proposal for a New Block Encryption Standard*, article qui propose IDEA — International Data Encryption Algorithm, un Algorithme de Chiffrement des Données Internationales — pour remplacer le DES. IDEA emploie une clef de 128 bits et utilise des opérations convenant bien à tout type d'ordinateurs, permettant donc une programmation plus efficace. Il s'agit d'un des meilleurs algorithmes de chiffrement, si ce n'est le meilleur. Personne n'a dévoilé à ce jour avoir cassé d'une manière ou d'une autre le moindre bloc de texte chiffré par IDEA. Il est actuellement exploité par la société Mediacrypt.

Charles H. Bennett et Gilles Brassard publient en 1990 leurs résultats expérimentaux sur la Cryptographie Quantique, qui emploie des photons pour communiquer un flot de bits qui serviront de clefs pour un cryptage de type Vernam (ou d'autres utilisations). En supposant exactes les lois de la mécanique quantique, la Cryptographie Quantique offre non seulement le secret, mais permet aussi de savoir si la ligne a été écoutée. Cependant, la CQ exige actuellement un câble en fibres optiques entre les deux correspondants.

Phil Zimmermann sort sa première version de PGP (Pretty Good Privacy) en 1991, en réponse à la menace du FBI d'exiger l'accès au message en clair des citoyens. PGP offre une haute sécurité au citoyen et cela gratuitement. Depuis, l'IETF a éditée en 1998 une rfc, nommée OpenPGP, sous le numéro 2440.

Nicolas Gisin et son équipe distribuent en 1995 des clefs secrètes à l'aide d'un câble de fibres optiques de 25 kilomètres sous le lac Léman en codant les q-bits par la polarisation de photons (cryptographie quantique). La distance est le prochain obstacle que devront franchir les chercheurs, car le dispositif ne peut excéder 50 à 60 km, selon leurs estimations.

Août 1999 : LIX. 11 sites répartis dans 6 pays factorisent le premier nombre ordinaire de 155 chiffres décimaux (512 bits). Un tel nombre aurait pu servir de clef dans un système de chiffrement moderne de type RSA, qui est utilisé dans le commerce électronique. Un tel record remet en question l'utilisation de clefs trop petites dans de tels systèmes.

Chapitre 2

Origine d'AES

Le 2 janvier 1997, l'Institut National des Standards et Technologies du Département du Commerce des états-Unis d'Amérique annonce le développement d'une norme nationale pour le traitement de l'information nommée Advanced Encryption Standard, ou Norme de Chiffrement Avancé.

Apparu fin 1976, le DES (Data Encryption Standard) a alors près de vingt ans d'âge. Le NIST, lors de la dernière révision de sécurité de cet algorithme, en 1993, insère la note suivante dans le dossier :

« à la prochaine révision (1998), l'algorithme spécifié par ce standard aura plus de vingt ans. Le NIST se doit de considérer une alternative apportant un niveau supérieur de sécurité. Cette alternative devra être procédée en remplacement de ce standard lors de sa prochaine révision. »

Le NIST, conscient qu'un tel changement ne pourra s'opérer en une fois, annonce alors la fin du DES en decrescendo, au fur et à mesure que son remplaçant, l'AES, prendra place au sein de la communauté cryptographique et dans l'industrie.

L'organisme sollicite alors l'opinion publique, les manufacturiers, les organismes de standardisation, ainsi que les utilisateurs gouvernementaux afin d'obtenir leur point de vue dans le développement d'AES. Celui-ci devra s'orienter autour de quelques axes définis préalablement. L'algorithme choisi devra :

- être public, utilisant un chiffrement symétrique par bloc ;
- avoir une clef dont la longueur pourra être incrémentée autant que nécessaire ;
- garder la capacité d'être implémenté de manière matérielle ainsi que logicielle ;
- nécessairement être disponible librement, selon les termes inscrits dans les licences de l'American National Standards Institute (ANSI).

Tout algorithme présenté répondant à ces exigences sera alors jugé sur les critères suivant :

- la sécurité, comme par exemple l'effort effectué sur la cryptanalyse,
- l'efficacité informatique,
- l'utilisation de la mémoire,
- l'adéquation matérielle et logicielle,
- la simplicité,
- la flexibilité,
- le respect des licences.

Un ensemble de données nécessaire à la réalisation d'un dossier de postulation est également présenté, afin que les candidats intéressés puissent d'ores et déjà rassembler les pièces requises. Les intéressés doivent fournir :

- les spécifications complètes de l'algorithme, incluant les équations mathématiques, tables et paramètres adéquats,
- une implémentation logicielle ainsi que son code source programmé en langage C, pouvant être compilé sur un ordinateur personnel,
- ce programme permet la comparaison entre les diverses propositions, autant en termes de performances que d'utilisation mémoire,
- un rapport fournissant l'efficience tant au niveau matériel que logiciel,
- un exemple de texte fournit en clair et sa forme chiffrée,
- si l'algorithme présenté est lié à une quelconque licence ou à un brevet dans tout ou partie, tous ces éléments devront être joints,
- une analyse des attaques connues permet de s'assurer ou non de la solidité,
- pour finir, chaque concurrent devra faire une critique des atouts et limites de sa proposition.

Le 15 avril 1997, après avoir reçu trente-trois commentaires sur sa proposition, le NIST décide de valider son projet et d'en faire la base de travail de l'AES. C'est la première réunion de travail qui conduira à l'appel à candidatures le 12 septembre 1997.

Cette réunion de travail fait ressortir les points essentiels que le NIST désirerait voir apparaître dans l'appel à candidatures final :

- tout d'abord, il confirme que toutes les soumissions seront rendues publiques, ce qui permettra l'analyse des algorithmes par tout un chacun,
- les auteurs préféreront un chiffrement par bloc, qui traite les données par bloc d'octets, à un chiffrement par flot, qui s'applique bit à bit sur les données, afin de rester compatible avec DES,
- ces blocs pourront dépasser 64 bits, 128 bits semblant s'imposer comme un standard commun,
- les implémentations devront pouvoir être effectuées à la fois de manière logicielle et matérielle,
- chaque proposition ne pourra faire l'objet de taxe d'utilisation (royalty-free). Bien que les algorithmes sous licence soient acceptables, une préférence ira vers des licences libre de droit,
- afin de limiter les risques de sécurité, une explication mathématique rationnelle est fortement recommandée,
- en matière de tests, il est demandé aux participants de fournir une implémentation de référence en langage Java et une, optimisée, en C. Les tests d'utilisation de la mémoire s'effectueront sur une même et unique plateforme, dotée d'un processeur Pentium Pro,
- le nombre de cycles et la taille de la clef devra être fixe,
- pour finir, le NIST pense la durée de vie d'AES estimée entre 20 et 30 années.

Le 12 septembre 1997, comme prévu dans le déroulement initial, l'appel à candidatures est lancé. Il fait preuve d'un souci de résultat et de sécurité dès le départ en expliquant aux candidats que l'AES devra être plus efficace et plus résistant que ne l'est le triple DES.

Les candidats doivent expliquer de manière mathématique leurs algorithmes et les soumettre au test de MCT et KAT [MCT & KAT] afin de s'assurer de l'exactitude de leur implémentation.

Le premier tour de la compétition devant désigner l'algorithme qui sera finalement choisi pour AES s'étend sur la période d'août 1998 à avril 1999. Deux réunions seront organisées :

TAB. 2.1 – Présentation des candidats par référent et pays d'origine.

Nom	Soumis par	Pays
CAST256	Entrust	Canada
Crypton	Future Systems	Corée
Deal	Outter Bridge	Canada
DFC	ENS-CNRS	France
E2	NTT	Japon
Frog	TecApro	Costa Rica
HPC	Schrœppel	USA
LOKI97	Brown, Piprzyk, Seberry	Australie
Magenta	DeutscheTelekom	Allemagne
Mars	IBM	USA
RC6	RSA	USA
Rijndael	Daemen, Rijmen	Belgique
Safer	Cylink	USA
Serpent	Anderson, Biham, Knudsen	GB, Israël, Norvège
Twofish	Counterpane	USA

La première conférence, AES1, se déroule du 20 au 22 août 1998, à Ventura en Californie. Elle présente les quinze algorithmes retenus par la commission comme éligibles (Tableau 2.1) et en rejette six, jugés incomplets ou incorrects (Tableau 2). La communauté internationale est très représentée.

TAB. 2.2 – Candidats non retenus.

Algorithme	Soumis par
GEM	Lance Gharat
RAINBOW	Samsung Advanced Institute of Technology (S.A.I.T.)
Simple	Richard Frank
TMD	Jonathan Stiebel (de MobileSafe LLC)
Vobach	Technology Design Automations Systems, Inc.
WICKER'98	LAN Crypto, Inc. (representé par Anatoly)

La deuxième conférence, AES2, prend place à Rome, en Italie, les 22 et 23 mars 1999, alors que s'approche la fin de cette première étape.

à l'issue d'AES2, l'annonce est faite que seulement cinq algorithmes ou moins seront présents pour le deuxième tour. Un sondage est soumis aux 180 conférenciers présents leur demandant quels candidats ils aimeraient voir apparaître lors du prochain tour de sélection. 104 réponses favorables sont fournies, elles sont récapitulées dans le tableau 2

Le tour numéro deux commence en août 1999 pour se terminer en mai 2000. Il révèle enfin la *shortlist* qui composera l'ultime épreuve. Sans grande surprise, on retrouve les choix précédemment évoqué, à savoir :

- Rijndael
- RC6
- Mars
- Twofish

TAB. 2.3 – Vœux de présence des algorithmes au second tour.

	Pas de Réponse	OUI	?	NON	Abs.	RANG
Rijndael	7	77	19	1	76	1
RC6	4	79	15	6	73	2
Twofish	9	64	28	3	61	3
MARS	5	58	35	6	52	4
Serpent	6	52	39	7	45	5
E2	11	27	53	13	14	6
CAST-256	12	16	58	18	-2	7
SAFER+	13	20	47	24	-4	8
DFC	12	22	43	27	-5	9
Crypton	14	16	43	31	-15	10
DEAL	10	1	22	71	-70	11
HPC	12	1	13	78	-77	12
MAGENTA	9	1	10	84	-83	13
Frog	11	1	6	86	-85	14
LOKI97	10	1	7	86	-85	14

– Serpent

Ce choix a tout d'abord été effectué sur l'aspect sécurité. En effet, aucune attaque n'a pu être déterminée envers chacun des finalistes. Même s'il a été fait preuve pour certains qu'en diminuant le nombre de tours, des faiblesses apparaissaient, cela ne rentrait pas en compte dans les propositions faites.

Durant cette troisième et dernière conférence qui se déroule à New York, le NIST explique qu'il n'y aura qu'un seul gagnant et aucun algorithme de secours ne sera choisi. Ce choix s'explique par la lourdeur d'implémentation par les constructeurs d'une deuxième solution, qui de plus resterait en sommeil. Implémenter un deuxième algorithme aurait impliqué des coûts importants. Il a donc été décidé de n'utiliser, comme par le passé, qu'une seule solution, tout en vérifiant régulierement la sécurité offerte.

Chapitre 3

Définition et concepts

I Le chiffrement par bloc

Le chiffrement par bloc s'oppose à celui par flot. Contrairement à ce dernier, il ne chiffre pas bit à bit, mais découpe des blocs entiers de bits pour les passer à la moulinette des algorithmes. On peut ainsi schématiser le circuit emprunté par les données. Pour calculer la valeur chiffrée C, le bloc de bits M est utilisé avec une clef K de longueur définie, par l'algorithme E. Ce circuit peut être plus ou moins complexe en fonction du mode employé. Il en existe quatre pour le chiffrement par bloc, définis par le FIPS 81 [2]. Ces modes ont été définis lors de la mise en place de *Data Encryption Standard*. Il est possible d'en implémenter plusieurs sur un même support mais cela n'est pas une obligation.

1. Mode dictionnaire

Le *Electronic CodeBook mode* est le plus simple des quatre présentés. Il traite chaque bloc indépendement des autres, ce qui permet d'insérer un ordre aléatoire de traitement. En contre partie, il se voit plus sensible aux attaques par regroupement et analyse statistique.

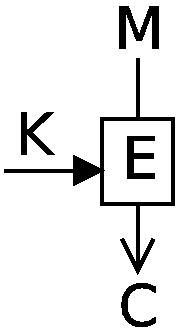

FIG. 3.1 – mode dictionnaire.

2. Mode Chaînage de Blocs Chiffrés

Mode le plus couramment utilisé, le *Cipher Block Chaining mode* permet d'augmenter la sécurité en introduisant une complexité supplémentaire dans le processus de chiffrement en créant une dépendance

entre les blocs successifs.

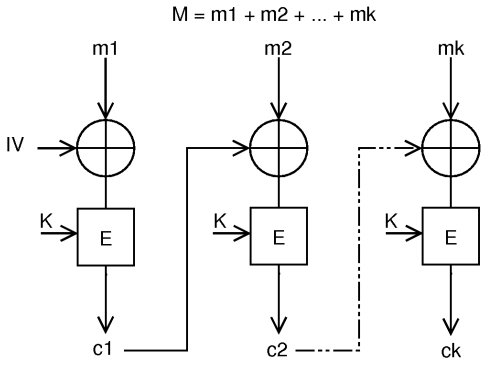

FIG. 3.2 – Cipher Block Chaining mode

Comme le montre la figure 3.2, chaque M_i est transformé avant le chiffrement. On applique $M_i \leftarrow M_i \oplus C_{i-1}$, où $\oplus$ représente le ou exclusif (*XOR*), et C_{i-1} le résultat du bloc précédent venant d'être chiffré.Pour le premier bloc, il est nécessaire d'employer un vecteur d'initialisation noté *IV*, puisqu'il n'y a pas de valeur C_{i-1} existante. Ce vecteur est constitué d'un bloc de bits aléatoires qui sera transmis au destinataire du message, afin de pouvoir effectuer le déchiffrement. Il peut être transmis publiquement sans que cela n'affecte la sécurité.

3. Mode de rebouclage chiffré

Le *Cipher FeedBack mode* se rapproche d'un algorithme par flot. Le message M est divisé en sous blocs M_i de taille allant de 1 à n bits. Ils sont ensuite XORés avec le C_{i-1}. Contrairement au mode précédent, le chiffrement *E* ne s'effectue pas après le *XOR* du bloc en cours et n'influe pas sur le C_i courant, mais sur le suivant.

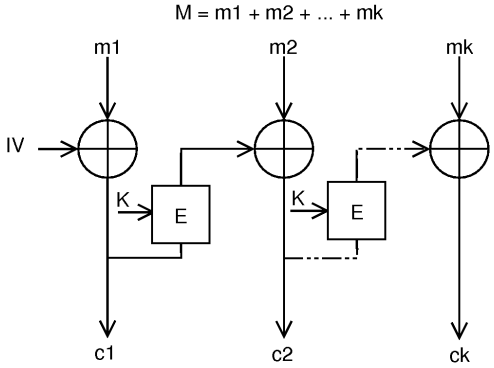

FIG. 3.3 – mode de rebouclage chiffré.

4. Mode rebouclage sur la sortie

Dans ce quatrième et dernier mode, nommé *OutputFeedBack*, l'opération effectuée est indépendante des données, qu'elles soient en clair ou chiffrées.

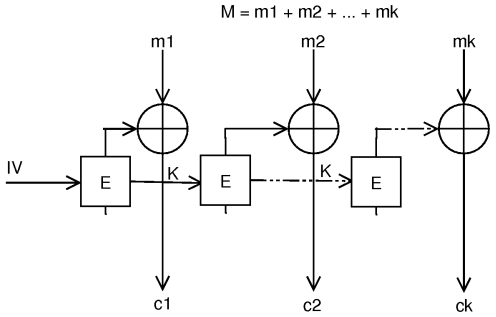

FIG. 3.4 – Mode rebouclage sur la sortie.

II Les S-boxes

Les s-boxes, pour *substitution boxes*, sont des composants essentiels du chiffrement symétrique. Dans les chiffrements par bloc, comme pour AES, ils sont utilisés pour obscurcir la relation entre le texte en clair et le texte chiffré selon propriété de la confusion énoncée par Shannon. Une S-box prend un nombre de bits venant du message m et le substitue par un nombre de bits identique venant du message n. Cette transformation est gérée par la constitution de tableaux statiques ou créés dynamiquement.

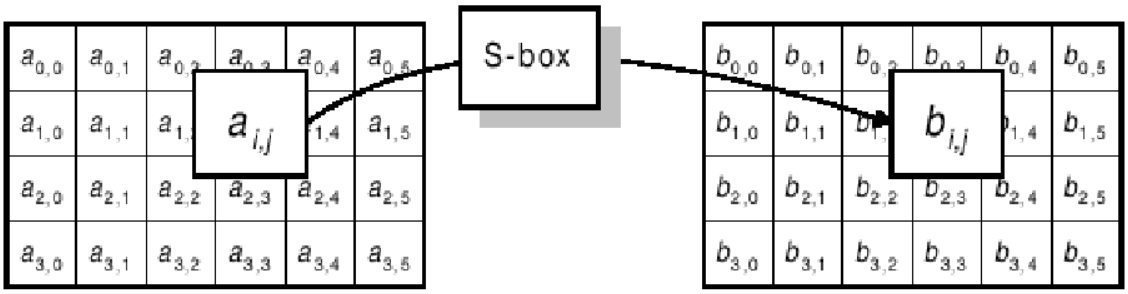

FIG. 3.5 – La matrice a est substituée en matrice b par l'intermédiaire d'une s-box.

III Les réseaux de Feistel

Un réseau de Feistel est une méthode générale de transformation de n'importe quelle fonction en une permutation. Il a été inventé par Horst Feistel pour le design de *Lucifer* et popularisé par le *DES*. On le retrouve dans bon nombre d'algorithmes de chiffrement par bloc dont *CAST-128*, *Blowfish* ou encore *RC5*. Concrètement, dans ce schéma, un bloc de texte en clair est découpé en deux ; la transformation est appliquée lors de ce tour à une des deux moitiés, et le résultat est combiné avec l'autre moitié par un ou exclusif. Les deux moitiés sont alors inversées pour l'application du tour suivant. Deux tours complémentaires forment un *cycle*. Lorsqu'un cycle est complet, chaque bit du bloc de texte à traiter a été modifié une fois.

Chapitre 4

Revue des finalistes

I Mars

1. Présentation de l'algorithme

Mars [3] reçoit en entrée quatre mots de trente-deux bits et restitue quatre mots de trente-deux bits chiffrés. C'est un réseau de Feistel divisé en trois phases :

- un *avant chiffrement* composé de huit tours,
- un *noyau* composé de seize tours,
- un *après chiffrement* composé de huit tours.

L'*avant* et l'*après* chiffrement sont presque l'inverse l'un de l'autre. Pour le premier, on commence par additionner la clef aux données, puis on effectue huit transformations par l'intermédiaire de S-boxes. Ensuite, on opère sur le premier mot qui servira de base pour les trois suivants. On divise ce mot en quatre octets b_0, b_1, b_2, b_3 qui servent de référence aux tableaux de transformation. Les entrées des S-boxés S_0 *et* S_1 sont Xorées ou ajoutées dans les trois autres mots. Les opérations sont les suivantes :

- Xor $S_0[b_0]$ sur le deuxième mot,
- Add $S_1[b_1]$ sur le deuxième mot,
- Add $S_0[b_2]$ sur le troisième mot,
- Xor $S_1[b_3]$ sur le quatrième mot,
- rotation de 24 bits vers la droite.

Pour terminer cette phase de pré-chiffrement, lors du premier et du cinquième tour, on ajoute le quatrième mot au premier. Lors du deuxième et sixième tour, ou ajoute le deuxième mot au premier.

Dans la phase de post chiffrement, on remplace les additions par des soustractions et pour finir, on soustrait le deuxième mot au premier aux tours quatre et huit, ainsi que le quatrième mot au premier lors des tours trois et sept, ceci afin d'éliminer les attaques différentielles.

Le noyau de l'algorithme se décompose également en deux parties de huit tours, comme indiqué par la figure 4.1

Le noyau se caractérise par quatre opérations majeures, symbolisée par la Efunction, représentée par la figure 4.2, une rotation de treize bits, des additions et un Xor.

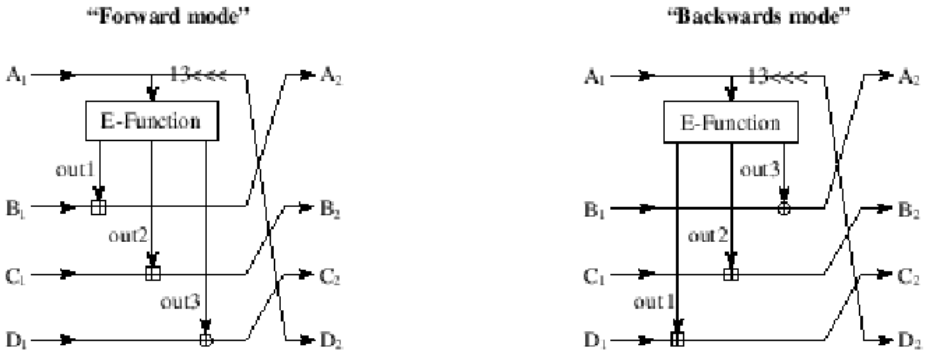

FIG. 4.1 – Schéma opératoire du noyau de MARS décomposé en deux parties.

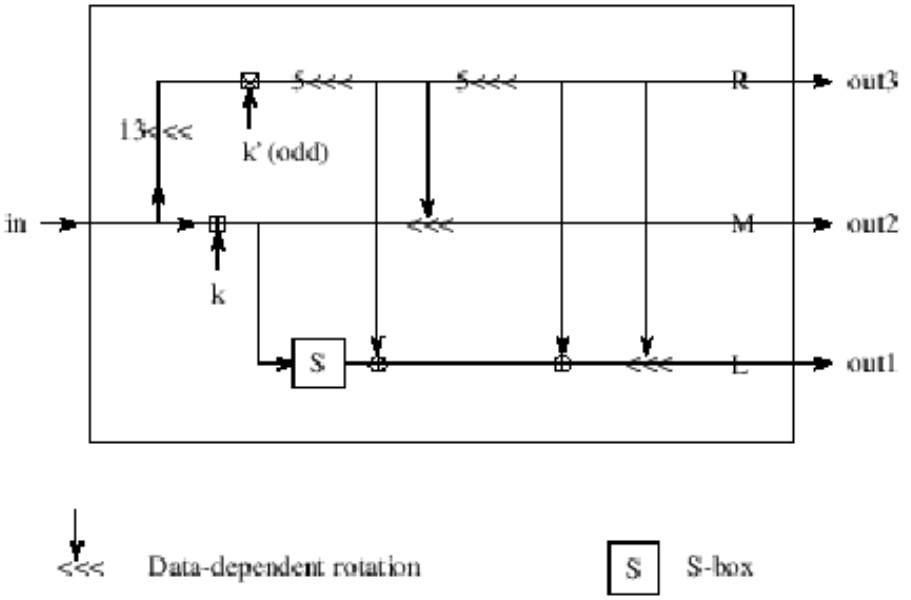

FIG. 4.2 – Schéma descriptif de la transformation interne du noyau de MARS par la Efunction.

2. Sécurité générale

Aucune attaque n'est connue pour MARS. Contrairement aux autres finalistes, MARS utilise à la fois des rotations de données et les S-Boxes comme composants non linéaires. La structure hétérogène des tours (16 mixages et 16 tours du noyau) de MARS le rend difficile à appréhender en matière de sécurité. MARS a reçu de nombreuses critiques sur sa complexité, qui a entravé l'analyse de sa sécurité pendant le temps offert durant le processus de développement d'AES.

3. Implémentations logicielle

L'efficacité logicielle de MARS dépend de la faculté des processeurs et langages à manipuler les multiplications de 32 bits et les opérations de rotation de variables. Ceci induit des variations de résultats, même entre processeurs de famille identique, et par conséquent cause des changements pour un compilateur sur un processeur donné. MARS se classe donc au milieu des finalistes en terme de performance pour le chiffrement, le déchiffrement et la mise en place de la clef.

4. Systèmes embarqués

MARS n'est pas bien adapté à un environnement dont la place est restreinte à cause de ses besoins en espace ROM.

5. Implémentations matérielle

MARS demande plus de ressources que la moyenne des autres algorithmes. Sa sortie est généralement en dessous de la moyenne et son efficacité n'en est pas meilleure. La rapidité de traitement de MARS est indépendante de la taille de la clef utilisée.

6. Attaques sur les implémentations

Lorsqu'il est implémenté sur du matériel vulnérable à des attaques par mesure du temps ou de la puissance, Mars semble difficile à protéger, cela à cause de son emploi des multiplications, rotations de variables et additions. L'emploi de méthodes de masquarade dégrade de manière sévère ses performances. La programmation de la clef est légèrement vulnérable à une attaque par analyse de puissance.

7. Chiffrement et déchiffrement

Le chiffrement et le déchiffrement appartiennent à la même fonction. La vitesse ne change donc pas entre l'un et l'autre des processus. L'espace nécessaire pour implémenter les deux fonctions est seulement supérieur de dix pourcents à la mise en place du chiffrement seul.

8. Manipulation de clef

MARS nécessite le calcul simultané de 10 sous-clefs parmi les 40, demandant des ressources additionnelles pour stocker ces dix résultats. Ceci joue en défaveur des espaces restreints en mémoire. MARS requiert également une exécution de la clef programmée pour générer toutes les sous-clefs avant le premier déchiffrement avec une clef spécifique. Le calcul de multiples sous-clefs en une seule fois utilise davantage de ressources que les autres algorithmes qui effectuent le calcul à la volée d'une seule et unique sous-clef.

9. Souplesse et flexibilité

MARS supporte des tailles de clef de 128 à 448 bits.

10. Potentiel de parallélisation

MARS possède un potentiel limité pour le parallélisme afin de chiffrer un seul bloc.

II RC6

1. Présentation de l'algorithme

RC6 [4] diffère un peu de ses concurrents, puisqu'il a la particularité d'hériter d'un autre algorithme, son prédécesseur, nommé RC5. Il profite également d'un grand nom, puisqu'un de ses concepteurs est le célèbre Rivest, le R de *RSA*.

L'algorithme se fonde sur un tableau de clefs précalculées. La figure 4.3 décrit le procédé de chiffrement employé.

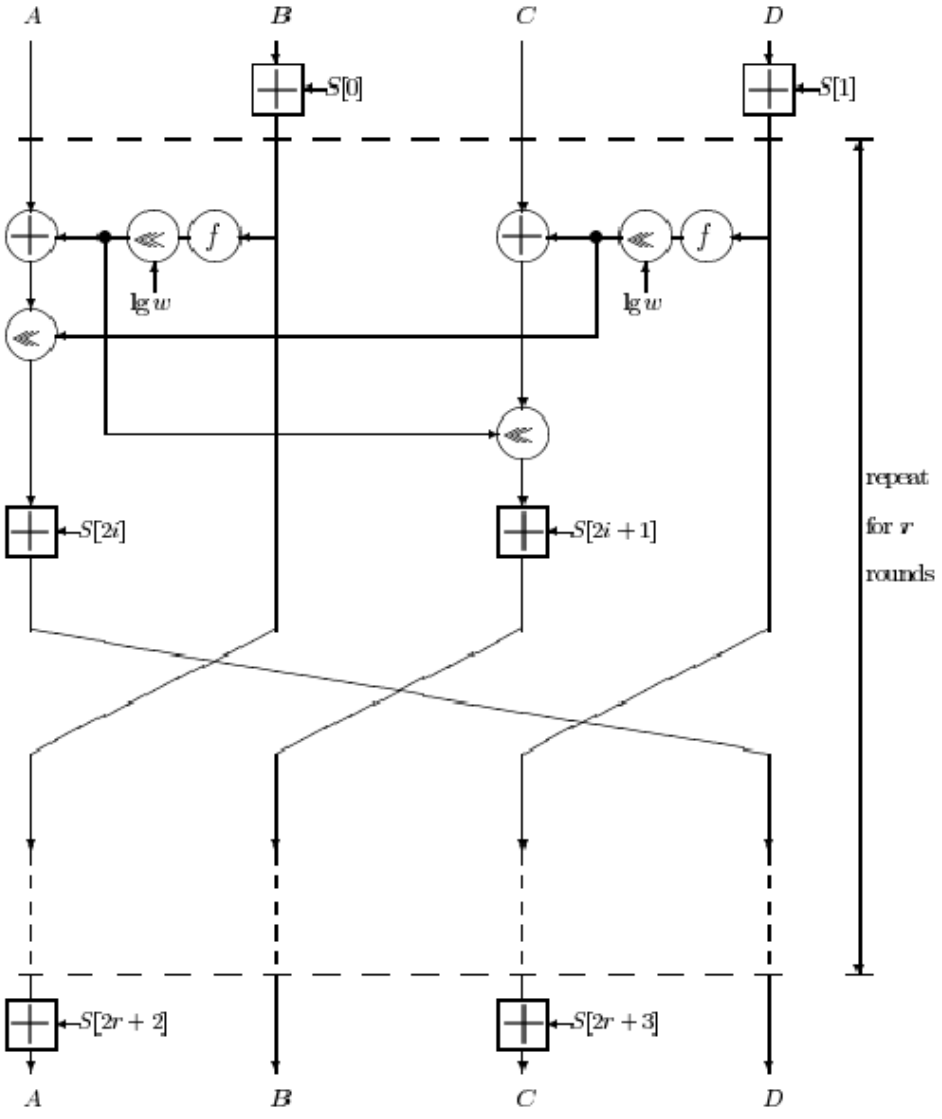

FIG. 4.3 – Schéma descriptif de l'algorithme de chiffrement *RC6*.

Il prend en entrée quatre mots de trente-deux bits, ici notés A, B, C et D, pour ressortir quatre mots chiffrés de longueur identique. Le tableau S[i] représente l'ensemble des mots de trente-deux bits dérivés des clefs précalculées, utilisant une clef initiale fournie par l'utilisateur, pouvant aller de taille de 0 à 255 octets. La fonction f est décrite par $f : x \rightarrow x * (2x + 1)$. Le nombre de tours proposé dans

cette version est de 20, avec des tailles de clefs fixes pouvant se constituer de 16, 24 ou 32 octets.

2. Sécurité générale

RC6 n'est faillible a aucune attaque de sécurité connue. RC6 utilise des rotations de données dépendentes comme composants non linéaires. Ces marges de sécurité paraissent adéquates. RC6 a été loué pour sa simplicité, ce qui a facilité l'analyse de sa sécurité durant le temps imparti du processus de développement d'AES. La lignée de RC6 est un plus : son prédécesseur, RC5, a été soumis à de nombreuses analyses de par le passé.

3. Implémentations logicielle

Les opérations prédominantes dans cet algorithme sont la multiplication et les rotations de variable. Les performances logicielles dépendent de l'appréhension du couple processeur/langage vis à vis de ces manipulations. Le comportement de RC6 est identique pour le chiffrement et le déchiffrement. De manière générale, RC6 est le finaliste le plus rapide sur les plateformes 32-bits. Cependant, sur des processeurs 64-bits, les résultats tombent de manière substancielle. Le temps de mise en place de la clef est moyen.

4. Systèmes embarqués

RC6 ne nécessite que très peu d'espace ROM, ce qui représente un avantage certain dans un environnement d'espace restreint. Malheureusement pour lui, il n'y a pas la possibilité de calculer une sous-clef à la volée pour le déchiffrement, ce qui nécessite une capacité importante de RAM.

5. Implémentations matérielle

RC6 peut être implémenté de manière compacte. Son débit est moyen, mais il reste très rapide pour les modes sans réinjection. Le dit débit est indépendant de la taille de la clef employée.

6. Attaques sur les implémentations

Lorsqu'on l'implémente sur des périphériques vulnérables aux attaques d'analyse de puissance ou de temps, RC6 s'avère difficile à défendre à cause de l'emploi des multiplications, des rotations de variables et des additions. L'emploi de méthodes de masquarade dégrade ses performances. Cependant, dans ces conditions, l'occupation de la RAM et de la ROM reste très raisonnable comparée aux autres finalistes. RC6 est légèrement vulnérable aux attaques par analyse de puissance.

7. Chiffrement et déchiffrement

Le chiffrement et le déchiffrement sont des fonctions similaires. C'est pour cela que l'efficacité de RC6 ne varie pas entre les deux méthodes. Un rapport d'étude sur FPGA annonce que les deux actions ne prennent pas plus de 10% supplémentaires de ressources que le chiffrement seul.

8. Manipulation de clef

RC6 supporte le calcul de la sous-clef à la volée seulement pour le chiffrement, donnant quelques cent octets de valeurs intermédiaires. La sous-clef de déchiffrement doit être pré-calculée. Ce comportement réduit le niveau de manipulation de clef apporté par RC6.

9. Souplesse et flexibilité

Les blocs, la clef et le nombre de tours sont paramètrables. L'algorithme supporte des tailles de clefs supérieures à 256 bits.

10. Potentiel de parallélisation

Le potentiel pour le chiffrement parallèle d'un bloc seul est limité.

III Rijndael

1. Présentation de l'algorithme

L'algorithme se présente en deux temps, tout d'abord une procédure d'expansion de la clef, puis la fonction principale de chiffrement [5].

La fonction de chiffrement se divise en trois : une transformation initiale avec la clé, une série de tours puis une transformation finale.

Le nombre de tours s'établit en fonction de la taille des blocs et de la clé :
- 9 tours si la taille des blocs et de la clé sont de 128 bits,
- 11 tours si la taille des blocs ou de la clé est de 192 bits,
- 13 tours si la taille des blocs ou de la clé est de 256 bits.

a. Description de la fonction d'expansion de la clef

La clef étendue est un tableau linéaire de 4 mots et se note $W[Nb * (N_r + 1)]$. Les premiers N_k mots contiennent la clef de chiffrement. Tous les autres mots sont définis de manière récursive avec des indices plus petits. La fonction d'expansion dépend de la valeur de N_k. Les mots suivants sont calculés en effectuant $W[i] \leftarrow W[i-1]\ XOR\ W[i-nk]$.

Pour les mots situés sur une position qui est un multiple de N_k , une transformation est appliquée à $W[i-1]$ avant le *Xor*. Cette transformation consiste en une permutation cyclique nommée *RotWord()* d'un cran vers la gauche suivie d'une application de la emphS-box, *SubWord()*, séparément sur chaque octet du mot puis d'un *Xor* avec un vecteur dépendant du tour, vecteur noté $Rcon[i/N_k]$.

b. Description d'un tour

Dans un tour, quatre transformations sont appliquées au bloc à chiffrer, elles sont représentées dans le schéma simplifié 4.4.

Dans la première étape nommée *ByteSub*, on substitue chaque octet des sous-blocs selon une s-box. Cette opération augmente la non-linéarité des données.

Lors du *shift row*, on décale les bits des sous-blocs. Pour une clef de 128 bits, la transformation donnerait :

$$\begin{pmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 5 & 9 & 13 \\ 14 & 2 & 6 & 10 \\ 11 & 15 & 3 & 7 \\ 8 & 12 & 16 & 4 \end{pmatrix}$$

Le *Mix Column* est une multiplication de la matrice obtenue par la suivante :

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Cette étape augmente la diffusion des données entre les tours. Les octets utilisés dans la multiplication sont traités comme des polynômes et non comme des nombres. Si l'un des produits est supérieur à une taille de huit bits, on applique un *Xor* plutôt qu'une troncature. Le masque employé est alors 100011011.

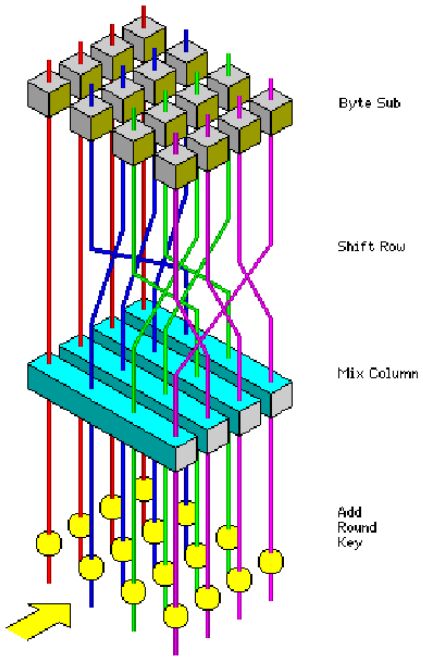

FIG. 4.4 – Schéma simplifié du mode de fonctionnement de l'algorithme.

Le *AddRoundKey* termine le tour en additionnant les sous-clefs aux sous-blocs.

Le dernier tour diffère légèrement, puisque le *Mix Column* n'est pas appliqué.

2. Sécurité générale

Rijndael n'est sensible à aucune attaque de sécurité connue. Il utilise des S-boxes comme composants non linéaires. L'algorithme semble garder une marge de sécurité adéquate, bien qu'il ait reçu des critiques sur sa structure mathématique qui pourrait mener à des attaques. D'un autre côté, cette simplicité a très largement facilité son analyse.

3. Implémentations logicielle

Rijndael fonctionne de manière optimale sur les plateformes 8 bits, 64 bits et DSP. Cependant, la baisse de performances est nette pour de grandes tailles de la clef, puisque cela augmente considérablement le nombre de tours à effectuer. La possibilité de traiter les tâches en parallèle facilite l'usage des ressources du processeur, ce qui permet de très bonnes performances au niveau logiciel, même lorsqu'il est implémenté dans un mode incapable d'entrelacement.

4. Systèmes embarqués

De manière générale, il s'insère très bien dans les systèmes embarqués pour le chiffrement ou le déchiffrement. Cependant, la mise en place simultanée des deux modes pose problème. Il ne nécessite que peu de RAM et de ROM. Son besoin en ROM augmente lors de la mise en place des deux modes de manière concurrente. C'est un désavantage, quoiqu'il semble rester approprié pour cet environnement. Le programme principal de déchiffrement est séparé de celui du chiffrement.

5. Implémentations matérielle

Rijndael est le finaliste possédant le plus grand rendement dans les modes avec *feedback* et se place second dans les modes sans *feedback*. Pour les clefs de taille 192 et 256 bits, le rendement tombe et se révèle plus lourd dans son implémentation à cause de l'augmentation du nombre de tours. Pour les implémentations entièrement chaînées, l'espace requis augmente bien que le débit reste inchangé.

6. Attaques sur les implémentations

Les opérations utilisées par Rijndael sont parmis les plus faciles à défendre contre des attaques par analyse de puissance ou de temps. L'utilisation de techniques de masquarade pour fournir quelques défenses de plus à l'algorithme n'influent pas sur ses performances, contrairement aux autres finalistes. Sa demande en RAM reste raisonnable. Il apparaît donc comme ayant un avantage vis-à-vis de ses compétiteurs lorsque l'on considère la mise en place de ces protections. Cependant, ces implémentations restent faillible à des attaques par analyse de puissance.

7. Chiffrement et déchiffrement

Le chiffrement et le déchiffrement diffèrent. L'étude sur FPGA démontre que l'implémentation des deux transformations prend 60% d'espace supplémentaire que le chiffrement seul. La vitesse ne varie pas de façon significative entre les deux, bien que la mise en place de la clef soit plus longue pour le déchiffrement.

8. Manipulation de clef

Rijndael supporte le calcul de sous-clefs à la volée pour le chiffrement. Il nécessite une exécution préalable pour la génération des sous-clefs de déchiffrement pour une clef spécifique. Cela alourdit légèrement la manipulation de ses clefs.

9. Souplesse et flexibilité

L'algorithme supporte pleinement les blocs et les clefs de taille 128 bits, 192 bits et 256 bits, selon n'importe quelle combinaison. En principe, sa structure peut s'accomoder à tout bloc et toute clef qui serait multiple de 32, du moment que le nombre de tours adéquat est spécifié.

10. Potentiel de parallélisation

Le support du parallélisme pour le chiffrement d'un bloc simple est excellent.

IV Serpent

1. Présentation de l'algorithme

Serpent [6] est un algorithme par bloc qui accepte en entrée quatre mots de trente-deux bits. Son système de chiffrement principal se compose de trente-deux tours. Il utilise 33 sous-clefs précalculées. La longueur de la clef donnée par l'utilisateur peut varier selon les tailles 128 bits, 196 bits ou 256 bits. Dans le cas où cette longueur est inférieure à 256 bits, on remplit l'espace avec un "1" suivit de "0".

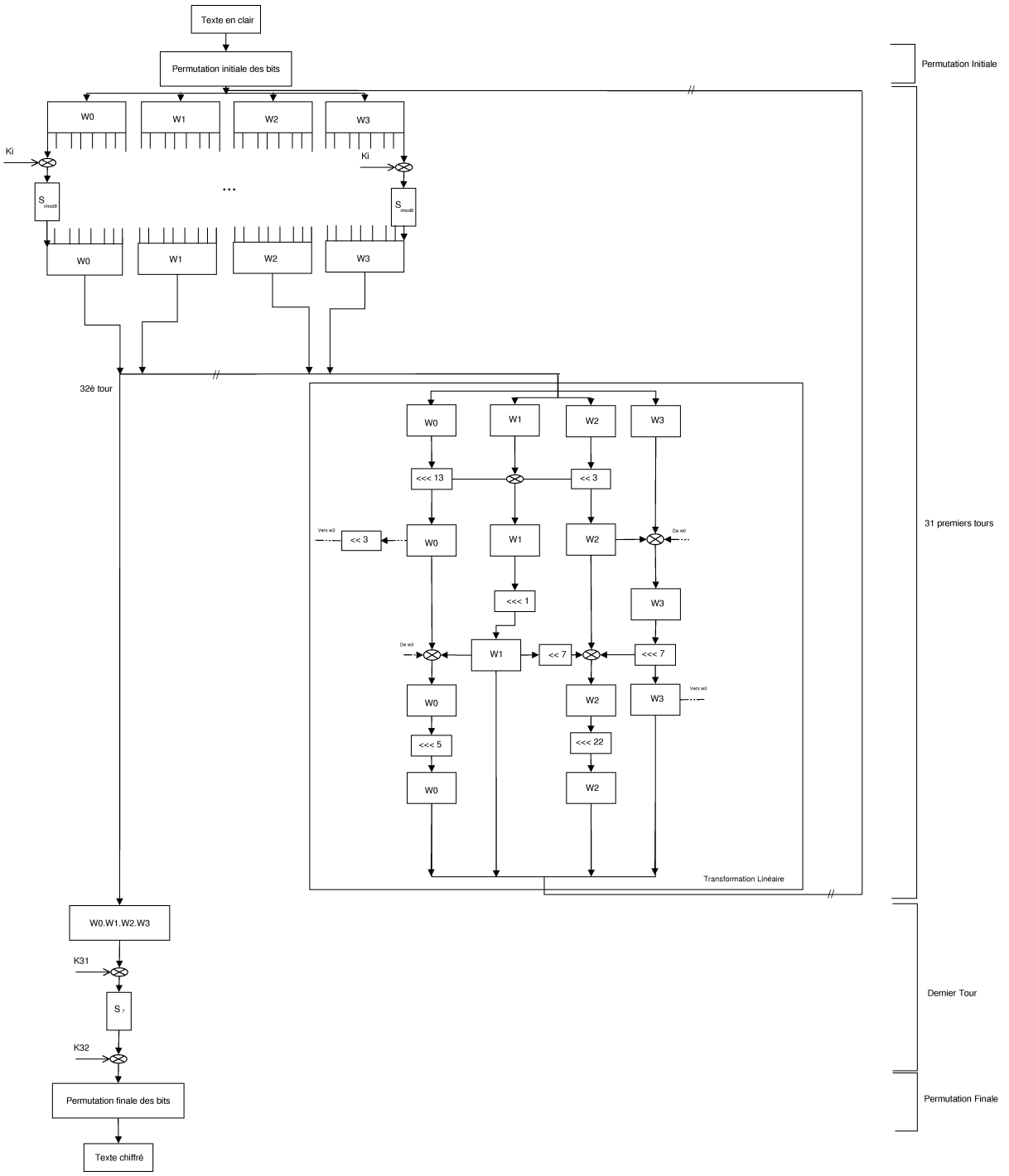

FIG. 4.5 – mode de fonctionnement de l'algorithme Serpent.

La figure 4.5 montre le schéma de chiffrement :
- une permutation initiale qui change l'ordre des bits dans le bloc,
- trente-deux tours, chacun consistant en un *Xor* d'un sous bloc de 4 bits avec la clef du tour, un passage à travers une S-Box, terminé par une transformation linéaire,
- une permutation finale qui est l'inverse de la permutation initiale.

à chaque tour, une S-box est répliquée trente-deux fois pour être utilisée avec les trente-deux sous-ensembles de bloc. Il y a en tout huit S-boxes différentes. Cela signifie qu'au neuvième tour, on reprendra la S-box numérotée 0, puis au dixième la S_1, etc ... Lors du dernier tour, la transformation linéaire laisse la place à deux *Xor* entrecoupés par une opération de S-Box.

La transformation linéaire emploie des *Xor*, ainsi que des rotations de bits ($\lll$) et des décalages ($\ll$). Le résultat de cette transformation termine un tour et sert de base de bloc pour le tour suivant.

Le déchiffrement diffère du chiffrement dans le fait qu'il inverse les S-boxes et que celles-ci doivent être employées dans l'ordre inverse, tout comme la transformation linéaire et l'ordre des sous-clefs qui sont également inversés.

2. Sécurité générale

Serpent n'est faillible à aucune attaque connue. Il utilise des S-boxes comme composants non-linéaires. Serpent possède une grande marge de sécurité ainsi qu'une structure simple, ce qui a facilité les analyses de sécurité dans la période dédiée à cet usage lors du processus de sélection d'AES.

3. Implémentations logicielle

Serpent est de manière générale l'algorithme le plus lent pour le chiffrement et le déchiffrement lorsqu'il est implémenté logiciellement. Le temps de la mise en place de la clef est moyen.

4. Systèmes embarqués

Serpent ne nécessite que peu de RAM et de ROM. Un retour d'expérience permet cependant de voir que l'usage de la ROM augmente si le chiffrement et le déchiffrement sont implémentés simultanément. Malgré cette constatation, il reste utilisable dans un système embarqué.

5. Implémentations matérielle

Ses implémentations liées font de lui le finaliste ayant le plus fort débit en mode sans retour. Lorsqu'on l'intègre dans un mode avec retour, il se positionne alors en deuxième place. Son efficacité est généralement bonne et sa vitesse reste indépendante de la taille de sa clef.

6. Attaques sur les implémentations

Les opérations utilisées par l'algorithme sont très faciles à défendre contre des attaques de temps ou de puissance. Serpent se défend le mieux lorsqu'il utilise le mode de découpage de bits. Il est très lent et nécessite beaucoup de ROM. Ses implémentations sont vulnérables à des attaques par analyse de puissance.

7. Chiffrement et déchiffrement

Le chiffrement et le déchiffrement sont deux fonctions différentes qui ne partagent que très peu de ressources physiques. Une étude sur FPGA montre que la mise en place simultanée des deux modes prend deux fois plus d'espace que le chiffrement seul. C'est un gros désavantage matériel lorsque les deux fonctions doivent être présentes, puisque la vitesse ne varie que très peu selon le mode employé.

8. Manipulation de clef

Serpent supporte le calcul à la volée des clefs de chiffrement et de déchiffrement. Dans le deuxième cas, un seul calcul est nécessaire pour obtenir une sous-clef à partir de la clef originale. Ce calcul est différent de la transformation utilisée pour chaque sous-clef. Ce dernier point alourdi légèrement la manipulation des clefs.

9. Souplesse et flexibilité

L'algorithme est en mesure de manipuler des clefs de taille allant jusqu'à 256 bits. De plus, une technique de découpage de bits sur les processeurs 32-bit peut être employée pour améliorer ses performances.

10. Potentiel de parallélisation

Le potentiel de parallélisation du chiffrement d'un bloc simple est limité.

V Twofish

1. Présentation de l'algorithme

a. Algorithme général

Twofish [7] utilise une structure approchante de *Feistel*, avec en sus un *whitening*, un blanchiment, de l'entrée et de la sortie. La rotation d'un bit rend ce schéma différent d'un réseau de *Feistel*. Les données en clair sont découpées en quatre mots de trente-deux bits. Le blanchiment s'effectue sur chaque mot par un *Xor* avec des clefs précalculées. Cette étape, ainsi que le blanchiment de la sortie, permettent d'accroître la difficulté d'attaque, en masquant les entrées spécifiques du premier et dernier tour. Lors des seize tours suivants, les deux mots de gauche sont utilisés comme entrées de la fonction g. Le résultat obtenu sur ces deux mots est combiné à l'utilisation d'une *pseudo transformation de Hamard* et ajouté à deux mots. Ces deux résultats sont *Xorés* avec les deux mots de droite. Ceux-ci subiront un décalage. Pour l'un, un décalage d'un bit vers la gauche avant le *Xor*, pour l'autre, un décalage d'un bit vers la droite après le *Xor*. Lors du tour suivant, les mots sont inversés afin de pouvoir effectuer un cycle. Lorsque le nombre des seize tours est atteint, on défait la dernière inversion effectuée puis on effectue le blanchiment sur la sortie, à savoir un *Xor* sur chaque mot avec des clefs précalculées.

Dans la figure 4.6, on peut noter la présence des fonctions f et g.

b. La fonction *f*

La première prend trois arguments, deux mots R_0 et R_1, ainsi que le nombre de tour r, afin de sélectionner les clefs correspondantes. R_0 est passé à la fonction g. R_1 est quant à lui décalé de huit bits vers la gauche avant d'être envoyé à g. En sortie de g, les deux blocs sont combinés par l'intermédiaire d'une *pseudo transformation de Hamard*. Pour finir, deux clefs expansées sont ajoutées aux résultats.

c. La fonction *g*

Cette fonction forme le cœur de *Twofish*. Chaque mot positionné en entrée est divisé en quatre octets, puis transformés par des *S-boxes*. Les quatre résultats sont interprétés comme une matrice et multipliés par une matrice $4 * 4$. Le résultat est un vecteur interprété comme un mot.

2. Sécurité générale

Twofish n'est faillible à aucune attaque de sécurité connue. Il emploie des S-boxes comme composant non-linéaire. Il semble avoir une marge de sécurité suffisante, bien qu'il fut critiqué sur sa séparation d'attribut et pour sa complexité, ce qui a entravé son analyse de sécurité pendant le temps imparti pour AES.

3. Implémentations logicielle

Les résultats sont mixtes. La mise en place de la clef est lente, son temps décroît en fonction de sa largeur et de l'option utilisée.

4. Systèmes embarqués

La RAM et la ROM requises par l'algorithme semblent correctes pour un système embarqué.

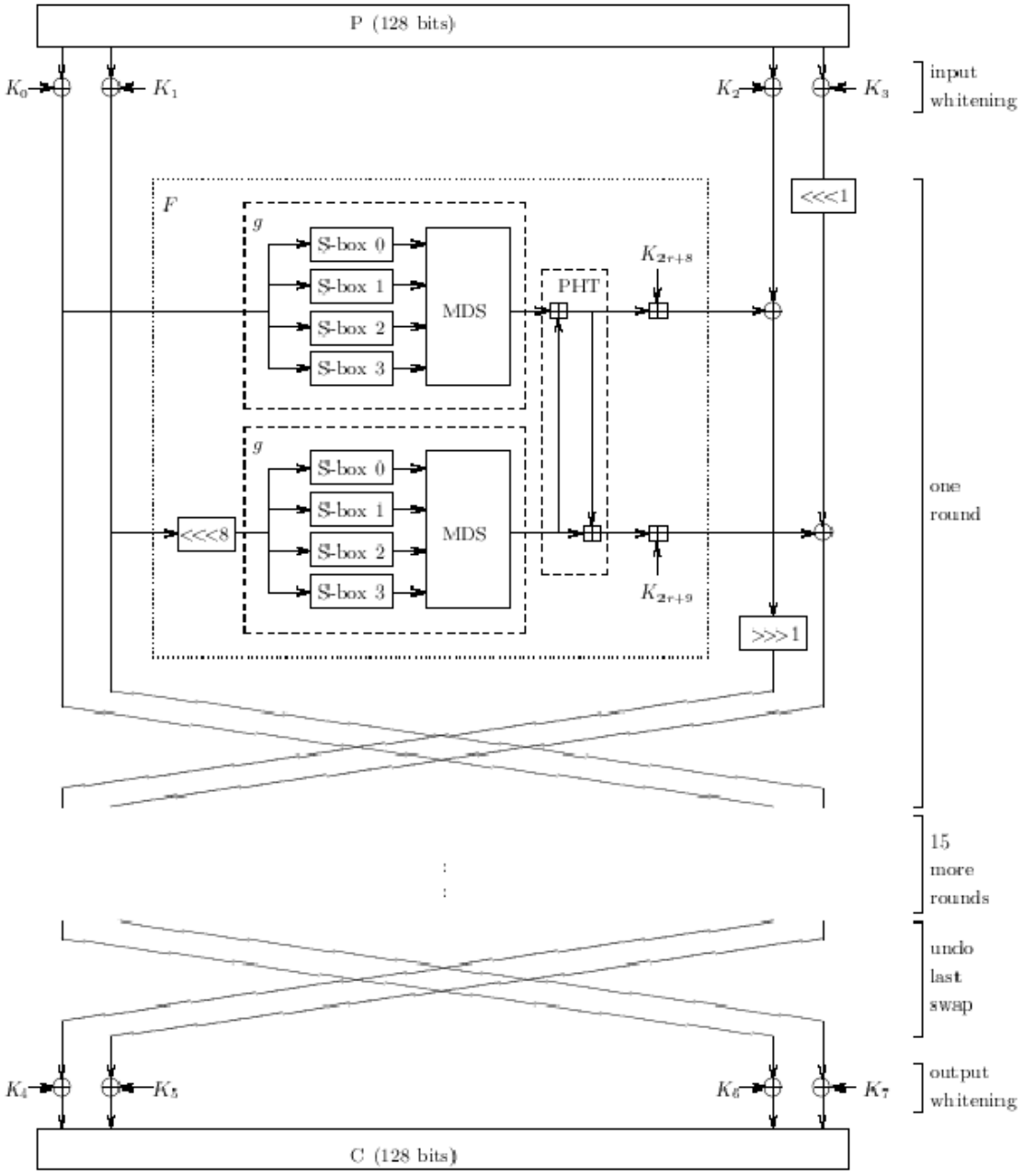

FIG. 4.6 – Schéma simplifié du mode de fonctionnement de l'algorithme Twofish.

5. Implémentations matérielle

Le débit et l'efficacité lors d'une implémentation sur une architecture basique ou liée sont généralement moyens. Le débit est quelque peu réduit pour des clefs de grande taille, que ce soit dans l'architecture standard ou liée. Les implémentations compactes sont possibles.

6. Attaques sur les implémentations

Twofish emploie la méthode d'addition qui est très difficile à défendre contre les attaques par analyse de puissance et de temps. L'utilisation de technique de masquarade ne dégrade pas trop ses perfor-

mances mais augmente de manière considérable la RAM employée. Il reste vulnérable aux attaques par analyse de puissance.

7. Chiffrement et déchiffrement

Le chiffrement et le déchiffrement sont presque des fonctions identiques. De ce fait, et d'après l'étude sur FPGA, l'implémentation des deux ne prend pas plus de dix pourcents supplémentaire que dans le cas du chiffrement seul.

8. Manipulation de clef

Twofish supporte le calcul à la volée des sous-clefs de chiffrement et de déchiffrement, ce qui rend la manipulation de sa clef très facile.

9. Souplesse et flexibilité

La spécification de Twofish décrit quatre options pour implémenter la clef dépendante des S-boxes, permettant des transactions de performances variées. La clef peut avoir une taille arbitraire allant jusqu'à 256 bits.

10. Potentiel de parallélisation

La parallélisation du chiffrement d'un bloc simple est limité.

VI Quel choix pour AES ?

Chacun des finalistes offre une sécurité correcte et un nombre d'avantages non négligeable. Chacun de ces finalistes pourrait servir de manière admirable comme proposition pour l'AES. Cependant, chaque algorithme possède un ou plusieurs domaines où il pêche vis à vis de ses pairs ; c'est ainsi que l'on peut dire qu'aucun des présents n'écrase les autres par sa supériorité. Le NIST a choisi Rijndael comme proposition à AES après un très long cycle de processus d'évaluation complexe. Durant cette pèriode, le NIST a analysé tous les commentaires publics, les papiers, les débats lors des conférences, ainsi que ses propres études et rapports. C'est ainsi que Rijndael s'est présenté comme la meilleure solution pour AES. Il apparait comme ayant de très bonnes performances tant sur les traitements matériels que logiciels, et cela en dépit du mode utilisé, c'est à dire avec ou sans réinjection. Son temps de mise en place de la clef est excellent et son traitement est correct. Le peu de ressources mémoire requises par Rijndael le rendent idéal pour les environnements embarqués, dans lesquels il montre encore une fois d'excellentes performances. En matière de sécurité, les opérations employées sont les plus faciles à défendre contre les attaques par analyses de temps ou de puissance, tout en minimisant l'impact de telles restrictions sur les performances du système. Le design de l'algorithme est flexible tant en terme de bloc que de tailles de clef et il peut s'accomoder d'une modification du nombre de tour, bien que cette option requièrt d'avantage d'études qui n'ont pas été considérées à cette étape de la sélection. Pour finir, la structure interne des tours semblent adéquate pour bénéficier d'instruction de parallélisme. Il y a beaucoup d'inconnues concernant les futures plateformes informatiques et l'environnement lié dans lesquels devra s'intégrer AES. Cependant, lorsque l'on considère simultanément la sécurité, la performance, l'efficacité, l'implémentabilité et la flexibilité, tous ces éléments font de Rijndael la sélection appropriée pour AES, pour un usage d'aujourd'hui et de demain. C'est tout du moins les qualités qu'a retenu le NIST dans ce concours.

VII La sécurité d'AES en question

Bruce Schneier disait lors de la mise en place de la sélection pour ce nouveau standard que l'algorithme devrait faire face à quelques soixante années d'épreuves. Qu'en est-il seulement quatre ans après la sélection de *Rijndael* ? Il faut tout d'abord se pencher du côté du grand ennemi, la *National Security Agency*. Schneier dans un article sur la NSA et AES [9] écrit ceci : « La NSA n'a pas statué sur le fait d'employer AES pour les informations classées protégées, ni sur le fait qu'elle l'utiliserait largement. Il a seulement été dit que, lorsque cela semblait possible, AES pourrait être employé. » Le F.I.P.S. 197 [10] déclare dès la première page que les organismes nationaux peuvent utiliser cet algorithme à condition que les données ne soient pas sensibles. En juin 2003, le *CNSS*, et donc la NSA, approuve l'AES avec une clef de 128 bits pour les documents *SECRET* et requiert une clef minimum de 196 ou 256 bits pour les documents *TOP SECRET* [8]. Cependant, toute implémentation utilisée pour chiffrer ses documents devra au préalable recevoir l'aval et la certification de la NSA.

éric Filiol, Chef du Laboratoire de virologie et de cryptologie de l'ESAT, est actuellement en contact avec Vincent Rijmen, un des deux créateurs de *Rijndael*. Il a un peu plus de 13000 cryptanalyses calculées tentant à prouver des faiblesses de l'algorithme implémenté en mode *Cipher Block Chaining*. Ces calculs sont présentés dans un papier non publié [11] et contesté par Nicolas Courtois [12]. Présent lors de la dernière conférence qui se tenait à New York, il fait part du lobbying ouvert de la part des états-Unis et parle de leur préférence en faveur de *MARS*. Il reste étonné que malgré cette poussée américaine, l'armée US préfère employer des chiffrements par flot, méthode également utilisée par l'armée Française avec du flot synchronisé. Il note toutefois que l'OTAN vient d'accepter *AES* comme procédé de chiffrement et reste étonné de ce choix. Pour finir, suite à ses recherches encore confidentielles, il conseille fortement l'abandon de tout système de chiffrement par bloc, en faveur d'un chiffrement par flot.

Que faire dans ce cas ? Le processus de sélection a pris plus de deux ans et remplace petit à petit le DES. C'est un cheminement long et coûteux. Alors que les machines deviennent plus puissantes chaque jour, est-il judicieux de placer sa confiance dans une base de calcul définie ?

Conclusion

Au travers de ce mémoire, nous avons pu parcourir la cryptographie de sa naissance aux temps modernes. Tout au long du processus de sélection de l'*Advanced Encryption Standard*, nous avons pu prendre conscience de l'implication de la communauté cryptographique dans ce procédé. L'étude des cinq finalistes a permis de se plonger de manière concrête dans le mode de fonctionnement d'un algorithme de chiffrement symétrique et d'en comprendre les rouages élémentaires.

Bien que succincte, l'analyse de chacun des finalistes sur différents points de comparaison met en évidence la difficulté à choisir distinctement un algorithme parmi les autres. La *shortlist* s'est avérée très difficile à départager, malgré de nombreux tests rigoureux et variés. Comment alors être sûr que le choix fait par le NIST est le bon ? Ils pensent, ils souhaitent, mais en aucun cas ne peuvent affirmer. Et qu'en est il lorsque l'on se penche sur les travaux de M. Filiol ? Ceux-là même qui tendent à mettre en évidence les faiblesses du chiffrement de Rijndael. La seule chose dont on peut s'assurer, c'est que le choix du NIST est ferme et définitif et que de nombreux organismes nationaux ou internationaux, comme l'OTAN, se calquent sur cette décision pour valider l'algorithme comme choix de chiffrement par défaut. Peut-être la solution à ces problèmes se trouvera t'elle dans le chiffrement quantique, mais pour le moment nous devrons nous contenter d'un flot d'algorithmes par bloc.

Pour ce qui concerne mon travail, cette recherche était en tout point intéressante et enrichissante. Alors que mes différentes expériences m'amènent de manière plus générale sur la conception interne des systèmes et leur mode de fonctionnement, j'ai pu prendre un peu plus de hauteur et appréhender le fonctionnement de la sécurité sur laquelle se fonde une grande partie des systèmes actuels.

Les enseignements du CNAM m'auront aidé dans l'utilisation des diverses ressources qui étaient à ma disposition, tant en termes de gestion du temps que du système d'information.

Glossaire

Acrostiche : Texte ou poème composé de telle sorte qu'en lisant verticalement la première lettre de chaque ligne, on trouve un mot ou le nom d'une personne..

Algorithme : Suite d'opérations élémentaires à appliquer à des données pour aboutir à un résultat désiré. Par exemple, une recette de cuisine est un algorithme..

Antigramme : Texte déjà chiffré qui va être surchiffré..

ASCII : (American Standard Code for Information Interchange) Code standard américain pour l'échange d'information qui traduit en nombres les caractères de l'alphabet et autres. Exemple : A est traduit 65..

Asymétrique : Se dit d'un algorithme de cryptage utilisant une clef publique pour chiffrer et une clef privée (différente) pour déchiffrer les messages..

Attaque : Tentative de cryptanalyse..

Authentifier : S'assurer de l'identité de l'émetteur d'un message, et de l'intégrité du message reçu..

Autoclave : Chiffre qui utilise le message clair lui-même comme clef..

Bigramme : Séquence de deux lettres consécutives. Exemples : ee, th, ng, ...

Casser : Dans l'expression casser un code, trouver la clef du code ou le moyen d'accéder à ce qu'il protégeait..

Chiffre : Ensemble de procédés et ensemble de symboles (lettres, nombres, signes, etc.) employés pour remplacer les lettres du message à chiffrer. On distingue généralement les chiffres à transposition et ceux à substitution..

Chiffrement : Procédé grâce auquel on peut rendre la compréhension d'un document impossible à toute personne qui n'a pas la clef d'encodage..

Chiffrer : Transformer un message afin qu'il ne soit lisible qu'à l'aide d'une clef..

Clef : Dans un système de chiffrement, elle correspond à un nombre, un mot, une phrase, etc. qui permet, grâce à l'algorithme de chiffrement, de chiffrer ou de déchiffrer un message..

Clef faible : Clef qui, pour une raison quelconque (à cause de sa longueur, d'une propriété mathématique. etc.), permet de casser rapidement le code..

Clef privée : Clef permettant le déchiffrement d'un message et donc restant secrète. Dans le cas d'un système symétrique, elle sert aussi au chiffrement et est donc connue de l'émetteur comme du récepteur..

Clef publique : Clef servant au chiffrement d'un message dans un système asymétrique, et donc librement diffusée..

Code : Système de symboles (mots, nombres, signes, etc.) remplaçant des mots entiers. Exemples : 007 à la place de James Bond, wesax à la place de retraite immédiate, ...

Confidentialité : Assurer la confidentialité de données, c'est assurer que seules des personnes autorisées auront accès à l'information..

Cryptanalyse : Art d'analyser un message chiffré afin de le décrypter..
Crypter : voir chiffrer..
Cryptogramme : Message chiffré ou codé..
Cryptographie : Discipline incluant les principes, les moyens et les méthodes de transformation des données, dans le but de masquer leur contenu, d'empêcher leur modification ou leur utilisation illégale..
Cryptologie : Science des messages secrets. Se décompose en cryptographie et cryptanalyse..
Cryptosystème : Voir Chiffre..
DES : (Data Encryption Standard) : Algorithme standardisé de chiffrement des données mis au point par IBM..
Dictionnaire : Liste de mots et d'expressions très utilisés servant de base pour la recherche d'un mot de passe. L'utilisation d'un dictionnaire permet d'accélérer notablement le temps de recherche (les combinaisons qui ne veulent rien dire sont évitées) à condition que le mot de passe soit classique (i.e. prénoms, noms, suites de chiffres simples comme 123456, etc.)..
Double clef (chiffre à) : Autre terme pour chiffre polyalphabétique..
Déchiffrement : Opération inverse du chiffrement, i.e. obtenir la version originale d'un message qui a été précédemment chiffré. Ici, on connaît la méthode de chiffrement, contrairement au décryptement..
Décrypter : Parvenir à restaurer des données qui avaient été chiffrées, donc à leur faire retrouver leur état premier (en clair), sans disposer des clefs théoriquement nécessaires..
Encoder : Modifier la structure d'un ensemble de données en lui appliquant un algorithme (chiffrement, méthode de compression...). L'encodage n'a pas forcément un but cryptographique..
Force brute : L'attaque par la force brute est la seule à laquelle aucun algorithme ne résiste : elle consiste à tester toutes les clefs possibles, jusqu'à trouver la bonne. Elle ne constitue pas souvent une bonne approche car elle nécessite souvent des jours, des mois, voire des années pour trouver la clef. On peut aussi l'optimiser en se servant d'un dictionnaire..
Fréquence : Pourcentage d'apparition d'une lettre ou d'un mot dans une langue donnée. Calculer les fréquences d'apparition est souvent la première étape d'un processus de décryptement..
Hachage : Fonction appliquée à un document de longueur variable qui renvoie un nombre de longueur fixe caractéristique du document : c'est l'empreinte du document. Une légère modification du document entraînant une modification visible de l'empreinte, celle-ci permettra de vérifier l'intégrité du document..
IDEA : (International Data Encryption Algorithm) : Algorithme standardisé de chiffrement des données très largement répandu et utilisé notamment dans PGP..
Intégrité : D'un point de vue cryptographique, assurer l'intégrité de données consiste à permettre la détection des modifications volontaires de ces données..
Masque jetable : Seule méthode de chiffrement absolument sûre connue. Elle repose sur une clef aléatoire de même longueur que le message. Chaque clef ne doit être utilisée qu'une seule fois..
Message clair : Version intelligible d'un message et compréhensible par tous..
Monoalphabétique : Se dit d'un chiffre où une lettre du message clair est toujours remplacée par le même symbole. On a donc une bijection entre les lettres claires et les symboles de l'alphabet de chiffrage. Exemple : le chiffre de César..
Nomenclateur : Voir Répertoire..
Nulles : Symboles sans signification rajoutés dans un message pour certains algorithmes. On les emploie soit pour compléter un message de manière à atteindre une certaine

longueur, soit pour tromper ceux qui cherchent à décrypter le message en noyant les informations utiles au milieu de caractères, mots ou phrases inutiles (on parle alors de stéganographie)..

One time pad : Voir masque jetable..

Padding : Ajout de valeurs aléatoires pour obtenir une longueur de message constante..

PGP : (Pretty Good Privacy) Algorithme de chiffrement informatisé développé par Phil Zimmermann et basé sur le RSA..

Polyalphabétique : Se dit d'un chiffre où plusieurs alphabets de chiffrage sont utilisés en même temps. Exemples : le chiffre de Porta et le chiffre de Vigenère..

Polygrammique : Se dit d'un chiffre où un groupe de n lettres est codé par un groupe de n symboles. Exemples : le chiffre de Playfair (avec n = 2) et le chiffre de Hill..

Rot13 : Méthode de chiffrement très simple qui consiste à remplacer un caractère par un autre à 13 caractères de là. A devient N, par exemple. Cas particulier du chiffre de César..

RSA : (Initiales de Rivest, Shamir, Adleman) : Algorithme de chiffrement à clef publique utilisé notamment dans PGP, utilisé principalement pour le chiffrement de la signature, permettant donc l'authentification du document..

Répertoire : Table mettant en correspondance un code (par exemple un nombre mais cela peut aussi être un mot) et sa signification. .

Scytale : Une scytale consiste en un bâton de bois autour duquel est entourée une bande de cuir ou de parchemin..

Signature : Donnée de taille faible qui, jointe à un message, prouve l'identité de l'émetteur du message..

Stéganographie : Branche particulière de la cryptographie qui consiste non pas à rendre le message inintelligible, mais à le camoufler dans un support de manière à masquer sa présence..

Substitution : Un chiffre à substitution remplace les caractères du message en clair par des symboles (caractères, nombres, signes, etc.) définis à l'avance..

Surchiffrement : Fait de chiffrer un message déjà chiffré avec une autre méthode..

Symétrique : Se dit d'un algorithme de cryptage utilisant une même clef pour chiffrer et déchiffrer les messages..

Sémagramme : Dans un sémagramme, les éléments du texte codé ou chiffré ne sont ni des lettres, ni des chiffres : le sens est véhiculé par différents éléments, par exemple des points de jetons de dominos, l'emplacement d'objets sur une image, etc..

Tomogrammique : Dans les systèmes tomogrammiques, chaque lettre est tout d'abord représentée par un groupe de plusieurs symboles. Ces symboles sont ensuite chiffrés séparément ou par groupes de taille fixe. Exemples : le code morse fractionné, le chiffre de Delastelle..

Transposition : Un chiffre de transposition ne modifie pas le contenu du message mais mélange les caractères selon une méthode prédéfinie..

Trigramme : Séquence de trois lettres consécutives. Exemples : ehe, thi, ong, ...

Watermarking : Application particulière de la stéganographie consistant à camoufler dans une image des informations sur son origine (nom de l'auteur, copyright...). .

XOR : Opération logique Ou exclusif : 0+0=0, 0+1=1, 1+0=1, 1+1=0..

Bibliographie

[1] L. S. Hill, « Cryptography in an Algebraic Alphabet », American Mathematical Monthly, 36, 1929, pp. 306-312.

[2] N.I.S.T, « DES MODES OF OPERATION », http://www.itl.nist.gov/fipspubs/fip81.htm.

[3] IBM Corp., « MARS - a candidate cipher for AES », http://www.research.ibm.com/security/mars.pdf.

[4] RSA Laboratories, « The RC6 block cipher », ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf.

[5] J. Daemen et V. Rijmen , « The Rijndael block cipher », http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf.

[6] R. Anderson, E. Biham et L. Knudsen, « Serpent : A Prop osal for the Advanced Encryption Standard », http://www.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf.

[7] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall et N. Ferguson, « Twofish : A 128-Bit Block Cipher », http://www.schneier.com/paper-twofish-paper.pdf.

[8] CNSS, « National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information », NSA, http://www.nstissc.gov/Assets/pdf/fact%20sheet.pdf.

[9] B. Schneier, « NSA on AES », Crypto-Gram Newsletter, 15 Octobre 2000, http://www.schneier.com/crypto-gram-0010.html#9.

[10] N.I.S.T., « ADVANCED ENCRYPTION STANDARD », http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[11] E. Filiol, « Plaintext-dependant Repetition Codes Cryptanalysis of Block Ciphers - The AES Case », http://eprint.iacr.org/2003/003.ps

[12] N. Courtois, R. Johnson, P. Junod, T. Pornin et M. Scott,« Did Filiol Break AES ? », http://eprint.iacr.org/2003/022.pdf

Quelques codes sources des finalistes

I RC6

```
/*
      author: John Hughes (jhughes@frostburg.edu)
    modified: 3/16/02
*/

#include "platform.h"

/*
    function: rc6_generateKeySchedule

    description: This function takes a 16-, 24-, or 32-byte key and
                 generates the RC6 key schedule in array S.
*/

void rc6_generateKeySchedule(unsigned char* initKey,
                             unsigned int keyLength,
                             unsigned int S[]
                            )
{
unsigned int L[8];  /* We need up to 32 bytes. */
    register unsigned int A, B, i, j, s, v;

    /* Point to the lowest byte of L. */

    unsigned char* bPtr = (unsigned char*)L;

    /* Move the bytes of initKey into L, little-endian fashion. */

    memcpy(bPtr, initKey, keyLength);

    /* Set S[0] to the constant P32, then generate the rest of S. */

    S[0] = 0xB7E15163;
    for (i = 1; i < 44; i++)
S[i] = S[i - 1] + 0x9E3779B9;
```

```
    A = B = i = j = 0;
v = 132;
    keyLength >>= 2;
for (s = 1; s <= v; s++)
{
A = S[i] = ROL(S[i] + A + B, 3);
B = L[j] = ROL(L[j] + A + B, A + B);
i = (i + 1) \% 44;
j = (j + 1) \% keyLength;
}
}

/*
    function: rc6_encrypt

    description: This function takes a 16-byte block and encrypts it
into 'output'.
*/

void rc6_encrypt(unsigned char* input, unsigned int S[], unsigned char* output)
{
    register unsigned int A, B, C, D;
    unsigned int regs[4];
    register unsigned int t, u, temp, j;
unsigned char* regPtr;

    regPtr = (unsigned char*)&regs[0];
    memcpy(regPtr, input, 16);
    A = regs[0]; /* Cook up A, B, C, and D as our four 32-bit registers. */
    B = regs[1];
    C = regs[2];
    D = regs[3];
B += S[0];
D += S[1];
for (j = 1; j <= 20; j++)  /* Perform 20 rounds. */
{
    t = ROL(B * ((B << 1) + 1), 5);
        u = ROL(D * ((D << 1) + 1), 5);
        A = ROL(A ^ t, u) + S[j << 1];
        C = ROL(C ^ u, t) + S[(j << 1) + 1];
temp = A;
A = B;
B = C;
C = D;
D = temp;
    }
A += S[42];
```

```
C += S[43];
    regs[0] = A;
    regs[1] = B;
    regs[2] = C;
    regs[3] = D;
    memcpy(output, regPtr, 16);
}

/*
    function: rc6_decrypt

    description: This function takes a 16-byte block and decrypts it into
                 'output.'
*/

void rc6_decrypt(unsigned char* input, unsigned int S[], unsigned char* output)
{
    register unsigned int A, B, C, D;
    unsigned int regs[4];
register unsigned int t, u, temp, temp2, j;
unsigned char* regPtr;

    regPtr = (unsigned char*)&regs[0];
    memcpy(regPtr, input, 16);
    A = regs[0];
    B = regs[1];
    C = regs[2];
    D = regs[3];
C -= S[43];
A -= S[42];
for (j = 20; j >= 1; j--)
{
    temp = A;
A = D;
temp2 = B;
B = temp;
temp = C;
C = temp2;
D = temp;
t = ROL(B * ((B << 1) + 1), 5);
u = ROL(D * ((D << 1) + 1), 5);
A = ROR(A - S[j << 1], u) ^ t;
C = ROR(C - S[(j << 1) + 1], t) ^ u;
}
D -= S[1];
B -= S[0];
    regs[0] = A;
```

```
    regs[1] = B;
    regs[2] = C;
    regs[3] = D;
    memcpy(output, regPtr, 16);
}
```

II Rijndael

```
/* rijndael - An implementation of the Rijndael cipher.
 * Copyright (C) 2000, 2001 Rafael R. Sevilla <sevillar@team.ph.inter.net>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the Free
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "rijndael.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#if 0

static void
print_block(UINT8 *block)
{
  int i;

  for (i=0; i<RIJNDAEL_BLOCKSIZE; i++)
    printf("%02x", block[i]);
  printf("\n");
}
#endif

/* These tables combine both the S-boxes and the mixcolumn transformation, so
```

```
   that we can perform a round's encryption or by means of four table lookups
   and four XOR's per column of state.  They were generated by the
   makertbls.pl script. */
UINT32 dtbl[] = {
  0xa56363c6, 0x847c7cf8, 0x997777ee, 0x8d7b7bf6,
  0x0df2f2ff, 0xbd6b6bd6, 0xb16f6fde, 0x54c5c591,
  0x50303060, 0x03010102, 0xa96767ce, 0x7d2b2b56,
  0x19fefee7, 0x62d7d7b5, 0xe6abab4d, 0x9a7676ec,
  0x45caca8f, 0x9d82821f, 0x40c9c989, 0x877d7dfa,
  0x15fafaef, 0xeb5959b2, 0xc947478e, 0x0bf0f0fb,
  0xecadad41, 0x67d4d4b3, 0xfda2a25f, 0xeaafaf45,
  0xbf9c9c23, 0xf7a4a453, 0x967272e4, 0x5bc0c09b,
  0xc2b7b775, 0x1cfdfde1, 0xae93933d, 0x6a26264c,
  0x5a36366c, 0x413f3f7e, 0x02f7f7f5, 0x4fcccc83,
  0x5c343468, 0xf4a5a551, 0x34e5e5d1, 0x08f1f1f9,
  0x937171e2, 0x73d8d8ab, 0x53313162, 0x3f15152a,
  0x0c040408, 0x52c7c795, 0x65232346, 0x5ec3c39d,
  0x28181830, 0xa1969637, 0x0f05050a, 0xb59a9a2f,
  0x0907070e, 0x36121224, 0x9b80801b, 0x3de2e2df,
  0x26ebebcd, 0x6927274e, 0xcdb2b27f, 0x9f7575ea,
  0x1b090912, 0x9e83831d, 0x742c2c58, 0x2e1a1a34,
  0x2d1b1b36, 0xb26e6edc, 0xee5a5ab4, 0xfba0a05b,
  0xf65252a4, 0x4d3b3b76, 0x61d6d6b7, 0xceb3b37d,
  0x7b292952, 0x3ee3e3dd, 0x712f2f5e, 0x97848413,
  0xf55353a6, 0x68d1d1b9, 0x00000000, 0x2cededc1,
  0x60202040, 0x1ffcfce3, 0xc8b1b179, 0xed5b5bb6,
  0xbe6a6ad4, 0x46cbcb8d, 0xd9bebe67, 0x4b393972,
  0xde4a4a94, 0xd44c4c98, 0xe85858b0, 0x4acfcf85,
  0x6bd0d0bb, 0x2aefefc5, 0xe5aaaa4f, 0x16fbfbed,
  0xc5434386, 0xd74d4d9a, 0x55333366, 0x94858511,
  0xcf45458a, 0x10f9f9e9, 0x06020204, 0x817f7ffe,
  0xf05050a0, 0x443c3c78, 0xba9f9f25, 0xe3a8a84b,
  0xf35151a2, 0xfea3a35d, 0xc0404080, 0x8a8f8f05,
  0xad92923f, 0xbc9d9d21, 0x48383870, 0x04f5f5f1,
  0xdfbcbc63, 0xc1b6b677, 0x75dadaaf, 0x63212142,
  0x30101020, 0x1affffe5, 0x0ef3f3fd, 0x6dd2d2bf,
  0x4ccdcd81, 0x140c0c18, 0x35131326, 0x2fececc3,
  0xe15f5fbe, 0xa2979735, 0xcc444488, 0x3917172e,
  0x57c4c493, 0xf2a7a755, 0x827e7efc, 0x473d3d7a,
  0xac6464c8, 0xe75d5dba, 0x2b191932, 0x957373e6,
  0xa06060c0, 0x98818119, 0xd14f4f9e, 0x7fdcdca3,
  0x66222244, 0x7e2a2a54, 0xab90903b, 0x8388880b,
  0xca46468c, 0x29eeeec7, 0xd3b8b86b, 0x3c141428,
  0x79dedea7, 0xe25e5ebc, 0x1d0b0b16, 0x76dbdbad,
  0x3be0e0db, 0x56323264, 0x4e3a3a74, 0x1e0a0a14,
  0xdb494992, 0x0a06060c, 0x6c242448, 0xe45c5cb8,
  0x5dc2c29f, 0x6ed3d3bd, 0xefacac43, 0xa66262c4,
```

```
  0xa8919139, 0xa4959531, 0x37e4e4d3, 0x8b7979f2,
  0x32e7e7d5, 0x43c8c88b, 0x5937376e, 0xb76d6dda,
  0x8c8d8d01, 0x64d5d5b1, 0xd24e4e9c, 0xe0a9a949,
  0xb46c6cd8, 0xfa5656ac, 0x07f4f4f3, 0x25eaeacf,
  0xaf6565ca, 0x8e7a7af4, 0xe9aeae47, 0x18080810,
  0xd5baba6f, 0x887878f0, 0x6f25254a, 0x722e2e5c,
  0x241c1c38, 0xf1a6a657, 0xc7b4b473, 0x51c6c697,
  0x23e8e8cb, 0x7cdddda1, 0x9c7474e8, 0x211f1f3e,
  0xdd4b4b96, 0xdcbdbd61, 0x868b8b0d, 0x858a8a0f,
  0x907070e0, 0x423e3e7c, 0xc4b5b571, 0xaa6666cc,
  0xd8484890, 0x05030306, 0x01f6f6f7, 0x120e0e1c,
  0xa36161c2, 0x5f35356a, 0xf95757ae, 0xd0b9b969,
  0x91868617, 0x58c1c199, 0x271d1d3a, 0xb99e9e27,
  0x38e1e1d9, 0x13f8f8eb, 0xb398982b, 0x33111122,
  0xbb6969d2, 0x70d9d9a9, 0x898e8e07, 0xa7949433,
  0xb69b9b2d, 0x221e1e3c, 0x92878715, 0x20e9e9c9,
  0x49cece87, 0xff5555aa, 0x78282850, 0x7adfdfa5,
  0x8f8c8c03, 0xf8a1a159, 0x80898909, 0x170d0d1a,
  0xdabfbf65, 0x31e6e6d7, 0xc6424284, 0xb86868d0,
  0xc3414182, 0xb0999929, 0x772d2d5a, 0x110f0f1e,
  0xcbb0b07b, 0xfc5454a8, 0xd6bbbb6d, 0x3a16162c,
};

UINT32 itbl[] = {
  0x50a7f451, 0x5365417e, 0xc3a4171a, 0x965e273a,
  0xcb6bab3b, 0xf1459d1f, 0xab58faac, 0x9303e34b,
  0x55fa3020, 0xf66d76ad, 0x9176cc88, 0x254c02f5,
  0xfcd7e54f, 0xd7cb2ac5, 0x80443526, 0x8fa362b5,
  0x495ab1de, 0x671bba25, 0x980eea45, 0xe1c0fe5d,
  0x02752fc3, 0x12f04c81, 0xa397468d, 0xc6f9d36b,
  0xe75f8f03, 0x959c9215, 0xeb7a6dbf, 0xda595295,
  0x2d83bed4, 0xd3217458, 0x2969e049, 0x44c8c98e,
  0x6a89c275, 0x78798ef4, 0x6b3e5899, 0xdd71b927,
  0xb64fe1be, 0x17ad88f0, 0x66ac20c9, 0xb43ace7d,
  0x184adf63, 0x82311ae5, 0x60335197, 0x457f5362,
  0xe07764b1, 0x84ae6bbb, 0x1ca081fe, 0x942b08f9,
  0x58684870, 0x19fd458f, 0x876cde94, 0xb7f87b52,
  0x23d373ab, 0xe2024b72, 0x578f1fe3, 0x2aab5566,
  0x0728ebb2, 0x03c2b52f, 0x9a7bc586, 0xa50837d3,
  0xf2872830, 0xb2a5bf23, 0xba6a0302, 0x5c8216ed,
  0x2b1ccf8a, 0x92b479a7, 0xf0f207f3, 0xa1e2694e,
  0xcdf4da65, 0xd5be0506, 0x1f6234d1, 0x8afea6c4,
  0x9d532e34, 0xa055f3a2, 0x32e18a05, 0x75ebf6a4,
  0x39ec830b, 0xaaef6040, 0x069f715e, 0x51106ebd,
  0xf98a213e, 0x3d06dd96, 0xae053edd, 0x46bde64d,
  0xb58d5491, 0x055dc471, 0x6fd40604, 0xff155060,
  0x24fb9819, 0x97e9bdd6, 0xcc434089, 0x779ed967,
```

```
  0xbd42e8b0, 0x888b8907, 0x385b19e7, 0xdbeec879,
  0x470a7ca1, 0xe90f427c, 0xc91e84f8, 0x00000000,
  0x83868009, 0x48ed2b32, 0xac70111e, 0x4e725a6c,
  0xfbff0efd, 0x5638850f, 0x1ed5ae3d, 0x27392d36,
  0x64d90f0a, 0x21a65c68, 0xd1545b9b, 0x3a2e3624,
  0xb1670a0c, 0x0fe75793, 0xd296eeb4, 0x9e919b1b,
  0x4fc5c080, 0xa220dc61, 0x694b775a, 0x161a121c,
  0x0aba93e2, 0xe52aa0c0, 0x43e0223c, 0x1d171b12,
  0x0b0d090e, 0xadc78bf2, 0xb9a8b62d, 0xc8a91e14,
  0x8519f157, 0x4c0775af, 0xbbdd99ee, 0xfd607fa3,
  0x9f2601f7, 0xbcf5725c, 0xc53b6644, 0x347efb5b,
  0x7629438b, 0xdcc623cb, 0x68fcedb6, 0x63f1e4b8,
  0xcadc31d7, 0x10856342, 0x40229713, 0x2011c684,
  0x7d244a85, 0xf83dbbd2, 0x1132f9ae, 0x6da129c7,
  0x4b2f9e1d, 0xf330b2dc, 0xec52860d, 0xd0e3c177,
  0x6c16b32b, 0x99b970a9, 0xfa489411, 0x2264e947,
  0xc48cfca8, 0x1a3ff0a0, 0xd82c7d56, 0xef903322,
  0xc74e4987, 0xc1d138d9, 0xfea2ca8c, 0x360bd498,
  0xcf81f5a6, 0x28de7aa5, 0x268eb7da, 0xa4bfad3f,
  0xe49d3a2c, 0x0d927850, 0x9bcc5f6a, 0x62467e54,
  0xc2138df6, 0xe8b8d890, 0x5ef7392e, 0xf5afc382,
  0xbe805d9f, 0x7c93d069, 0xa92dd56f, 0xb31225cf,
  0x3b99acc8, 0xa77d1810, 0x6e639ce8, 0x7bbb3bdb,
  0x097826cd, 0xf418596e, 0x01b79aec, 0xa89a4f83,
  0x656e95e6, 0x7ee6ffaa, 0x08cfbc21, 0xe6e815ef,
  0xd99be7ba, 0xce366f4a, 0xd4099fea, 0xd67cb029,
  0xafb2a431, 0x31233f2a, 0x3094a5c6, 0xc066a235,
  0x37bc4e74, 0xa6ca82fc, 0xb0d090e0, 0x15d8a733,
  0x4a9804f1, 0xf7daec41, 0x0e50cd7f, 0x2ff69117,
  0x8dd64d76, 0x4db0ef43, 0x544daacc, 0xdf0496e4,
  0xe3b5d19e, 0x1b886a4c, 0xb81f2cc1, 0x7f516546,
  0x04ea5e9d, 0x5d358c01, 0x737487fa, 0x2e410bfb,
  0x5a1d67b3, 0x52d2db92, 0x335610e9, 0x1347d66d,
  0x8c61d79a, 0x7a0ca137, 0x8e14f859, 0x893c13eb,
  0xee27a9ce, 0x35c961b7, 0xede51ce1, 0x3cb1477a,
  0x59dfd29c, 0x3f73f255, 0x79ce1418, 0xbf37c773,
  0xeacdf753, 0x5baafd5f, 0x146f3ddf, 0x86db4478,
  0x81f3afca, 0x3ec468b9, 0x2c342438, 0x5f40a3c2,
  0x72c31d16, 0x0c25e2bc, 0x8b493c28, 0x41950dff,
  0x7101a839, 0xdeb30c08, 0x9ce4b4d8, 0x90c15664,
  0x6184cb7b, 0x70b632d5, 0x745c6c48, 0x4257b8d0,
};


/* Needed only for the key schedule and for final rounds */
UINT8 sbox[256] = {
  99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171,
```

```
  118, 202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164,
  114, 192, 183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113,
  216,  49,  21,  4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226,
  235,  39, 178, 117,  9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214,
  179,  41, 227,  47, 132, 83, 209,   0, 237,  32, 252, 177,  91, 106, 203,
  190,  57,  74,  76,  88, 207, 208, 239, 170, 251,  67,  77,  51, 133,  69,
  249,   2, 127,  80,  60, 159, 168, 81, 163,  64, 143, 146, 157,  56, 245,
  188, 182, 218,  33,  16, 255, 243, 210, 205,  12,  19, 236,  95, 151,  68,
  23, 196, 167, 126,  61, 100,  93,  25, 115, 96, 129,  79, 220,  34,  42,
  144, 136,  70, 238, 184,  20, 222,  94,  11, 219, 224,  50,  58,  10,  73,
  6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121, 231, 200,  55, 109,
  141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8, 186, 120,  37,
  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138, 112,  62,
  181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158, 225,
  248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
  140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,
  22,
};

UINT8 isbox[256] = {
  82,   9, 106, 213,  48,  54, 165,  56, 191,  64, 163, 158, 129, 243, 215,
  251, 124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,  68, 196, 222,
  233, 203, 84, 123, 148,  50, 166, 194,  35,  61, 238,  76, 149,  11,  66,
  250, 195,  78,  8,  46, 161, 102,  40, 217,  36, 178, 118,  91, 162,  73,
  109, 139, 209,  37, 114, 248, 246, 100, 134, 104, 152,  22, 212, 164,  92,
  204,  93, 101, 182, 146, 108, 112,  72,  80, 253, 237, 185, 218,  94,  21,
  70,  87, 167, 141, 157, 132, 144, 216, 171,   0, 140, 188, 211,  10, 247,
  228,  88,   5, 184, 179,  69,   6, 208,  44,  30, 143, 202,  63,  15,   2,
  193, 175, 189,   3,   1,  19, 138, 107,  58, 145,  17,  65,  79, 103, 220,
  234, 151, 242, 207, 206, 240, 180, 230, 115, 150, 172, 116,  34, 231, 173,
  53, 133, 226, 249,  55, 232,  28, 117, 223, 110,  71, 241,  26, 113,  29,
  41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27, 252,  86,  62,  75,
  198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,  31, 221, 168,
  51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,  95,  96,  81,
  127, 169,  25, 181,  74,  13,  45, 229, 122, 159, 147, 201, 156, 239, 160,
  224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,  83, 153,  97,
  23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,  85,  33,  12,
  125,
};

/* Used only by the key schedule */
UINT8 Logtable[256] = {
  0,   0,  25,   1,  50,   2,  26, 198,  75, 199,  27, 104,  51, 238, 223,  3,
  100,   4, 224,  14,  52, 141, 129, 239,  76, 113,   8, 200, 248, 105,  28,
  193, 125, 194,  29, 181, 249, 185,  39, 106,  77, 228, 166, 114, 154, 201,
  9, 120, 101,  47, 138,   5,  33,  15, 225,  36,  18, 240, 130,  69,  53,
  147, 218, 142, 150, 143, 219, 189,  54, 208, 206, 148,  19,  92, 210, 241,
```

```
  64,  70, 131,  56, 102, 221, 253,  48, 191,   6, 139,  98, 179,  37, 226,
  152,  34, 136, 145,  16, 126, 110,  72, 195, 163, 182,  30,  66,  58, 107,
  40,  84, 250, 133,  61, 186, 43, 121,  10,  21, 155, 159,  94, 202,  78,
  212, 172, 229, 243, 115, 167,  87, 175,  88, 168,  80, 244, 234, 214, 116,
  79, 174, 233, 213, 231, 230, 173, 232, 44, 215, 117, 122, 235,  22,  11,
  245,  89, 203,  95, 176, 156, 169,  81, 160, 127,  12, 246, 111,  23, 196,
  73, 236, 216,  67,  31,  45, 164, 118, 123, 183, 204, 187,  62,  90, 251,
  96, 177, 134,  59,  82, 161, 108, 170,  85,  41, 157, 151, 178, 135, 144,
  97, 190, 220, 252, 188, 149, 207, 205,  55,  63,  91, 209, 83,  57, 132,
  60,  65, 162, 109,  71,  20,  42, 158,  93,  86, 242, 211, 171, 68,  17,
  146, 217,  35,  32,  46, 137, 180, 124, 184,  38, 119, 153, 227, 165, 103,
  74, 237, 222, 197,  49, 254,  24,  13,  99, 140, 128, 192, 247, 112,   7,
};

UINT8 Alogtable[256] = {
  1,   3,   5,  15,  17,  51,  85, 255,  26,  46, 114, 150, 161, 248,  19,
  53, 95, 225,  56,  72, 216, 115, 149, 164, 247,   2,   6,  10,  30,  34,
  102, 170, 229,  52,  92, 228,  55,  89, 235,  38, 106, 190, 217, 112, 144,
  171, 230,  49,  83, 245,   4,  12,  20,  60,  68, 204,  79, 209, 104, 184,
  211, 110, 178, 205,  76, 212, 103, 169, 224,  59,  77, 215,  98, 166, 241,
  8,  24,  40, 120, 136, 131, 158, 185, 208, 107, 189, 220, 127, 129, 152,
  179, 206,  73, 219, 118, 154, 181, 196,  87, 249,  16,  48,  80, 240,  11,
  29,  39, 105, 187, 214,  97, 163, 254,  25,  43, 125, 135, 146, 173, 236,
  47, 113, 147, 174, 233,  32,  96, 160, 251,  22,  58,  78, 210, 109, 183,
  194,  93, 231,  50,  86, 250,  21,  63,  65, 195,  94, 226,  61,  71, 201,
  64, 192,  91, 237,  44, 116, 156, 191, 218, 117, 159, 186, 213, 100, 172,
  239,  42, 126, 130, 157, 188, 223, 122, 142, 137, 128, 155, 182, 193,  88,
  232,  35, 101, 175, 234,  37, 111, 177, 200,  67, 197,  84, 252,  31,  33,
  99, 165, 244,   7,   9,  27,  45, 119, 153, 176, 203,  70, 202,  69, 207,
  74, 222, 121, 139, 134, 145, 168, 227,  62,  66, 198,  81, 243,  14,  18,
  54,  90, 238,  41, 123, 141, 140, 143, 138, 133, 148, 167, 242,  13,  23,
  57,  75, 221, 124, 132, 151, 162, 253,  28,  36, 108, 180, 199,  82, 246, 1,
};

#define ROTBYTE(x) (((x) >> 8) | (((x) & 0xff) << 24))
#define ROTRBYTE(x) (((x) << 8) | (((x) >> 24) & 0xff))
#define SUBBYTE(x, box) (((box)[((x) & 0xff)]) | \
                         ((box)[(((x) >> 8) & 0xff)] << 8) | \
                         ((box)[(((x) >> 16) & 0xff)] << 16) | \
                         ((box)[(((x) >> 24) & 0xff)] << 24))

static UINT8
xtime(UINT8 a)
{
  UINT8 b;

  b = (a & 0x80) ? 0x1b : 0;
```

```
  a<<=1;
  a^=b;
  return(a);
}

static UINT8
mul(UINT8 a, UINT8 b)
{
  if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
  else return 0;
}

static void
inv_mix_column(UINT32 *a, UINT32 *b)
{
  UINT8 c[4][4];
  int i, j;

  for(j = 0; j < 4; j++) {
    for(i = 0; i < 4; i++) {
      c[j][i] = mul(0xe, (a[j] >> i*8) & 0xff)
^ mul(0xb, (a[j] >> ((i+1)%4)*8) & 0xff)
^ mul(0xd, (a[j] >> ((i+2)%4)*8) & 0xff)
^ mul(0x9, (a[j] >> ((i+3)%4)*8) & 0xff);
    }
  }
  for(i = 0; i < 4; i++) {
    b[i] = 0;
    for(j = 0; j < 4; j++)
      b[i] |= c[i][j] << (j*8);
  }
}

void
rijndael_setup(RIJNDAEL_context *ctx, size_t keysize, const UINT8 *key)
{
  int nk, nr, i, lastkey;
  UINT32 temp, rcon;

  /* Truncate keysizes to the valid key sizes provided by Rijndael */
  if (keysize >= 32) {
    nk = 8;
    nr = 14;
  } else if (keysize >= 24) {
    nk = 6;
    nr = 12;
  } else { /* must be 16 or more */
```

```
    nk = 4;
    nr = 10;
  }

  lastkey = (RIJNDAEL_BLOCKSIZE/4) * (nr + 1);
  ctx->nrounds = nr;
  rcon = 1;
  for (i=0; i<nk; i++) {
    ctx->keys[i] = key[i*4] + (key[i*4+1]<<8) + (key[i*4+2]<<16) +
      (key[i*4+3]<<24);
  }

  for (i=nk; i<lastkey; i++) {
    temp = ctx->keys[i-1];
    if (i % nk == 0) {
      temp = SUBBYTE(ROTBYTE(temp), sbox) ^ rcon;
      rcon = (UINT32)xtime((UINT8)rcon&0xff);
    } else if (nk > 6 && (i%nk) == 4) {
      temp = SUBBYTE(temp, sbox);
    }
    ctx->keys[i] = ctx->keys[i-nk] ^ temp;
  }
  /* Generate the inverse keys */
  for (i=0; i<4; i++) {
    ctx->ikeys[i] = ctx->keys[i];
    ctx->ikeys[lastkey-4 + i] = ctx->keys[lastkey-4 + i];
  }
  for (i=4; i<lastkey-4; i+=4)
    inv_mix_column(&(ctx->keys[i]), &(ctx->ikeys[i]));
}

/* Key addition that also packs every byte in the key to a word rep. */
static void
key_addition_8to32(const UINT8 *txt, UINT32 *keys, UINT32 *out)
{
  const UINT8 *ptr;
  int i, j;
  UINT32 val;

  ptr = txt;
  for (i=0; i<4; i++) {
    val = 0;
    for (j=0; j<4; j++)
      val |= (*ptr++ << 8*j);
    out[i] = keys[i]^val;
  }
}
```

```
static void
key_addition32(const UINT32 *txt, UINT32 *keys, UINT32 *out)
{
  int i;

  for (i=0; i<4; i++)
    out[i] = keys[i] ^ txt[i];
}

static void
key_addition32to8(const UINT32 *txt, UINT32 *keys, UINT8 *out)
{
  UINT8 *ptr;
  int i, j;
  UINT32 val;

  ptr = out;
  for (i=0; i<4; i++) {
    val = txt[i] ^ keys[i];
    for (j=0; j<4; j++)
      *ptr++ = (val >> 8*j) & 0xff;
  }
}

static int idx[4][4] = {
  { 0, 1, 2, 3 },
  { 1, 2, 3, 0 },
  { 2, 3, 0, 1 },
  { 3, 0, 1, 2 } };

void
rijndael_encrypt(RIJNDAEL_context *ctx,
 const UINT8 *plaintext,
 UINT8 *ciphertext)
{
  int r, j;
  UINT32 wtxt[4], t[4]; /* working ciphertext */
  UINT32 e;

  key_addition_8to32(plaintext, &(ctx->keys[0]), wtxt);
  for (r=1; r<ctx->nrounds; r++) {
    for (j=0; j<4; j++) {
      t[j] = dtbl[wtxt[j] & 0xff] ^
ROTRBYTE(dtbl[(wtxt[idx[1][j]] >> 8) & 0xff]^
 ROTRBYTE(dtbl[(wtxt[idx[2][j]] >> 16) & 0xff] ^
  ROTRBYTE(dtbl[(wtxt[idx[3][j]] >> 24) & 0xff])));
```

```
    }
    key_addition32(t, &(ctx->keys[r*4]), wtxt);
  }
  /* last round is special: there is no mixcolumn, so we can't use the big
     tables. */
  for (j=0; j<4; j++) {
    e = wtxt[j] & 0xff;
    e |= (wtxt[idx[1][j]]) & (0xff << 8);
    e |= (wtxt[idx[2][j]]) & (0xff << 16);
    e |= (wtxt[idx[3][j]]) & (0xff << 24);
    t[j] = e;
  }
  for (j=0; j<4; j++)
    t[j] = SUBBYTE(t[j], sbox);
  key_addition32to8(t, &(ctx->keys[4*ctx->nrounds]), ciphertext);
}

static int iidx[4][4] = {
  { 0, 1, 2, 3 },
  { 3, 0, 1, 2 },
  { 2, 3, 0, 1 },
  { 1, 2, 3, 0 } };

void
rijndael_decrypt(RIJNDAEL_context *ctx,
 const UINT8 *ciphertext,
 UINT8 *plaintext)
{
  int r, j;
  UINT32 wtxt[4], t[4]; /* working ciphertext */
  UINT32 e;

  key_addition_8to32(ciphertext, &(ctx->ikeys[4*ctx->nrounds]), wtxt);
  for (r=ctx->nrounds-1; r> 0;  r--) {
    for (j=0; j<4; j++) {
      t[j] = itbl[wtxt[j] & 0xff] ^
ROTRBYTE(itbl[(wtxt[iidx[1][j]] >> 8) & 0xff]^
 ROTRBYTE(itbl[(wtxt[iidx[2][j]] >> 16) & 0xff] ^
  ROTRBYTE(itbl[(wtxt[iidx[3][j]] >> 24) & 0xff])));
    }
    key_addition32(t, &(ctx->ikeys[r*4]), wtxt);
  }
  /* last round is special: there is no mixcolumn, so we can't use the big
     tables. */
  for (j=0; j<4; j++) {
    e = wtxt[j] & 0xff;
    e |= (wtxt[iidx[1][j]]) & (0xff << 8);
```

```
    e |= (wtxt[iidx[2][j]]) & (0xff << 16);
    e |= (wtxt[iidx[3][j]]) & (0xff << 24);
    t[j] = e;
  }
  for (j=0; j<4; j++)
    t[j] = SUBBYTE(t[j], isbox);
  key_addition32to8(t, &(ctx->ikeys[0]), plaintext);
}

void
block_encrypt(RIJNDAEL_context *ctx, UINT8 *input, int inputlen,
   UINT8 *output, UINT8 *iv)
{
  int i, j, nblocks, carry_flg;
  UINT8 block[RIJNDAEL_BLOCKSIZE], block2[RIJNDAEL_BLOCKSIZE], oldptxt;

  nblocks = inputlen / RIJNDAEL_BLOCKSIZE;

  switch (ctx->mode) {
  case MODE_ECB: /* electronic code book */
    for (i = 0; i<nblocks; i++) {
      rijndael_encrypt(ctx, &input[RIJNDAEL_BLOCKSIZE*i],
       &output[RIJNDAEL_BLOCKSIZE*i]);
    }
    break;
  case MODE_CBC: /* Cipher block chaining */
    /* set initial value */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i< nblocks; i++) {
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++)
block[j] ^= input[i*RIJNDAEL_BLOCKSIZE + j] & 0xff;
      rijndael_encrypt(ctx, block, block);
      memcpy(&output[RIJNDAEL_BLOCKSIZE*i], block, RIJNDAEL_BLOCKSIZE);
    }
    break;
  case MODE_CFB: /* 128-bit cipher feedback */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
      rijndael_encrypt(ctx, block, block);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++)
block[j] ^= input[i*RIJNDAEL_BLOCKSIZE + j];
      memcpy(&output[RIJNDAEL_BLOCKSIZE*i], block, RIJNDAEL_BLOCKSIZE);
    }
    break;
  case MODE_OFB: /* 128-bit output feedback */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
```

```
      rijndael_encrypt(ctx, block, block);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[RIJNDAEL_BLOCKSIZE*i + j] = block[j] ^
  input[RIJNDAEL_BLOCKSIZE*i + j];
      }
    }
    break;
  case MODE_CTR: /* counter */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
      rijndael_encrypt(ctx, block, block2);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[RIJNDAEL_BLOCKSIZE*i + j] = block2[j] ^
  input[RIJNDAEL_BLOCKSIZE*i + j];
      }
      block[RIJNDAEL_BLOCKSIZE-1]++;
      carry_flg = block[RIJNDAEL_BLOCKSIZE-1] ? 0 : 1;
      for (j=RIJNDAEL_BLOCKSIZE-2; j>=0; j--) {
if (carry_flg) {
  block[j]++;
         carry_flg = block[j] ? 0 : 1;
       } else
  break;
      }
    }
    break;
  default:
    break;
  }
}

void
block_decrypt(RIJNDAEL_context *ctx, UINT8 *input, int inputlen,
   UINT8 *output, UINT8 *iv)
{
  int i, j, nblocks, carry_flg;
  UINT8 block[RIJNDAEL_BLOCKSIZE], block2[RIJNDAEL_BLOCKSIZE];

  nblocks = inputlen / RIJNDAEL_BLOCKSIZE;
  switch (ctx->mode) {
  case MODE_ECB:
    for (i = 0; i<nblocks; i++) {
      rijndael_decrypt(ctx, &input[RIJNDAEL_BLOCKSIZE*i],
       &output[RIJNDAEL_BLOCKSIZE*i]);
    }
    break;
  case MODE_CBC:
```

```
    /* first block */
    rijndael_decrypt(ctx, input, block);
    /* XOR the block with the IV to get the output */
    for (i=0; i<RIJNDAEL_BLOCKSIZE; i++)
      output[i] = block[i] ^ iv[i];
    for (i=1; i<nblocks; i++) {
      rijndael_decrypt(ctx, &input[i*RIJNDAEL_BLOCKSIZE], block);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[i*RIJNDAEL_BLOCKSIZE + j] = block[j] ^
  input[(i-1)*RIJNDAEL_BLOCKSIZE + j];
      }
    }
    break;
  case MODE_CFB: /* 128-bit cipher feedback */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
      rijndael_encrypt(ctx, block, block); /* ENCRYPT is right! */
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[RIJNDAEL_BLOCKSIZE*i + j] = block[j] ^
  input[RIJNDAEL_BLOCKSIZE*i + j];
      }
      memcpy(block, &input[RIJNDAEL_BLOCKSIZE*i], RIJNDAEL_BLOCKSIZE);
    }
    break;
  case MODE_OFB: /* 128-bit output feedback */
    /* this is exactly the same as encryption in OFB...in fact you can use
       the encryption in OFB mode to decrypt! */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
      rijndael_encrypt(ctx, block, block);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[RIJNDAEL_BLOCKSIZE*i + j] = block[j] ^
  input[RIJNDAEL_BLOCKSIZE*i + j];
      }
    }
    break;
  case MODE_CTR: /* counter */
    memcpy(block, iv, RIJNDAEL_BLOCKSIZE);
    for (i=0; i<nblocks; i++) {
      rijndael_encrypt(ctx, block, block2);
      for (j=0; j<RIJNDAEL_BLOCKSIZE; j++) {
output[RIJNDAEL_BLOCKSIZE*i + j] = block2[j] ^
  input[RIJNDAEL_BLOCKSIZE*i + j];
      }
      block[RIJNDAEL_BLOCKSIZE-1]++;
      carry_flg = block[RIJNDAEL_BLOCKSIZE-1] ? 0 : 1;
      for (j=RIJNDAEL_BLOCKSIZE-2; j>=0; j--) {
```

```
if (carry_flg) {
  block[j]++;
          carry_flg = block[j] ? 0 : 1;
        } else
  break;
      }
    }
    break;
  default:
    break;
  }
}
```