MASTER EN SCIENCES INFORMATIQUES

De l'utilisation de fonctions d'authentification pour la vérification de l'intégrité des fichiers

Gillet, Patrick

Award date:
1988

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1987-1988

DE L'UTILISATION DE FONCTIONS D'AUTHENTIFICATION POUR LA VERIFICATION DE L'INTEGRITE DES FICHIERS

Patrick GILLET

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en informatique

RESUME

Une modification criminelle des programmes ou des données peut entraîner des comportements non souhaités de la part des systèmes informatiques. C'est pourquoi une technique permettant de détecter l'altération de ces informations constitue un outil précieux pour la sécurité.

Les fonctions d'authentification offrent la possibilité de calculer un code à partir de tout fichier. Si l'intégrité de ce fichier est mise en doute, le code est généré à nouveau et comparé avec le premier. En cas de différence, une manipulation est diagnostiquée.

Une bonne fonction doit être telle qu'un ennemi ne puisse pas trouver un fichier frauduleux ayant le même code qu'un fichier protégé, sinon il pourra faire l'échange sans que cela ne soit détecté. Elle doit également être rapide pour entraîner le moins de désagréments possibles pour l'utilisateur.

Si une telle fonction est trouvée, elle peut servir de base à un système assurant la protection des fichiers. Celui-ci doit empêcher un fraudeur de passer outre le diagnostic de la fonction. Ceci est possible, mais le prix à payer est parfois bien lourd.

SUMMARY

A program or data criminal modification can lead to undesired behavior of computer systems. This is why technics allowing alteration detection of these informations is a valuable tool for security.

Authentification functions give the possibility to calculate a code from a file. If the integrity of the file is suspicious, the code is generated again and compared with the first one. In case of difference, a manipulation is diagnosed.

A good function must be such that an enemy can't find a fraudulent file having the same code as one protected, otherwise he will be able to make an exchange without detection. It must also be rapid enough to involve only little annoyance to user.

If such a function is find, it can be used to be the heart of a system ensuring program and data protection. The system must be designed in such a way that the attacker can't avoid the function diagnostics. This is possible, but the price to pay is sometimes heavy.

Qu'il me soit permis de remercier ici toutes les personnes qui m'ont aidé à réaliser ce mémoire, et en particulier :

- Le personnel du Centre de Technologies Informatique (C.T.I.) à Fontenay aux Roses.

- Monsieur Jean Jacques QUISQUATER pour avoir accepté si gentiment de me consacrer une partie de son précieux temps afin de me recevoir et de me faire profiter de sa grande expérience.

- Mademoiselle Cécile MAHIAT pour sa grande disponibilité et les suggestions pertinentes dont elle m'a fait part.

- Monsieur Ronald FERREIRA pour avoir été à l'origine du thème de ce travail, pour l'intérêt qu'il a manifesté tout au long de mon séjour au C.T.I. et pour les nombreux conseils judicieux qu'il m'a prodigués.

Enfin, je tiens à exprimer ma sincère gratitude à Monsieur le Professeur Jean RAMAEKERS pour avoir accepté de diriger ce mémoire et pour m'avoir procuré un stage si passionnant.

AVERTISSEMENT AU LECTEUR

En fin de mémoire, le lecteur trouvera une liste des notations et des abréviations employées tout au long de ce travail.

Il pourra également consulter un glossaire dont l'objectif est de fixer la sémantique des termes et expressions techniques utilisées.

La lecture de ce travail ne requiert, en principe, aucune connaissance particulière. Néanmoins, toute personne désirant la précéder d'une étude des grands principes de la sécurité informatique, peut consulter <MEYER, 82> et <DAVIES, 84>.

TABLE DES MATIERES

1ère PARTIE : L'INTEGRITE DES FICHIERS : L'ANALYSE DU BESOIN ET DE DIVERSES SOLUTIONS

2ème PARTIE : UN MODELE DE FONCTION D'AUTHENTIFICATION

3ème PARTIE : DES ATTAQUES CONTRE LES FONCTIONS D'AUTHENTIFICATION

4ème PARTIE : L'ANALYSE DE QUELQUES FONCTIONS D'AUTHENTIFICATION

1. INTRODUCTION

2. QUELQUES FONCTIONS A ABANDONNER

   2.1. LE MODE CBC DEFINI POUR LE DES

   2.2. LE DSA

   2.3. UNE PREMIERE FONCTION DUE A MEYER ET MATYAS

   2.4. UNE FONCTION DUE A RABIN

   2.5. UNE FONCTION DUE A BITZER

   2.6. UNE FONCTION DUE A DAVIES ET PRICE

   2.7. UNE FONCTION DUE A DAVIES

   2.8. QUELQUES FONCTIONS UTILISANT DES OPERATIONS MODULO

   2.9. UNE FONCTION DUE A GILBERT, MAC WILLIAMS ET SLOANE

   2.10. UNE FONCTION DUE A WEGMAN ET CARTER

3. QUELQUES FONCTIONS INTERESSANTES

   3.1. UNE DEUXIEME FONCTION DUE A MEYER ET MATYAS

   3.2. UNE FONCTION DUE A MEYER ET SHILLING

   3.3. UNE FONCTION DUE A QUISQUATER

   3.4. LE " MESSAGE AUTHENTIFICATOR ALGORITHM " (MAA)

5ème PARTIE : UNE PROPOSITION DE SYSTEME D'AUTHENTIFICATION

TABLE DES FIGURES

INTRODUCTION

A l'heure actuelle, de plus en plus d'informations sont enregistrées et traitées par des ordinateurs. Les intérets en jeux sont considérables et c'est pourquoi il est primordial d'assurer un maximum de sécurité dans ce domaine.

Dans ce mémoire, nous traiterons d'un problème clé de la sécurité informatique : celui de l'**intégrité des fichiers.**

Ces derniers sont des éléments fondamentaux pour tout système informatique. Ils renferment en effet les informations permettant au système d'accomplir l'ensemble des fonctions pour lesquelles il est conçu. Une modification accidentelle ou frauduleuse de ces données peut entraîner des comportements inadmissibles du processus informatique.

Face à ce risque, il est possible d'adopter deux attitudes. La première consiste à enregistrer les informations dans un environnement destiné à empêcher les modifications. Nous montrerons que cette option n'est pas pratique et qu'elle possède des failles lorsque les manipulations ont un caractère frauduleux. Nous citerons notamment le cas de l'attaque par virus.

La seconde attitude est d'avoir recours à l'authentification : une technique permettant de vérifier qu'un ensemble donné d'informations est resté identique par rapport au moment où il pouvait être considéré comme correct. Les modifications ne sont donc pas empêchées, mais il devient possible de les détecter avant qu'elles ne provoquent des dégâts irréparables.

Le problème de l'authentification a été abondamment traité dans la littérature, et ce dans le cadre de la protection de l'intégrité des messages financiers. Nous nous proposons d'adapter les techniques développées à cette occasion à la protection des fichiers.

Pour cela, nous étudierons dans une première partie les différentes techniques permettant d'authentifier un fichier, et nous montrerons pourquoi **l'emploi de fonctions d'authentification** peut être considéré comme l'une des meilleures solutions.

Une fonction d'authentification calcule un code à partir du fichier à protéger. Si l'intégrité de ce fichier est mise en doute, un code est à nouveau générer et il est comparé avec le premier. En cas de différence, une manipulation est diagnostiquée.

Le principe est simple, mais son application nécessite la résolution de deux problèmes :

- Comment trouver une bonne fonction d'authentification?

- Comment l'intégrer au sein d'un système informatique?

Pour répondre à la première question, nous développerons un modèle qui mettra en évidence les éléments constitutifs et les qualités d'une bonne fonction d'authentification (partie deux). Ensuite, nous étudierons les attaques qui forment l'arsenal des fraudeurs. Cet examen nous donnera l'occasion de définir les critères de sécurité auxquels devront répondre les fonctions (partie trois).

Munis de ces informations, nous pourrons analyser quelques-unes des nombreuses fonctions d'authentification publiées dans la littérature. Quatre d'entre elles, particulièrement intéressantes, retiendrons notre attention (partie quatre).

Nous tâcherons de répondre à la deuxième question dans la cinquième partie de ce mémoire. Nous y ébaucherons un système composé d'éléments matériels et logiques concourant à mettre en oeuvre une fonction d'authentification. Pour ce faire, nous nous placerons dans le cadre d'un système informatique 'standard' que nous aurons préalablement défini. Une analyse critique nous permettra de déterminer les conditions d'applicabilité du système d'authentification proposé.

Première Partie

L'INTEGRITE DES FICHIERS :
L'ANALYSE DU BESOIN ET DE DIVERSES SOLUTIONS

1. INTRODUCTION

Un système informatique est un ensemble de matériels, de programmes, de données et, le cas échéant, de personnes. Il est organisé de façon à accomplir un ensemble de fonctions déterminées de traitement de l'information. Les programmes et les données y sont représentés sous forme binaire et contenus dans des fichiers.

Dans l'ensemble des mesures de sécurité visant à protéger un tel système, la protection des fichiers(*) occupe une place privilégiée. La principale raison en est que, contrairement aux composants matériels, les fichiers peuvent être très facilement modifiés. Des modifications sont possibles aussi bien durant la phase de développement que pendant celle de l'utilisation. Ceci permet de donner au système informatique de nouvelles propriétés. Toutefois, cette flexibilité est une menace pour la sécurité.

Il existe en effet de nombreux fichiers dont l'intégrité est essentielle pour le bon fonctionnement d'un système informatique. Ces fichiers doivent être protégés. Et s'il est impossible d'assurer que jamais ils ne seront accidentellement ou frauduleusement modifiés, il est nécessaire de mettre en oeuvre un mécanisme de détection des modifications.

Ce problème possède de multiples facettes, c'est pourquoi , dans le **chapitre 2**, nous commencerons par le spécifier. Nous y évoquons les principales techniques de protection des fichiers contre les modifications frauduleuses, puis nous montrerons qu'elles ne sont pas suffisantes dans de nombreuses situations, et qu'il existe donc un réel besoin pour une technique complémentaire : la vérification de l'intégrité des fichiers.

Cette technique , appelée l'authentification, sera définie et caractérisée dans le **chapitre 3**. Le **chapitre 4**, quant à lui, passera en revue les différentes méthodes d'authentification proposées dans la littérature. Au terme de cette analyse, nous retiendrons celle possédant le meilleur rapport sécurité / performances : l'utilisation de codes d'authentification.

(*) Par la suite, afin d'éviter des lourdeurs d'écriture, nous utiliserons souvent le terme **fichier** pour désigner le contenu d'un fichier

2. LA SPECIFICATION DU BESOIN

2.1. LA PROTECTION DES FICHIERS

Les fichiers sont des éléments fondamentaux dans tout système informatique. Ils renferment en effet toutes les informations permettant au système d'accomplir l'ensemble des fonctions pour lesquelles il est conçu. Cela explique pourquoi le problème de la protection des fichiers suscite tant d'intérêts. Celui-ci se décompose en une série de sous-problèmes dont les principaux consistent à empêcher les actions illégales

- de modification
- de création
- de destruction
- d'utilisation
- de copie
- de divulgation.

Tous ces dangers sont sérieux et ont de ce fait été l'objet de nombreuses études. Nous pensons toutefois que la modification accidentelle ou frauduleuse des fichiers est particulièrement grave. Elle peut en effet entraîner des comportements inadmissibles pour un système informatique:

"For example, data corruption in a hospital database could result in patients receiving the wrong medication. (...) corruption of military information could endanger national security. Erroneous information in (files of an airline company) could threaten passenger safety." <FERNANDEZ, 81, 2>.

Dans la suite de ce mémoire, nous nous attacherons à étudier le sous-problème de la protection d'un ensemble donné de fichiers contre toute modification non légale. La création et la destruction seront simplement évoquées. Les autres sous-problèmes, quant à eux, ne seront pas traités.

2.2. LES MODIFICATIONS DE FICHIERS

Il est courant de distinguer les fichiers de données - contenant des informations utilisables pour une application informatique quelconque - des fichiers exécutables (ou programmes) - contenant des instructions relatives à la manipulation d'informations. Cette dernière définition sous-entend la possibilité de modifier des fichiers grâce à l'exécution d'instructions contenues dans d'autres fichiers. Mais avant d'aborder ce cas, examinons celui d'une modification accidentelle.

2.2.1. LES MODIFICATIONS ACCIDENTELLES

Il est possible qu'un fichier soit altéré par des phénomènes physiques au cours de son existance. Ces phénomènes peuvent être par exemple le bruit présent sur une ligne de transmission par laquelle il transite, ou une source électromagnétique proche de son support de stockage. Nous qualifions ce type d'altération d'accidentel.

Par la suite, nous ne traiterons plus de ces modifications, et cela pour les deux raisons suivantes: elles ont déjà été l'objet de très nombreuses études (cfr notamment <MACWILLIAMS, 77>), mais surtout, il est possible de les considérer comme un sous-ensemble des modifications frauduleuses que nous allons introduire dans le paragraphe ci-dessous.

2.2.2. LES MODIFICATIONS AUTORISEES ET FRAUDULEUSES

Un fichier peut être modifié sciemment. Dans ce cas, le changement résulte de l'exécution par un processeur d'une suite d'instructions contenues dans un programme.

Nous considérons qu'une telle modification est autorisée (ou qu'elle est légale) si elle émane d'une source ayant le droit de l'effectuer. Si elle ne l'est pas, nous la qualifions de frauduleuse, et ceci même si elle n'entre pas dans le cadre d'une fraude proprement dite. Il peut s'agir par exemple d'une maladresse d'un utilisateur ou d'un disfonctionnement du mécanisme gérant les accès aux fichiers.

Dans la suite de ce mémoire, nous étudierons ces modifications frauduleuses (*). Nous ne nous intéresserons que très peu à la façon de les réaliser, si ce n'est en évoquant dans le paragraphe suivant le cas particulier de la modification par un virus. Peu nous importe également de savoir qui les perpétue. Nous prenons le parti de les considérer comme une réalité dont il faut tenir compte, et notre but est de définir une technique capable de protéger les fichiers contre leurs effets.

(*) Par la suite, nous omettrons souvent le qualificatif 'frauduleuses' pour parler simplement de 'modifications'. Comme synonymes, nous emploierons 'manipulations' ou 'altérations'.

2.2.3. LES MODIFICATIONS DUES AUX 'COMPUTER VIRUSES'

Nous voulons montrer dans ce paragraphe la nécessité de prendre des mesures pour lutter contre les modifications frauduleuses de fichier. Le danger est en effet réel et peut prendre différentes formes :

- Des informations peuvent être interceptées alors qu'elles transitent sur une ligne de communication, être modifiées et ensuite réinjectées <DAVIES, 84, 7>.

- Des informations peuvent être directement modifiées sur leur support de stockage en courcircuitant les programmes d'applications qui garantiraient la validité de l'accès.

Nous avons choisi d'illustrer le risque constitué par les modifications de fichiers en évoquant une menace particulièrement grave pour la sécurité informatique : les 'virus'('computer viruses'). Les commentaires ci-dessous sont à attribuer, sauf mention contraire, à un grand spécialiste des virus, le Dr Frederick COHEN <COHEN, 87>.

"We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected may also act as a virus".

La meilleure façon d'expliquer les caractéristiques d'un virus est sans doute d'en donner un exemple simple. Nous utiliserons pour cela un langage défini en annexe 1.

La figure 1.1. décrit un virus qui cherche un programme non infecté (ne commençant pas par '1234567') pour y inclure une copie de lui-même. Le programme ainsi modifié devient infecté. Le virus vérifie alors la valeur d'une condition de déclenchement de dégats. S'il s'agit de la valeur vraie, le virus exécute une série d'instructions correspondant aux dégats voulus. Finalement, le virus exécute la suite du programme au début duquel il est placé.

Quand un utilisateur exécute le fichier infecté, il commence sans le savoir par exécuter ce virus, ce qui a pour effet d'étendre l'infection.

FIGURE 1.1 : Un exemple de virus

```
program virus :=
   {1234567;

    subroutine Infection-programme :=
       {boucle : fichier = Choix-aléatoire-de-programme;
                 if Première-ligne-du-fichier = 1234567
                    then goto boucle;
        Ajouter-virus-au-début-du-fichier;
       }

    subroutine Faire-des-dégats :=
       {...}

    subroutine Déclanchement-dégats :=
       {if année>1988 then return (vrai)
                      else return (faux)
       }

    main-program :=
       {Infection-programme;
          if Déclanchement-dégats
               then Faire-des-dégats;
          goto suite;
       }

    suite :
   }
```

Le virus peut servir à véhiculer n'importe quelle application. Il est donc clair que dans les mains de personnes mal intentionnées, il constitue une arme redoutable. Ce danger est à prendre d'autant plus au sérieux que :

"None of the published proposed systems defines or implements a policy which could stop a virus.(...) The only provably 'safe' polycy as of this time is isolationism".

Pour compléter cette inquiétante description, notons encore que :

"Viral attacks appear to be easy to develop in a very short time, can be designed to leave few if any traces in most current systems, are effective against modern security policies for multilevel usage, and require only minimal expertise to implement"

Quelques expériences récentes concernant une entreprise multinationale (<HIGHLAND, 87>, et même la très sérieuse NASA (<A.P., 88>, <FERREIRA, NP>) sont là pour nous rappeler que le virus doit aujourd'hui être considéré comme une réalité avec laquelle il faut compter.

Nous ferons par la suite de nombreuses références à cet exemple de la modification de fichiers par virus afin d'illustrer les divers aspects de la protection de l'intégrité des données.

2.3. LA PROTECTION DES FICHIERS CONTRE LES MODIFICATIONS

Les moyens de protection des fichiers contre les modifications peuvent être classés en deux groupes.

Le premier est constitué des protections 'logiques' qui consistent à exécuter des instructions refusant ou acceptant le droit de manipuler le fichier en fonction du contexte de l'exécution et de l'identité de l'entité responsable de la demande de modification.

Le second comprend les protections 'physiques' où le fichier est placé sur un support de stockage non altérable.

2.3.1. LES PROTECTIONS LOGIQUES

Il existe, au niveau du système d'exploitation de la plupart des ordinateurs, des mécanismes mettant en oeuvre les protections nécéssaires pour qu'il soit impossible de modifier sans autorisation des informations dans le système d'exploitation lui-même ou dans les fichiers des utilisateurs. Pour rester simple, on peut considérer que ces mécanismes consistent à accepter ou à refuser le droit d'exécution des instructions qui auraient pour conséquences de modifier les fichiers en question. Ce choix est guidé par le contexte de l'exécution et par un jeu de privilèges basé sur l'identification de l'entité demandant la modification <MEINADIER, 84>.

Le même type de protection est présent dans la plupart des Systèmes de Gestion de Base de Données ou SGBD <FERNANDEZ, 81>.

Les protections logiques ont toutes un inconvénient de taille: elles ne peuvent être efficaces que lorsqu'elles sont actives. Si un fichier est stocké sur un disque qui est 'emprunté ' par un fraudeur pour le modifier sur son propre système, l'attaque ne peut être contrée. La même situation existe lorsque le fichier est intercepté durant un transfert électronique, qu'il est modifié et est ensuite réinjecté sur le moyen de transport.

Cette faille peut-être également illustrée par l'exemple du virus. L'expérience a montré qu'une fois le virus introduit dans un système, il parvient très rapidement à infecter des fichiers à

tous les niveaux de sécurité, outrepassant ainsi les mécanismes de protection des accès gérés par le système d'exploitation (<COHEN, 87>).

Une dernière critique de ce type de protection tient au fait que puisque les protections logiques sont elles mêmes contenues dans des fichiers, elles peuvent subir le même sort que les objets qu'elles sont sensées protéger.

2.3.2. LES PROTECTIONS PHYSIQUES

Il existe sur le marché toute une gamme de supports qui permettent de protéger physiquement un fichier contre les modifications: les ROM, les CD-ROM, les disques optiques numériques ineffaçables, ...

Nous allons rapidement en donner les grandes caractéristiques dont la principale est évidemment d'être théoriquement non modifiable.

- **La ROM** (Read Only Memory) est utilisée par les fabriquants de composants informatiques pour stocker des programmes figés. Elles sont programmées de façon irréversibles par fusion sélective de liaisons <MENADIER, 84, 356>.

- **Le CD-ROM** (Compact Disc - Read Only Memory) est le frère jumeau du disque compact audio. Son gros inconvénient est d'avoir un mode d'écriture par pressage d'un master par le constructeur. Il est donc extrêmement cher s'il n'est pas produit en de multiples copies. Il semble être intéressant pour la distribution en de multiples exemplaires d'une même base de données <ANONYME, 86>.

- **Le disque optique numérique ineffaçable** offre lui l'avantage de permettre directement à l'utilisateur d'écrire une fois pour toutes sur un disque vierge à l'origine. Deux applications intéressantes semblent être le stockage de fichiers qui ne subissent que rarement des mises-à-jours, et l'archivage de toute information digitale <ANONYME, 86>.

Une première limite à ces moyens de protection physique est qu'il n'est pas imaginable de placer tous les fichiers d'un système informatique sur de tels supports. Cela signifierait en effet qu'aucune modification ne pourrait être apportée au système sans remplacer un moyen de stockage. Dans beaucoup d'applications, le taux de mise-à-jour des fichiers est souvent tel que cette solution est inapplicable.

De plus, il serait illusoire de croire que ces

supports résolvent tous les problèmes liés à la sécurité des fichiers. Une ROM peut être lue et son contenu analysé. Il est donc possible d'en réaliser une version frauduleuse qui pourrait remplacer l'ancienne <DAVIES, 84, 6>. Pour limiter ce risque, des procédures de vérification du matériel sont nécéssaires. Une autre attaque consiste à intercepter et modifier un fichier au moment ou il est transféré d'un support vers un autre.

Il est bien évident que les supports de stockage non effaçables sont très utiles. Ils servent de base à bon nombre de techniques relatives à la sécurité. Ils ne constituent toutefois pas à eux seuls une solution complète au problème de la modification des fichiers.

2.3.3. LE BESOIN RELATIF A LA DETECTION DES MODIFICATIONS

En conclusion des deux sections précédentes, nous pouvons affirmer qu'il n'est pas possible de faire aveuglément confiance aux techniques de protection pour empêcher toute modification des fichiers. Il existe donc un besoin pour une technique de détection des modifications qui auraient échappé aux moyens classiques de protection.

Ce besoin est exprimé par de nombreux auteurs, et notamment par POZZO et GRAY (<POZZO, 86>). Ceux-ci disent préférer un mécanisme détectant les modifications à un autre qui aurait pour mission de les empêcher. Ces chercheurs justifient leur choix de la manière suivante: dans l'approche 'detection', le mécanisme de protection est lié à l'objet à protéger (le fichier), et non au système d'exploitation ou au moyen de stockage comme dans l'approche 'd'empêchement'. La détection offre donc l'avantage de rester opérationnelle même en dehors du champ d'action d'un système particulier. C'est le cas par exemple lorsqu'un fichier est transféré d'un site à un autre. Les modifications éventuelles peuvent toujours être détectées au site de destination.

Ce choix est confirmé si l'on se réfère de nouveau au cas de l'attaque par virus. Si, comme nous l'avons déjà fait remarquer, il est très difficile d'empêcher un virus de contaminer un fichier, il est par contre possible de détecter cette infection puisqu'elle provoque une modification du fichier en question. Si toute exécution est précédée d'une vérification de non modification, les programmes infectés pourront être décelés et leur exécution refusée. La propagation du virus est ainsi arrêtée. Cette technique de lutte contre les virus est proposée dans <POZZO, 86>.

2.4. LA DETECTION DES MODIFICATIONS DE FICHIERS

Nous sommes arrivés à la conclusion que vérifier l'intégrité d'un fichier est un réel besoin pour assurer la protection d'un système informatique. Avant d'envisager la manière de répondre à ce besoin, il est nécéssaire d'en éclaircir plusieurs aspects importants:

- quels fichiers vérifier?
- à partir de quand les vérifier?
- et dans quelles circonstances le faire?

Ceci va nous donner l'occasion de poser un certain nombre d'hypothèses très utiles pour la suite de ce travail.

2.4.1. QUELS FICHIERS VERIFIER ?

La réponse la plus logique à la question "quels fichiers faut-il vérifier?" semble être "ceux qui en valent la peine". Derrière cette trivialité se cache l'éternel problème du rapport coûts-pertes : si le prix à payer pour assurer l'intégrité d'un fichier (investissements, baisse des performances, ...) est inférieur aux pertes qui pourraient être occasionnées par une modification de celui-ci, alors cela vaut la peine de le protéger. Nous développons cette problématique dans le chapitre 5 de la troisième partie.

Dans <POZZO, 86>, les objets à protéger sont restreints aux fichiers exécutables parce que :

"Modifications to executables are much less noticeable than those made to text or data files, and often go undetected. Such modifications often cause unexpected, unauthorized, or malicious side-effects in computer system"

Ce choix est dicté par une volonté clairement exprimée de lutter contre la propagation des virus qui se fait, rappelons le, par infection des programmes.

Nous estimons toutefois qu'il existe de nombreuses applications où les données sont tout aussi précieuses que les programmes qui les manipulent. C'est notamment le cas dans le domaine des bases de données.

Mais en fait, il est impossible de spécifier arbitrairement quels sont les fichiers dignes de protection. Nous conseillons un audit du système informatique qui se chargera de répondre à cette question en toute connaissance de cause <FERNANDEZ, 81, 149-177> .

N'ayant posé aucune restriction particulière sur la nature des fichiers à traiter, nous nous réferrons simplement à une hypothèse minimale couramment adoptée dans la littérature (cfr notamment <AKL, 84> et <WINTERNITZ, 84a>), à savoir qu'un fichier à protéger est une suite d'un nombre fini de b bits (b étant un entier positif quelconque).

2.4.2. A PARTIR DE QUAND VERIFIER LES FICHIERS ?

Déterminer l'instant à partir duquel un fichier doit être vérifié est relativement complexe. Il semble pourtant raisonnable de choisir le moment de la création. C'est précisément ici que les choses se compliquent: un fichier passe très souvent par une série d'étapes plus ou moins longues avant d'être effectivement utilisé. Pendant tout ce temps, des modifications frauduleuses sont possibles.

Nous ne nous étendrons pas sur cet aspect des choses et nous ferons simplement l'hypothèse que la phase de développement d'un fichier se déroule dans un univers protégé de toute manipulation. C'est donc un fichier considéré comme correct par ses concepteurs (ou par un organisme d'audit) qui est mis à la disposition du système chargé de le protéger. Ce fichier est la base de la technique de vérification. Dans notre terminologie, il porte le nom de **fichier authentique.**

2.4.3. DANS QUELLES CIRCONSTANCES VERIFIER LES FICHIERS ?

De façon schématique, nous pouvons considérer qu'un fichier subit trois types d'opérations tout au long de son existance: il est bien sûr utilisé par un processeur, mais est égalemment stocké et transmis.

- Examinons tout d'abord le cas du stockage. Un fichier authentique est placé sur un support accessible d'une manière ou d'une autre en écriture. Nous avons fait l'hypothèse que, quelles que soient les techniques de protection utilisées, modifier ce fichier est toujours possible. Il est donc nécessaire de le vérifier dès qu'il doit être utilisé pour une opération nécéssitant son intégrité.

- Passons à la transmission. Des trois opérations, c'est certainement celle qui encourt les plus grands risques. Que cette transmission s'effectue localement entre un terminal intelligent possédant des capacités de stockage et un ordinateur, ou entre deux ordinateurs situés sur des continents

différents, le risque est le même: un ennemi a la possibilité de piéger le support de communication (cable, ondes, commutateurs, multiplexeurs, ...). Le fichier peut alors être intercepté, modifié et replacé sur le réseau de communication.

Ce danger pour l'intégrité des fichiers risque bien de suivre la même courbe croissante que celle des développements en matière de télécommunications et de systèmes informatiques répartis <CARLSON, 86>.

- Nous ne traiterons pas le cas de la modification frauduleuse d'un fichier pendant son traitement par un processeur. Nous faisons l'hypothèse que le processeur est exempt de tout soupçon, ce que nous exprimons par le 'processeur sûr' (secure processor), une expression empruntée à Carl MEYER <MEYER, 88>. Rien ne peut en effet altérer le fichier entre sa vérification et la fin de son utilisation, sinon c'est durant ce laps de temps que portera l'attaque. Nous reviendrons plus en détails sur ce point dans la dernière partie de ce travail.

Dans la suite de ce mémoire, nous allons nous attacher à découvrir le meilleur moyen de mettre en oeuvre l'énoncé suivant:

- Un fichier authentique doit être protégé car il va évoluer dans un environnement non sûr où il peut subir des manipulations.

- Une copie de ce fichier est placée dans un milieu inviolable (un coffre-fort par exemple) préalablement à l'exposition de la version originale.

- A un moment donné, il est nécessaire de s'assurer que le fichier n'a pas subi de modifications. Dans le cas contraire, la copie est récupérée.

Cette spécification peut sembler vague, mais nous l'avons voulu ainsi. En restant général (et plus particulièrement en ne faisant pas la distinction entre stockage et transmission), nous espérons couvrir le plus grand nombre de situations possibles.

Cette généralité nous sera d'un précieux intérêt lorsque, dans la cinquième partie, nous donnerons l'ébauche d'une solution standard au problème de la vérification de l'intégrité d'un fichier.

3. L'AUTHENTIFICATION ET LES SYSTEMES D'AUTHENTIFICATION

3.1. DEFINITIONS

L'authentification d'un fichier est une technique qui consiste à appliquer à ce fichier un traitement dont le résultat permet de déterminer s'il a subi des modifications par rapport au moment où il était considéré comme authentique. Nous disons que cette technique vérifie l'intégrité ou l'authenticité du fichier.

Nous appelons **système d'authentification** l'ensemble des matériaux, programmes et données qui doivent être mis en oeuvre pour assurer le bon fonctionnement de cette technique.

3.2. L'AUTHENTIFICATION DES MESSAGES

La technique de l'authentification est actuellement appliquée presque exclusivement dans le domaine de la protection des messages transmis par l'intermédiaire de réseaux électroniques de communication. Les utilisations les plus courantes portent sur la protection des messages financiers pour laquelle il existe plusieurs standards (ou projets) de normalisation <ANSI, 82>, <ISO, 86a>, <ISO, 86b>.

3.3. L'AUTHENTIFICATION DES FICHIERS

L'authentification des fichiers pourrait faire l'objet d'une simple généralisation de celle utilisée pour les messages. Elle possède pourtant des caractéristiques qui la différencient nettement et parmi lesquelles nous pouvons citer:

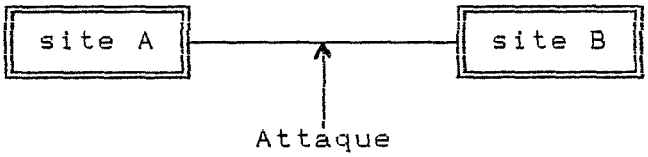
- La taille: un fichier peut atteindre une taille se chiffrant en dizaine de Mo tandis qu'un message, au sens financier du terme, dépasse rarement quelques Ko. L'authentification des fichiers doit donc posséder des performances très élevées pour être utilisable en pratique. Ces performances sont à évaluer en fonction du contexte, mais il nous semble raisonnable de prendre comme critère le fait que l'authentification d'un fichier de 1 Mo (Mega octets) prenne moins d'une dizaine de secondes sur un processeur de puissance moyenne.

- Le caractère durable: l'attaque contre un message financier doit être exécutée très rapidement, c'est-à-dire durant le transfert d'un endroit à un autre. Elle pose souvent d'énormes difficultés techniques au fraudeur <FERREIRA, NP>. Par contre, en raison du fait

qu'un fichier possède généralement une durée de vie plus longue, l' ennemi possède plus de temps pour l'altérer. L'authentification d'un fichier doit donc être sensiblement plus efficace que celle des messages.

- **La difficulté de conserver un secret:** dans le cadre de l'authentification de messages, il est courant de supposer que l'attaque provient uniquement de l'extérieur du système d'authentification. Autrement dit, on présume qu'elle se porte sur la ligne de communication reliant les deux sites échangeant les messages, ce qu'illustre la figure 1.2.

FIGURE 1.2. : Attaque sur une ligne de communication reliant deux sites



Il est donc facile de prendre comme acquis le fait que tout ce qui concerne le système d'authentification est gardé secret entre les deux sites communiquant , comme c'est le cas dans <ANSI, 82> et <ISO, 86b>.

En matière d'authentification de fichiers, nous n'avons pas fait de restrictions sur l'identité de l'ennemi et la nature de l'attaque. Il peut donc très bien y avoir des attaques 'de l'intérieur'.

De plus, les installations informatiques actuelles sont souvent faites de plusieurs ordinateurs de différentes puissances interconnectés et accessibles aussi bien localement qu'à distance. Les fichiers sont donc situés à de nombreux endroits, ce qui nécessite une technique d'authentification décentralisée <CARLSON, 86>.

La conjugaison de ces deux éléments doit nous amener à être beaucoup plus sévères dans nos critères de sécurité en matière de protection de fichiers. Il n'est en effet plus question de baser la sécurité de l'authentification sur la confidentialité d'éléments répartis sur plusieurs sites et donc d'autant plus susceptibles d'être découverts par des attaques de l'intérieur. C'est pourquoi, nous garderons toujours à l'esprit l'affirmation suivante:

Moins il y a d'éléments secrets nécessaires à la garantie de la sécurité de notre système d'authentification et plus il pourra être considéré comme solide.

A titre de conclusion, nous spécifions le besoin relatif à l'authentification des fichiers comme suit:

Il nous faut un système assurant **rapidement** la vérification de l'authenticité d'un fichier. Ce système doit être **sûr** sans pour autant se baser sur le fait que les divers éléments le constituant sont inconnus d'un éventuel ennemi.

Par système 'sûr', nous entendons qu'un ennemi ne peut parvenir à modifier les fichiers protégés sans que cela ne soit détecté.

4. LES DIFFERENTES METHODES D'AUTHENTIFICATION

Nous cherchons un système qui nous permette de vérifier si un fichier a été modifié. La base de ce système doit être une fonction qui, appliquée au fichier, fournisse un résultat tel qu'il soit possible de déduire à coup sûr si une modification a eu lieu.

Plusieurs méthodes peuvent être envisagées pour mettre ceci en oeuvre. Nous allons passer les principales en revue.

4.1. LES CODES DETECTEURS D'ERREURS

4.1.1. LE PRINCIPE

Il est très fréquent que des phénomènes physiques provoquent une modification des données transmises sur une ligne de communication ou stockées sur un support magnétique. Il est vital de détecter ces accidents. Ceci se fait en utilisant des Codes Détecteurs d'Erreurs ou CDE.

Le principe de ces codes est simple. L'ensemble des données est divisé en ' blocs ' de taille fixe pour chacun desquels est calculée une valeur relativement plus courte : le CDE. Cette valeur est accolée au bloc avant l'envoi ou le stockage. Par la suite, il est facile de vérifier si une modification a eu lieu en recalculant le code et en le comparant avec celui associé au bloc considéré :

- s'ils sont différents, nous pouvons être certains qu'une modification a eu lieu (bien qu'il soit parfois impossible de dire si elle porte sur le bloc, sur le code ou sur les deux à la fois)

- s'ils sont égaux, nous pouvons être raisonnablement sûr que rien n'a altéré le bloc.

4.1.2. UNE EVALUATION

Les vérifications décrites ci-dessus assurent une protection efficace des fichiers contre les modifications accidentelles (ou erreurs). Par contre, ce n'est pas le cas en ce qui concerne les modifications frauduleuses parce que la technique de calcul d'un CDE et les éventuels paramètres sont en général de connaissance publique. Un ennemi peut donc facilement modifier les données et faire les corrections nécessaires sur les CDE. Une telle

fraude ne sera pas détectée <DAVIES, 84, 123>.

Il est possible de faire objection à cette remarque en disant qu'il suffit de garder secret la méthode de calcul. A cela, nous répondons en trois points :

- Dans le cadre actuel de l'informatique, il est essentiel d'avoir un minimum de standardisation. Si chacun garde secret sa technique de calcul du CDE, aucune communication n'est possible.

- Les CDE sont conçus pour protéger les données contre les erreurs. Sous cette hypothèse, les CDE peuvent être efficaces tout en étant très rapides. Malheureusement, cette rapidité est le fruit d'une simplicité de conception qui ne leur permet pas d'espérer résister à une analyse d'un spécialiste en fraudes cryptologiques.

- Si les CDE sont conçus pour protéger les données contre les fraudes, ils devront obligatoirement être calculés de façon beaucoup plus complexe, et donc plus lente. Ceci est inadmissible quand on sait que dans la plupart des cas, l'utilisateur préfère la rapidité avec une sécurité correcte que la lenteur avec une sécurité parfaite.

En conclusion, nous affirmons qu'il est nécessaire de séparer le traitement des erreurs de celui des fraudes. Le premier est standardisé, rapide et obligatoire. Le second peut être 'particularisé '. Il est forcément plus lent et optionnel. Selon cette optique, un fichier n'est soumis à l'authentification que s'il est exempt d'erreurs.

Nous ne reviendrons plus sur le traitement des erreurs, c'est pourquoi nous attirons maintenant l'attention sur le fait que celui-ci est fondamental dans le cadre de l'authentification des fichiers.

Imaginons en effet que, tous les Mo, une erreur échappe à la détection d'un CDE. Dans ce cas, un fichier ayant au moins cette taille **NE POURRA JAMAIS ETRE AUTHENTIFIE CORRECTEMENT** puisque, fraude ou pas, l'authentification diagnostiquera une modification.

Nous pouvons en déduire qu'un système d'authentification ne peut fonctionner que s'il repose sur un système de détection (et correction) des erreurs proches de la perfection.

4.2. LE CHIFFREMENT SYMETRIQUE

Il n'entre pas dans le cadre de ce mémoire de décrire de façon précise les techniques et propriétés du chiffrement symétrique. Une brève description en est cependant donnée. Pour de plus amples renseignements, nous conseillons de consulter notamment <DAVIES, 84>, et <MEYER, 82>.

Le DES est l'algorithme de chiffrement le plus connu. Il est présenté à titre d'exemple.

4.2.1. LE PRINCIPE

Le chiffrement symétrique est une fonction à deux variables :

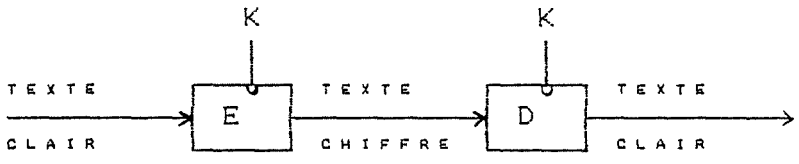
- un morceau de fichier, appelé le 'plaintext' ou 'texte clair',
- et une clé secrète.

Le résultat de la fonction est un autre morceau de fichier, appelé le 'ciphertext' ou 'texte chiffré'. Ce résultat est tel que, si la clé est inconnue d'un ennemi, il lui est extrêmement difficile de déduire le texte clair du texte chiffré. Par contre, grâce à la clé, cette opération est très simple. Elle s'effectue grâce à la fonction inverse du chiffrement : le déchiffrement. Celle-ci demande comme paramètres:

- le texte chiffré,
- et une clé secrète.

Ces opérations sont représentées sur la figure 1.3.

FIGURE 1.3. : chiffrement (E) et déchiffrement (D)



La clé de déchiffrement est identique à (ou peut être déduite facilement de) celle utilisée pour le chiffrement. Cette dualité est à l'origine du terme 'symétrique' qualifiant la technique présentement décrite. Il en existe une autre dite 'asymétrique' où la clé de déchiffrement ne peut être déduite de

celle employée pour le chiffrement <DIFFIE, 76>. Cette technique est à la base de la signature numérique qui est présentée au paragraphe suivant.

De façon plus concise, nous employons pour le chiffrement symétrique (*) les notations suivantes :

si x est le texte clair, alors

$$y = E_K(x)$$

est le texte chiffré grâce à la clé K. La fonction inverse

$$x = D_K(y)$$

exprime quant à elle le déchiffrement, grâce à la même clé K, du texte chiffré y pour retrouver la version en clair x.

Le chiffrement d'un fichier offre une certaine forme d'authentification <DAVIES, 84, 134>, <WEGMAN, 81>. Ceci se comprend plus facilement dans le cas d'un fichier de texte.

Ce texte est chiffré et la clé est gardée secrète. Quand plus tard il est déchiffré, une vérification s'assure que le contenu 'a du sens' (mots appartenant à un certain langage, phrases syntaxiquement correctes, sémantique logique, ...).

Si c'est le cas, nous pouvons être relativement sûrs qu'aucun changement n'a été opéré. Ceci s'explique par le fait que sans la clé, une personne malveillante ne peut prévoir quels effets une modification de la version chiffrée produit sur le texte clair. Au moindre bit changé, le déchiffrement résultera en un texte comprenant des passages 'sans aucun sens' comme par exemple "cher ami, je ¥ø├ ▓Ø T-┐à et ...".

4.2.2. UN EXEMPLE : LE DES

Le "Data Encryption Standard", mieux connu sous l'abréviation de DES, est un algorithme de chiffrement qui fut publié comme standard en 1977 <NBS, 77> et qui est maintenant largement employé pour de nombreuses applications cryptologiques. Bien que longtemps contesté <HELLMAN, 79>, le DES reste l'algorithme de chiffrement offrant à l'heure actuelle le meilleur

---

(*) Par la suite, le terme 'chiffrement' fait référence au 'chiffrement symétrique'

produit sécurité * vitesse <COPPERSMITH, 87>.

Nous n'allons pas nous étendre sur sa structure interne. Nous signalons simplement qu'une clé de 56 bits est utilisée pour transformer un texte clair de 64 bits en un texte chiffré de 64 bits.

Le DES est utilisé à titre d'exemple à de nombreuses reprises dans la suite de ce mémoire, c'est pourquoi nous invitons les 'non-initiés' à parcourir l'annexe 2 qui en donne une description plus détaillée. Pour de plus amples renseignements, nous conseillons <MEYER, 82> et <NBS, 81>.

4.2.3. UNE EVALUATION

Pour l'authentification de fichiers, nous ne retiendrons pas la technique du chiffrement symétrique, et ce pour les trois raisons suivantes :

- **La première** vient de la nécessité d'avoir la notion de 'sens'. Quelque chose 'a du sens' si elle contient suffisamment de redondance <DAVIES, 84, 134>. Pour un texte en français, cette redondance est aisément vérifiée avec un dictionnaire et une grammaire. Par contre,

  "*la nature même des données informatiques est à ce point diversifiée que l'on ne peut les supposer systématiquement émises par une source redondante, et ce même quand elles le sont*" <CAMPANA, 88>.

  Une solution à ce problème consiste à créer artificiellement de la redondance. Pour cela, on calcule, à partir d'un fichier authentique, une valeur appelée Code Détecteur de Manipulation ou CDM. Un CDM joue en quelque sorte pour la détection des manipulations frauduleuses le rôle tenu par les CDE dans la détection des modifications accidentelles. Ce code est ajouté au fichier et le tout est chiffré. Après déchiffrement, le CDM est recalculé pour s'assurer qu'aucune modification n'a été commise. Cette technique a été étudiée par JUENEMAN, MATYAS et MEYER dans <JUENEMAN, 84, 85, 87>. Nous y reviendrons dans la partie suivante, mais dans un autre contexte que celui du chiffrement.

- **La deuxième raison** tient au fait que si la clé est découverte par un ennemi, rien ne peut empêcher celui-ci de frauder. Il lui suffit de déchiffrer le fichier, de le modifier sans contrainte (si ce n'est celle consistant à corriger les éventuels CDM) et enfin de le

rechiffrer. Or, cette clé est nécessaire pour chaque authentification. Elle est donc largement utilisée et le risque d'indiscrétion est grand.

- **La troisième raison** est qu'il est intéressant dans beaucoup d'applications de dissocier l'authentification du chiffrement. Cela peut être le cas par exemple pour un fichier de données que l'on pourrait lire à volonté, mais qui ne demanderait à être authentifié qu'au moment où il est utilisé pour un traitement important.

4.3. LA SIGNATURE NUMERIQUE

4.3.1. LE PRINCIPE

La signature numérique (ou digitale) fut définie par DIFFIE et HELLMAN dans <DIFFIE, 76>. Elle a contribué à révolutionner les concepts relatifs à la protection des messages. Elle consiste en une série de bits qui est accolée à la suite du message. Le signataire est le seul à connaître la fonction utilisée pour générer cette suite de bits à partir du texte à protéger. La fonction de vérification est quant à elle rendue publique. Elle permet à n'importe qui de vérifier que la signature est valide pour le message considéré <WEGMAN, 81>.

La caractéristique cruciale d'un tel système est que sans la fonction 'signante', il est impossible de déterminer la signature de n'importe quel autre message. Le recepteur d'un texte signé peut donc non seulement se convainvre, mais également prouver que le texte est authentique et qu'il provient bien du signataire.

Nous pouvons décrire de façon plus formelle le processus d'authentification de la manière suivante:

- Le message M est signé par une fonction S de façon à produire la signature Ms qui accompagnera M:

$$Ms = S(M).$$

- Pour authentifier M, il nous suffit d'appliquer à Ms la fonction de vérification V:

$$M' = V(Ms),$$

et de comparer M et M'. S'ils sont égaux, nous sommes sûrs que M est bien le message signé par le possesseur de S, car il est le seul capable

d'avoir généré Ms.

Une signature numérique peut être générée à l'aide d'un algorithme de chiffrement symétrique tel le DES <MATYAS, 83>, mais une implémentation utilisant un algorithme asymétrique est beaucoup plus sûre <MEYER, 88>. Plusieurs méthodes sont présentes dans la littérature: <DIFFIE, 76>, <RABIN, 78>, <ONG, 84>... Au paragraphe suivant, nous présentons la technique qui est sans doute la plus connue : celle basée sur le RSA de RIVEST, SHAMIR et ADLEMAN <RIVEST, 78>.

4.3.2. UN EXEMPLE : LA SIGNATURE UTILISANT LE RSA

Le RSA est un système de chiffrement asymétrique. Tout comme pour le DES, aucune faiblesse sérieuse n'a été découverte depuis sa publication en 1978 <PAILLES, 86>. Il se base sur la théorie de l'arithmétique finie <DAVIES, 84, 256-262> où il est possible de trouver de très grands entiers e, d et n tels que:

$$(x^e)^d \bmod n = (x^d)^e \bmod n = x.$$

La sécurité du RSA réside dans le fait qu'étant donnés n et e, il n'existe aucun algorithme efficace qui permette de trouver d. A titre d'exemple, pour n ayant une taille de 512 bits, ce travail nécessite environ 300 000 ans sur un CRAY I <FERREIRA, NP>.

La clé publique est donc la paire (n, e) tandis que la clé secrète est d. Pour signer un fichier F, le possesseur de la clé secrète le convertit en un entier x inférieur à n (ou en plusieurs entiers si le fichier est trop long) et il calcule :

$$S(x) = x^d \bmod n.$$

Quiconque connaissant la clé publique peut vérifier la signature en calculant :

$$V(S(x)) = (x^d)^e \bmod n = x.$$

4.3.3. UNE EVALUATION

Bien que définie pour la protection des messages, la signature numérique est applicable à n'importe quel fichier. Elle pourrait donc s'avérer extrêmement intéressante pour leur protection. Elle possède en effet une qualité qui est fondamentale pour l'authentification de fichiers telle que nous l'avons définie: elle ne nécéssite qu'un minimum de secret dans le chef de la clé de signature. Pour ce

qui est de l'authentification proprement dite, seuls des éléments publics sont requis.

Malheureusement, elle pose, du moins sous la forme décrite ci-dessus, deux problèmes majeurs:

- **Le premier** provient du fait que la longueur de la signature numérique est du même ordre de grandeur que celle du fichier. Doubler le volume total du stockage est inadmissible dans beaucoup de configurations informatiques.

- **Le second problème** est encore plus grave: les systèmes de signature numérique sûrs sont extrêmement lents. Le RSA, par exemple, utilise en temps de calcul pour signer un message de deux à trois ordres de magnétude en plus que le DES pour chiffrer ce même message <WINTERNITZ, 84a>. Des systèmes plus rapides comme par exemple celui du à OKAMOTO et à SHIRAISHI <OKAMOTO, 85> se sont révélés peu sûrs <BRICKELL, 86>.

Nous n'allons pas abandonner la signature numérique pour autant. Elle redevient en effet très intéressante si elle est conjuguée à une technique connue sous le nom de condensation à sens unique <DIFFIE, 76>. Cette technique est introduite au paragraphe suivant.

4.4. L'UTILISATION DE CODES D'AUTHENTIFICATION

4.4.1. LES FONCTIONS DE CONDENSATION A SENS UNIQUE

La notion de code d'authentification est directement liée à celle de 'fonction de condensation à sens unique'.

Une fonction est dite de 'condensation' si elle associe à un fichier de b bits (b étant arbitraire) une et une seule valeur de m bits (m étant fixe et significativement plus petit que b) <AKL, 84>.

Si H est une fonction de condensation, si F est un fichier et si v est la valeur associée à F par H (ce que nous notons H(F) = v), v est appelée 'valeur de condensation' ou 'condensat' de F par H.

Une fonction H est dite 'à sens unique' si

- pour n'importe quel argument F appartenant au domaine de H, il est facile de calculer la valeur correspondante v = H(F),

- mais, par contre, "*given H and F, it is*

computationally infeasible to find an F' ≠ F such that H(F') = H(F)" <AKL, 84>.

Ici, l'expression 'computationally infeasible' exprime le fait qu'un ennemi possédant du temps et des moyens de calcul illimités parviendrait peut-être à trouver le F' en question. Mais dans la réalité où ses ressources sont limitées, cela lui est impossible <SIMMONS, 88>.

4.4.2. LE PRINCIPE

Plaçons-nous maintenant dans le cadre d'un système d'authentification de fichiers. Une fonction de condensation à sens unique est appliquée au fichier et le résultat est conservé dans un environnement sûr où il est protégé contre les modifications. Le fichier peut maintenant transiter par des milieux non protégés.

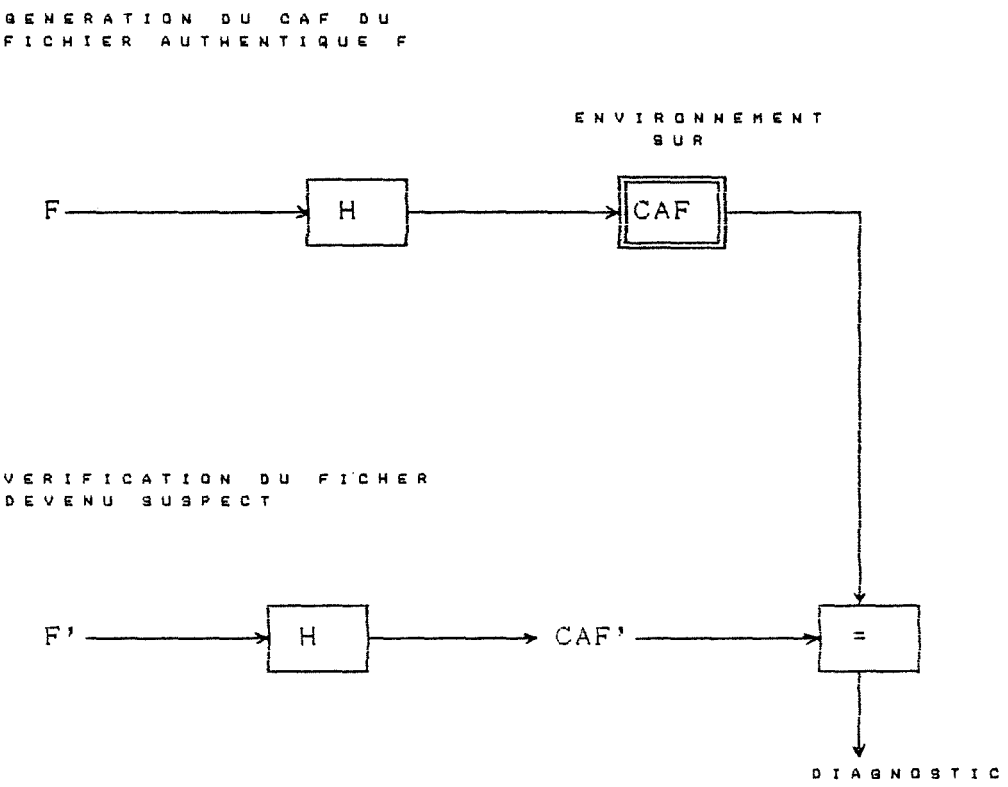
Chaque fois que l'intégrité du fichier doit être vérifiée, la même fonction lui est appliquée, et le résultat est comparé avec celui stocké en milieu sûr. Si les résultats sont différents, on est certain que le fichier a été modifié. Si les résultats sont identiques, de par la définition de la fonction de condensation à sens unique, on est quasiment certain que le fichier est resté inchangé. On a donc à ce moment authentifié le fichier.

Dans ce contexte, le condensat prend le nom de **CAF** (Code d'Authentification de Fichiers)(*), et la fonction de condensation à sens unique s'appelle **fonction d'authentification de fichiers**, ou plus simplement **fonction d'authentification**.

Cette méthode d'authentification est illustrée par la figure 1.4.

(*) Le terme "CAF" nous est personnel. Il nous semble mieux adapté ou moins ambigü que ceux couramment utilisés dans la littérature relative aux fonctions de compressions à sens unique, à savoir "digest" ou "summary" <PRICE, 85>, "compressed encoding" <AKL, 84>, "MAC" (Message Authentication Code) <ANSI, 82>, "MDC" (Manipulation Detection Code) <MEYER, 88>, "checksum" <YUVAL, 79>, "authentication tag" <WEGMAN, 81>, "authenticator" <SLOANE, 82>, ...

FIGURE 1.4. : Authentification par comparaison de CAF



4.4.3. UNE EVALUATION

Cette méthode a été proposée par différents auteurs pour assurer l'authentification des fichiers <AKL, 84>, <CAMPANA, 88>, <MEYER, 88>, <POZZO, 86>, <MONOD-BROCA, 86>. Elle possède en effet de nombreux avantages par rapport aux autres méthodes précédemment évoquées:

- Elle offre sans aucun doute possible plus de sûreté que l'emploi de CDE <DAVIES, 84, 123>.

- Elle est de façon générale extrêmement plus rapide que la signature numérique <PRICE, 85>.

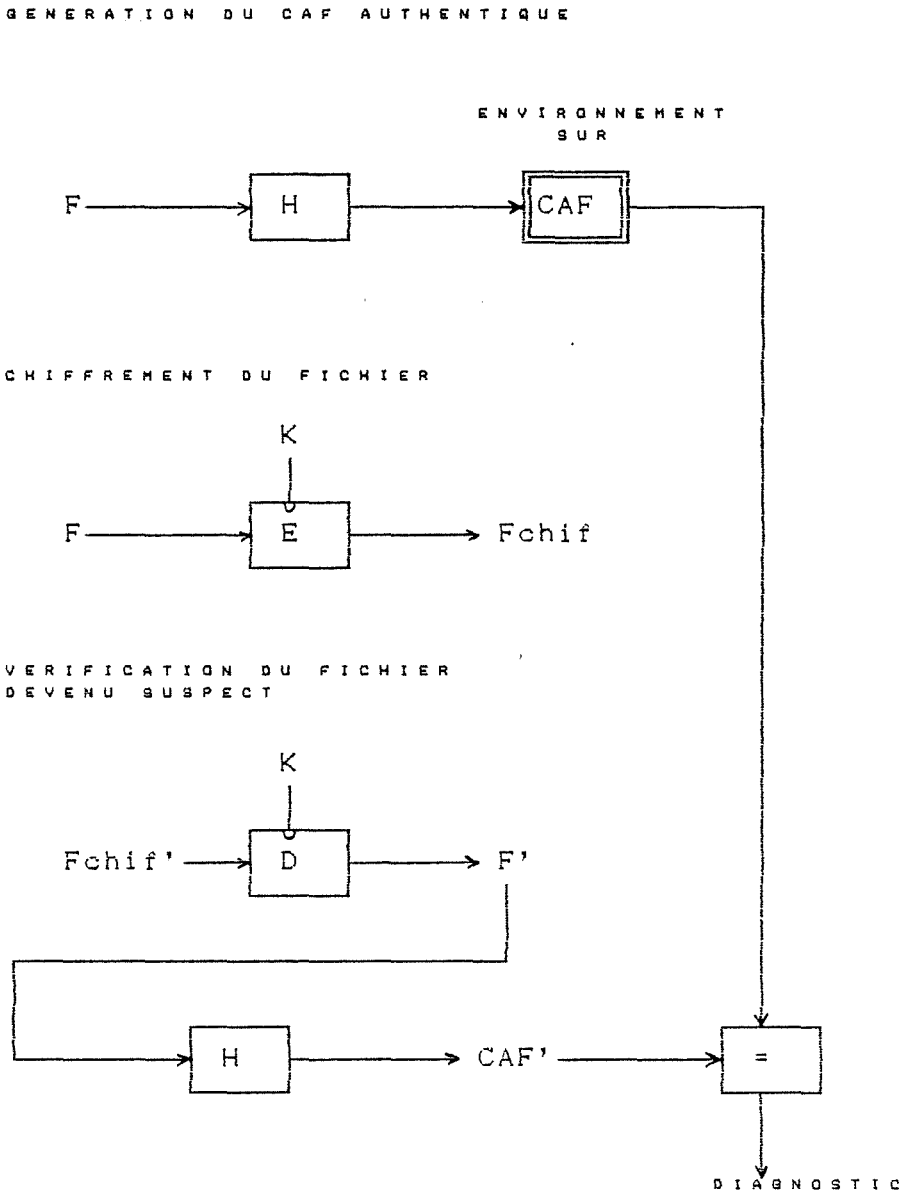
- Elle est plus pratique et plus sûre que la méthode utilisant le chiffrement.

Ce dernier point mérite quelques précisions:

- Les fonctions d'authentification sont plus pratiques que le chiffrement parce qu'elles permettent de conserver le fichier protégé 'en clair'. Pour les applications où non seulement l'intégrité, mais aussi la confidentialité des fichiers doivent être assurées, les fonctions d'authentification peuvent être combinées au

chiffrement. Dans ce cas, le CAF se calcule sur le fichier authentique avant chiffrement tandis que la vérification porte sur le fichier déchiffré, ce qu'exprime la figure 1.5.

FIGURE 1.5. : Authentification par comparaison de CAF conjugée au chiffrement



Cette approche sépare le traitement de la confidentialité de celui de l'intégrité, ce qui est essentiel dans un système informatique où ces deux aspects de la sécurité des fichiers ne vont pas nécéssairement de pair <JUENEMAN,84>, <JUENEMAN, 85>.

- Dans certains contextes, l'authentification par

chiffrement est peu sûre car elle repose sur le secret d'une clé. Nous avons déjà montré que si cette clé est découverte, modifier un fichier chiffré est un jeu d'enfant. Ce n'est pas le cas pour les fonctions d'authentification. Le fait qu'elles soient 'à sens unique' implique que même si l'ennemi en connaît tous les détails, il lui est impossible en pratique de trouver un fichier frauduleux correspondant à un CAF donné.

4.4.4. LE RETOUR A LA NOTION DE SIGNATURE NUMERIQUE

En plus des indéniables qualités citées ci-dessus, il est important de noter que le concept de fonction d'authentification peut être avantageusement combiné à celui de signature numérique.

Au paragraphe 4.3.3., nous avons associé deux limites aux techniques de signature numérique d'un fichier: la longueur excessive de la signature et les très faibles performances. Mais si la signature porte non plus sur le fichier, mais sur son CAF ces limites tombent <DIFFIE, 76>. L'utilité d'une telle technique est à chercher dans le besoin d'intégrité du condensat.

Si un CAF n'est pas protégé (imaginons qu'il soit placé dans un milieu non sûr en compagnie du fichier qu'il est sensé authentifier), et si la fonction d'authentification est connue de l'ennemi, ce dernier peut modifier le fichier, recalculer le nouveau CAF et le substituer à l'ancien. Pour résoudre ce problème nous proposons la méthode d'authentification suivante <POZZO, 86> qui est illustrée par la figure 1.6.

- Dans un premier temps, on calcule le CAF du fichier authentique F:

  $$CAF = H(F),$$

  et on le signe. La paire F - S(CAF) peut maintenant être exposée dans un environnement non sûr.

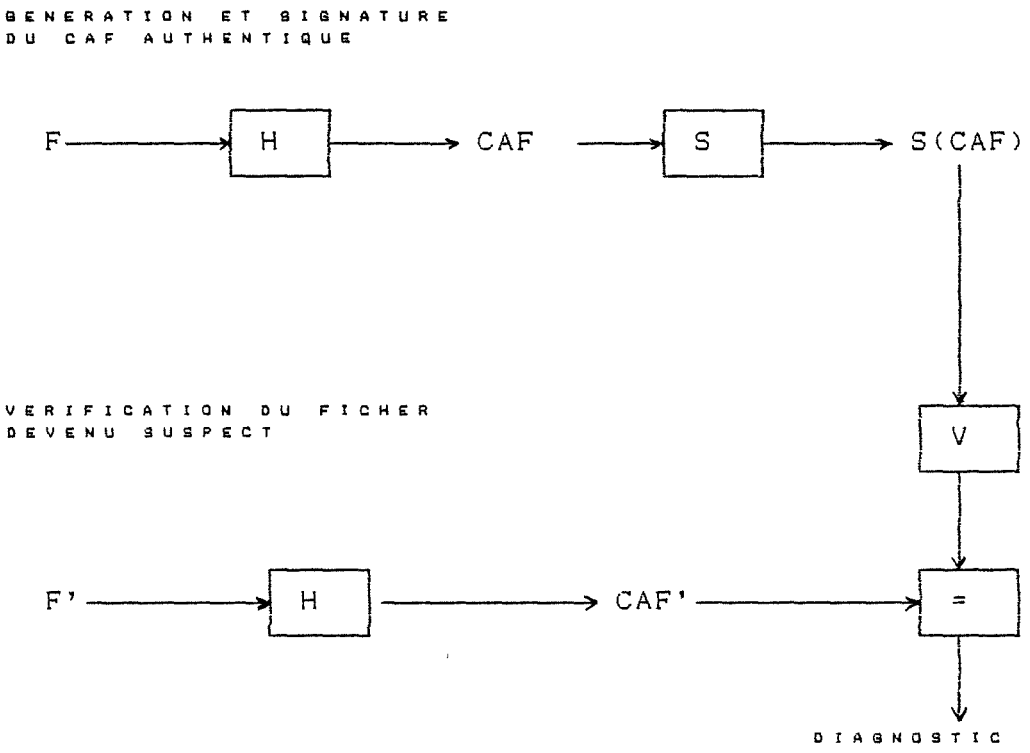
- Lorsque l'on désire vérifier l'authenticité du fichier F devenu suspect , on lui applique la fonction d'authentification

  $$CAF' = H(F')$$

  pour obtenir un code d'authentification. Ce dernier est alors comparé à celui du fichier authentique obtenu grâce à la fonction de vérification publique appliquée à S(CAF):

CAF = V(S(CAF)).

FIGURE 1.6. : Authentification par comparaison de CAF conjuguée à la signature numérique



Si CAF' ≠ CAF, on peut être certain qu'une modification a eu lieu (bien qu'il soit impossible de déterminer si cette modification porte sur F, sur S(CAF) ou sur les deux à la fois). Par contre, si CAF' = CAF, il est quasiment certain que F' est toujours un fichier authentique. La preuve de ceci peut être donnée en trois étapes.

1) Un ennemi peut changer F en un fichier frauduleux F'. Mais alors, de par les propriétés des fonctions de condensation à sens unique, CAF' = H(F') n'est plus égal à CAF, ce qui signifie que l'attaque est détectée.

2) Le premier point met en évidence le fait que l'ennemi, après avoir modifié F, doit également s'attaquer à sa signature. Pour cela, il calcule CAF' = H(F'), et essaie de trouver α = S(CAF') de telle sorte que CAF' = V(α). Mais malheureusement pour lui, il ne peut le faire de par les propriétés de la signature numérique (seule la possession de la fonction de signature secrète permet de générer α).

3) Une alternative à l'attaque décrite ci-dessus consiste à partir d'une signature existante S(CAF) pour laquelle on connaît la vérification V(S(CAF)) = CAF. L'ennemi doit alors trouver un fichier frauduleux F' associé à CAF par la fonction d'authentification : CAF = H(F'). Il en est toutefois incapable si la fonction de condensation est à sens unique.

Bien entendu, la signature numérique n'est pas la seule protection possible pour assurer l'intégrité des CAF. Les moyens de stockage non altérables évoqués précédemment au paragraphe 2.3.2. en sont une autre. Toutefois, cette protection reste entièrement locale. La cryptologie redevient nécessaire dès que le CAF est transféré d'un environnement sûr à un autre en passant par un milieu hostile.

4.5. NOTRE CHOIX

Nous venons d'examiner quatre méthodes d'authentification. Notre choix se porte sur l'utilisation de codes d'authentification, et ce pour les raisons évoquées dans les paragraphes 4.4.3. et 4.4.4.

Ce choix peut être contesté. Il existe en effet des situations où une autre méthode d'authentification serait sans doute plus adaptée. Nous pensons tout particulièrement au chiffrement qui peut être très interessant dans un système informatique centralisé pour lequel le secret des clés peut être garanti et où la confidentialité des fichiers est toujours requise. Plusieurs systèmes de protection des fichiers sont d'ailleurs basés sur ce principe (cfr notamment <WHITE,87>).

5. CONCLUSION

Au terme de cette première partie consacrée à l'étude du besoin relatif à la sécurité des fichiers et des diverses solutions envisageables, nous avons fixé notre choix sur l'utilisation de fonctions de condensation à sens unique générant des codes d'authentification de fichiers.

Il nous faut maintenant trouver de telles fonctions, ce qui n'est pas chose aisée à cause des hypothèses très strictes (rapidité et sécurité) que nous avons formulées à leur sujet.

Il nous faut ensuite définir les caractéristiques d'un système les mettant en oeuvre, car même si nous disposons d'une fonction d'authentification parfaite, elle ne sert à rien si un certain nombre de dangers ne sont pas écartés. A titre d'exemple, il est essentiel d'assurer l'intégrité de la fonction d'authentification et des CAF.

Ces différents points seront développés dans les parties suivantes de ce mémoire.

Deuxième Partie

UN MODELE DE FONCTION D'AUTHENTIFICATION

1. INTRODUCTION

L'objet de la seconde partie de ce travail est la recherche d'un **modèle de fonction d'authentification.** Ce modèle ne sera pas exhaustif. Il existe en effet une multitude de fonctions d'authentification présentes dans la littérature et il paraît donc difficile de trouver un modèle les incluant toutes.

Nous avons plutôt pris le parti de dégager les principales caractéristiques qui se retrouvent dans la grande majorité des fonctions d'authentification. A défaut d'être exhaustif, notre modèle sera donc pratique. Pour atteindre cet objectif, nous avons utilisé deux approches :

- La première considère la fonction (ici notée H) comme étant une opération élémentaire qui, à partir d'un fichier F, génère un CAF :

  $$CAF = H(F).$$

  Dans cette approche , H est étudiée suivant les propriétés liant F au CAF. Ces propriétés requises pour les fonctions d'authentification seront l'objet du **chapitre 2.**

- La deuxième approche 'entre' dans H . Elle dissèque les mécanismes qui permettent à la fonction d'authentification de satisfaire les propriétés qui lui sont requises. Ces mécanismes sont les briques de base grâce auxquelles la grande majorité des fonctions sont construites. Ils seront traités aux **chapitres 3 et 4.**

Le **chapitre 5,** sous le titre, " les qualités et éléments de structure d'une bonne fonction d'authentification ", constituera une synthèse de notre modèle.

2. LES PROPRIETES REQUISES

Nous avons défini une fonction d'authentification comme étant une fonction de condensation à sens unique qui, à un fichier de taille arbitraire, associe une et une seule valeur de taille fixe : le Code d'Authentification de Fichiers, ou CAF. Si H est une telle fonction et F un fichier, la génération du CAF s'exprime de façon synthétique par:

CAF = H(F).

Cette définition est peut-être un peu trop concise. Ci-dessous, nous allons l'affiner en donnant les propriétés requises pour F, pour le CAF et pour H.

2.1. LA PROPRIETE REQUISE POUR F

F doit pouvoir être n'importe quel fichier, de sorte qu'il puisse se définir comme "une suite d'un nombre quelconque de bits". Cette propriété est exprimée dans la définition de fonction d'authentification par le fait que celle-ci associe toujours **au moins un** CAF à n'importe quel fichier.

2.2. LES PROPRIETES REQUISES POUR LE CAF

Si un CAF peut être associé à plusieurs fichiers, plusieurs CAF ne peuvent l'être à un seul fichier. Cette propriété est exprimée dans la définition de fonction d'authentification par le fait que celle-ci associe toujours **au plus un** CAF à n'importe quel fichier.

Le CAF doit être significativement plus petit que le fichier, et ceci pour des raisons d'efficacité : plus le CAF est petit, et plus il est simple à protéger et à manipuler.

Le CAF doit également être de taille fixe. Ceci n'est pas une condition théorique, mais plutôt pratique. Il est en effet alors beaucoup plus simple à manipuler.

2.3. LES PROPRIETES REQUISES POUR H

H doit jouer le rôle d'une fonction détectrice d'erreur <AKL, 84>. En d'autres termes, **le changement d'un bit de F doit affecter H(F) d'une manière profonde et imprévisible.**

Si F' est le nouveau fichier obtenu par une légère modification de F, et si CAF' = H(F'), les termes 'profond' et 'imprévisible' peuvent alors se définir de la façon suivante:

- profond : Il n'y a aucune dépendance visible entre CAF' et CAF, ce qui s'exprime de façon plus formelle par: *chaque bit de CAF' a changé de valeur avec une probalité de 0,5 si on le compare au bit de même poids de CAF*;

- imprévisible : Il n'y a pas moyen de prévoir la valeur que va prendre CAF' sans effectivement calculer H(F').

Remarque : Nous venons en fait d'exprimer que H doit avoir les propriétés d'un générateur pseudo-aléatoire, ce qui signifie que pour un observateur externe, H produit des résultats qui ont pour source le pur hasard. Le terme 'pseudo' est nécessaire puisque H est déterministe, c'est-à-dire qu'à un même fichier correspond toujours le même CAF.

Le fait que H soit une fonction détectrice d'erreurs ne suffit pas. Une erreur est sensée être provoquée par le hasard, et le hasard n'est pas très intelligent. Les fraudeurs, eux, le sont. C'est pourquoi il nous faut en plus spécifier :

Etant donnés H et F, il est impossible, dans des conditions pratiques, de trouver F' ≠ F tel que H(F') = H(F).

Cette propriété est très importante. En effet, si un ennemi peut trouver F' tel que H(F') = H(F), il peut alors très facilement tromper le système d'authentification en remplaçant F' par F. Elle nécéssite toutefois quelques remarques:

- D'après Mr QUISQUATER <QUISQUATER, 88>, il est possible d'être encore plus sévère quant aux propriétés de la fonction H en lui imposant d'être "collision free". Cette expression, empruntée à <DAMGARD, AP>, signifie qu'**étant donnés H, il est impossible, dans des conditions pratiques, de trouver n'importe quelle paire F - F' telle que F ≠ F' et H(F) = H(F')**.

  Cette définition est beaucoup plus exigeante que la première car elle autorise l'ennemi à jouer sur deux variables: F et F' (alors que la définition précédente ne lui permettait que de manipuler F', F étant imposé). Selon Mr QUISQUATER, alors que de nombreuses fonctions d'authentification satisfont à la première définition, rares sont celles qui résistent aux exigences de la deuxième.

  Il est légitime de se poser la question de savoir comment un ennemi peut profiter d'une telle paire de fichiers frauduleux. En fait, elle ne lui sert à rien, sauf s'il peut amener l'autorité responsable du

système d'authenfication (*) à accepter l'un des deux fichiers (ayant l'air inoffensif). Le CAF est alors calculé et placé dans un environnement sûr (il est signé par exemple). Il suffit alors à l'ennemi de remplacer le fichier inoffensif par le deuxième (à la sémantique frauduleuse). Puisque ces deux fichiers ont même CAF, la supercherie n'est pas détectée <MEYER, 88>.

Devons-nous pour autant abandonner notre première définition de fonction d'authentification pour contrer ce type d'attaque? Nous n'en sommes pas certains.

Rappelons en effet que nous avons, dans la première partie, posé une hypothèse selon laquelle le contexte du développement des fichiers est sans reproche et que c'est un fichier authentique qui est présenté au système d'authentification. De plus, notre objectif est de protéger un certain nombre de fichiers importants et non d'empêcher un ennemi de créer ses propres fichiers frauduleux. C'est pourquoi, pour la suite du travail, nous conserverons notre première définition comme référence.

Toutefois nous sommes conscients qu'il existe des contextes où un tel choix n'est pas acceptable. Pour cette raison, nous examinerons dans la quatrième partie comment les fonctions satisfaisant au mieux à la première définition se comportent face au critère "collision free".

- Un fichier n'est intéressant pour un ennemi que s'il possède une signification pour une quelconque application informatique frauduleuse. Par mesure de sécurité, nous sommes beaucoup plus sévères en ce qui concerne les propriétés des fonctions d'authentification. Dans le cas où il est possible de découvrir un fichier frauduleux, même sans signification, nous considérerons cela comme une sérieuse indication de faiblesse et nous rejetterons la fonction d'authentification en question.

- L'ennemi peut se trouver dans bien des contextes différents. Suivant le cas, la tâche à laquelle il est confronté peut-être très simple ou pratiquement impossible. Par la suite, sauf stipulation contraire, l'ennemi sera toujours considéré comme étant dans l'environnement qui lui est le plus favorable. Cela signifie que nous supposerons toujours qu'il possède toute l'information concernant les fichiers, les CAF,

---

(*) Il est nécéssaire qu'une autorité ait le pouvoir de décider qu'un fichier est authentique. Cela peut être le ou les responsable de son développement, un organisme d'audit ou, pour reprendre une expression extraite de <POZZO, 86>, un 'System administrator' (*"one or more persons trusted not to compromise the security or integrity of the system"*).

la ou les fonctions d'authentification et la ou les clés.

Si un système d'authentification s'avère efficace sous cette hypothèse, il n'en sera que meilleur une fois placé dans un contexte plus réaliste où tout sera mis en pratique pour empêcher l'ennemi d'agir. De plus, l'hypothèse devient nécessaire lorsque l'attaquant est un utilisateur autorisé, qui possède un droit d'accès et connaît très bien les mesures de sécurité prises. Ce cas de figure est courant dans la pratique.

Répétons une fois encore que moins il y a d'éléments secrets nécessaires à la garantie de la sécurité de notre système et plus il peut être considéré comme solide.

2.4. EN RESUME

En résumé, la fonction d'authentification H qui associe un fichier F de n bits à un et un seul CAF de m bits doit avoir les propriétés suivantes <AKL, 84>:

(1) m est une constante significativement plus petite que n;

(2) calculer H(F) à partir de F est rapide et pratique;

(3) changer un bit de F altère H(F) de façon profonde et imprévisible;

(4) étant donné H et F, il est impossible, dans des conditions pratiques, de trouver un fichier F' ≠ F tel que H(F') = H(F).

Il est intéressant de remarquer que

" The seemingly conflicting efficiency requirements (1) and (2), on the one hand, security requirements (3) and (4), on the other, should be balanced when selecting the function H " <AKL, 84>.

3. TRAITEMENT ET CHAINAGE DES BLOCS D'UN FICHIER

Parmi les propriétés que nous avons associées à une bonne fonction d'authentification, il y en a une qui s'exprime de la façon suivante :

"Une variation d'un seul bit dans le fichier à protéger doit résulter en une modification profonde et imprévisible du CAF associé à ce fichier"

Ceci sous-entend que l'entièreté du fichier est traitée par la fonction d'authentification. Or, il est impensable de manipuler tout un fichier (qui peut avoir une taille se chiffrant en millions d'octets) en une seule opération. Il faut le découper en blocs qui sont alors traités successivement.

Dans ces conditions, afin de garantir la propriété citée ci-dessus, les blocs doivent être liés entre eux de façon à ce qu'une simple erreur sur un bit dans un bloc ait des effets qui se répercutent jusqu'au CAF. Ce lien porte le nom de 'chaînage des blocs'.

Le traitement et le chaînage des blocs d'un fichier sont étudiés dans ce chapitre. Nous attirons l'attention sur le fait que cette étude ne se veut pas exhaustive, mais plutôt intuitive. Elle a en effet pour objectif de mettre en évidence deux caractéristiques qui se retrouvent, sous une forme ou sous une autre, dans quasiment toutes les fonctions d'authentification.

3.1. DEUX MAUVAISES FONCTIONS D'AUTHENTIFICATION

Reprenons notre problème de base. F est un fichier que nous voulons authentifier. Pour cela, il nous est nécessaire de posséder une fonction d'authentification H qui calcule un code à partir de F : CAF = H(F). Pour pouvoir réaliser ce calcul, nous avons vu que F doit être découpé en blocs. Imaginons que F soit considéré comme une suite de n blocs de t bits : F1, F2, ..., Fn. Nous faisons pour le moment l'hypothèse que la découpe engendre un dernier bloc complet (Fn a une longueur de t bits).

Nous voici maintenant au pied du mur : il nous faut trouver une fonction H(F1, ..., Fn) qui ait si possible toutes les propriétés citées au chapitre précédent. Afin d'illustrer l'extrême complexité de ce problème, nous commencerons par donner deux mauvaises fonctions.

La première est très simple mais n'offre aucune sécurité. Elle construit le CAF en mettant bout à bout (nous disons également 'en concaténant') le premier bit de chaque bloc. Si F est constitué de mille blocs, le CAF aura donc une longueur de mille bits. Un tel CAF possède un inconvénient majeur : il reste constant pour toutes les modifications de F qui ne touchent pas au premier bit de chaque bloc.

Cet exemple illustre bien le fait que le résultat de la fonction d'authentification doit être dépendant de tous les bits du fichier à authentifier.

La deuxième fonction construit le CAF d'un fichier F en faisant un OU-exclusif bit-à-bit (abréviation : $\oplus$) entre tous les blocs. Il s'agit donc de faire :

$$CAF = F1 \oplus F2 \oplus \ldots \oplus Fn.$$

Le CAF ainsi obtenu possède une longueur fixe (t bits) et est dépendant de tous les bits du fichier à authentifier. Ces deux qualités n'existent pas dans le premier exemple.

Malheureusement, un ennemi peut facilement trouver un autre fichier, frauduleux celui-là, qui possède le même CAF. Supposons par exemple que X et Y soient deux blocs qu'il désire insérer dans le fichier F sans modifier le CAF. Nous ne nous étendrons pas sur la sémantique que peuvent prendre ces blocs. Le seul fait qu'ils puissent être introduits dans le fichier constitue un danger ouvrant la porte à toutes les fraudes imaginables.

L'ennemi, dans le cas où la deuxième fonction d'authentification est utilisée, va non seulement insérer X et Y, mais aussi un troisième bloc Z tel que $Z = X \oplus Y$. Cette situation peut être illustrée par la figure 2.1.

FIGURE 2.1 : Insertion des blocs frauduleux X, Y et Z dans le fichier F

F1	F2	.....	Fi	X	Y	Z	Fi+1	.....	Fn

Puisque par définition, le CAF est obtenu par un OU-exclusif bit-à-bit entre tous les blocs du fichier, cette insertion de trois blocs va impliquer trois $\oplus$ supplémentaires. Si CAF' est le nouveau CAF, il se calcule par :

$$CAF' = CAF \oplus X \oplus Y \oplus Z,$$

ou encore, puisque $Z = X \oplus Y$,

$$CAF' = CAF \oplus X \oplus Y \oplus X \oplus Y.$$

Or, de par les propriétés de $\oplus$,

$$X \oplus X = 0, \text{ et } Y \oplus Y = 0.$$

Ceci implique que

$$CAF' = CAF \oplus 0 \oplus 0 = CAF.$$

Dès lors, puisque CAF' = CAF, les insertions ne seront pas détectées.

3.2. LE TRAITEMENT D'UN BLOC

Cette deuxième fonction d'authentification est mauvaise parce qu'elle ne transforme pas les blocs qui sont introduits. Ceux-ci apparaissent en effet sous leur aspect initial dans la formule de construction du CAF :

$$CAF = F1 \oplus \ldots \oplus Fn.$$

Or, nous avons vu qu'un ennemi pouvait profiter de cette faiblesse pour insérer des blocs frauduleux non détectés. Pour pallier cette défaillance, et donc augmenter la sécurité de la fonction d'authentification, il est nécessaire que celle-ci transforme chaque bloc dès qu'il est introduit. Nous exprimerons cela sous la forme :

$$Hi = t(Fi),$$

où Hi est le bloc qui résulte "de la transformation" (nous disons aussi 'du traitement') du bloc Fi par la fonction t.

Cette transformation, pour être valable, doit respecter un certain nombre de règles dont les plus importantes sont :

- la transformation est profonde : Hi n'a en effet aucune dépendance visible avec Fi;
- elle est imprévisible : sans appliquer t au bloc Fi, il est impossible de prévoir ce que va être Hi;
- elle est 'à sens unique' : étant donné Hi = t(Fi), il est impossible de trouver un autre bloc Fi' ≠ Fi tel que Hi = t(Fi').

Si une fonction d'authentification réalise une telle transformation de chaque bloc introduit, la tâche de l'ennemi devient alors beaucoup plus compliquée.

Pour nous en convaincre, reprenons la deuxième mauvaise fonction d'authentification, et apportons lui une légère modification. Le CAF n'est plus construit par un ⊕ entre tous les blocs du fichier, mais par un ⊕ entre tous les blocs traités par une fonction de transformation ayant les propriétés susnommées. En d'autres termes :

$$CAF = H1 \oplus H2 \ldots \oplus Hn.$$

Si un ennemi veut introduire deux blocs frauduleux X et Y, il lui est maintenant très difficile de trouver un bloc Z qui 'compense' les deux premiers. Il faut en effet que Z soit tel que t(Z) = t(X) ⊕ t(Y) (alors que dans la première version, il suffisait de prendre Z = X ⊕ Y).

La solution la plus couramment employée pour réaliser la transformation

$$Hi = t(Fi)$$

est d'utiliser une fonction de chiffrement :

$$Y = E_K(X).$$

Une telle fonction possède la propriété intéressante de produire un résultat Y qui semble n'avoir aucune corrélation, ni avec la donnée X , ni avec la clé K. Concrètement, cette propriété permet à un bloc Fi d'être traité de deux façons différentes :

- l'une est classique et consiste à chiffrer le bloc

$$Hi = E_K(Fi),$$

- l'autre peut paraître plus étrange, mais est tout aussi efficace. Elle fait intervenir le bloc au niveau de la clé :

$$Hi = E_{Fi}(C) \quad \text{où C est une constante.}$$

Ces deux techniques seront illustrées d'un exemple à la section 3.5. de ce chapitre.

3.3. LE CHAINAGE DES BLOCS

La première étape de la construction d'une fonction d'authentification est franchie. Il faut maintenant se poser la question de savoir comment combiner les résultats Hi afin d'obtenir un CAF ayant les propriétés désirées.

Une première méthode consiste à effectuer une opération mathématique destinée à réunir les Hi en un seul résultat. On peut imaginer par exemple :

- de les concaténer (notation: $\ll$):

$$CAF = H1 \ll H2 \ll \ldots \ll Hn,$$

- ou, comme nous l'avons déjà suggéré, de les combiner à l'aide d'un OU-exclusif bit-à-bit :

$$CAF = H1 \oplus H2 \ldots \oplus Hn.$$

Ces solutions sont mauvaises. La première car elle conduit à former des CAF de très grande taille. La seconde parce que le traitement d'un bloc est totalement indépendant du traitement d'un autre.

Illustrons cette dernière faiblesse par un exemple. Que se passe-t-il si un ennemi introduit deux fois le meme bloc frauduleux X dans le fichier F ? Ces blocs génèrent deux fois le même résultat intermédiaire Hx = t(X), et le CAF n'est donc pas modifié (car Hx ⊕ Hx = 0).

Il est donc nécessaire que chaque résultat Hi soit fonction non seulement du bloc introduit, mais aussi de tous ceux qui le précèdent. Et la façon la plus simple de réaliser ce lien (nous parlons aussi de 'chaînage') est de faire en sorte que Hi soit fonction de Hi-1. Le chaînage des différents blocs d'un fichier peut donc s'exprimer de manière synthétique par la formule :

$$Hi = f(Hi-1, Fi),$$

où f est telle que si Fi ou Hi-1 est modifié, Hi l'est aussi. Nous disons que Hi est le résultat intermédiaire entre Fi et Fi+1. Hn quant à lui est le résultat final duquel est déduit directement le CAF.

Le principe du chaînage possède un effet de bord intéressant: lorsqu'une modification s'est glissée dans un bloc Fi, elle affecte Hi, et donc elle se répercute sur Hi+1, Hi+2, et ainsi de suite jusqu'à Hn. Ce mécanisme est connu dans la littérature sous le nom évocateur de 'propagation des erreurs'.

Nous illustrerons le concept de chaînage par deux exemples à la section 3.5. de ce chapitre. Nous pouvons toutefois préciser dès à présent que les techniques les plus couramment employées utilisent le OU-exclusif bit-à-bit :

$$Hi = t(Hi-1 \oplus Fi),$$

$$Hi = Hi-1 \oplus t(Fi), \text{ ou encore}$$

$$Hi = Fi \oplus t(Hi-1).$$

(Pour rappel, t est la fonction de transformation introduite au point 3.3 de cette partie).

3.4. EN SYNTHESE

Nous avons mis en évidence une série de caractéristiques que doit posséder une bonne fonction d'authentification. A titre de synthèse, nous retiendrons que la fonction d'authentification H(F1, ..., Fn) doit être sensible à tous les bits de F et comprendre une fonction de transformation d'un bloc, couplée à un mécanisme de chaînage.

La manière dont ces caractéristiques sont mises en oeuvre est différente d'une fonction d'authentification à une autre. Nous pouvons nous en rendre compte en examinant les deux exemples qui sont présentés à la section suivante.

3.5. DEUX EXEMPLES

3.5.1. LE MODE D'OPERATION CIPHER BLOCK CHAINING

Dans <NBS, 80>, deux méthodes d'utilisation du DES à des fins d'authentification sont décrites. Elles sont connues sous le nom de 'modes d'opération définis pour le DES'. En principe, n'importe quel algorithme de chiffrement peut être utilisé suivant ces modes d'opération, mais en pratique (ainsi que dans la suite de ce mémoire), c'est le DES qui sera préféré.

Ces deux méthodes d'authentification sont basées respectivement sur la technique dite "Cipher Block Chaining" (CBC), et sur celle dite "Cipher FeedBack" (CFB).

Nous ne développerons que la première, plus rapide (11 Mo/s pour la plus performante implémentation matérielle connue <FERREIRA, NP>) et donc mieux adaptée à l'authentification de gros fichiers. Pour plus de renseignements sur la technique CFB, on peut consulter notamment <DAVIES, 84, 97-103).

Pour la génération d'un CAF en mode CBC, le fichier F est tout d'abord divisé en n blocs de 64 bits : F1, ..., Fn. Si le dernier bloc possède moins de 64 bits, il est complété de 0 afin d'arriver à cette longueur.

La figure 2.2 illustre le fonctionnement de ce mode d'opération. Toutes les opérations travaillent sur des opérandes de 64 bits. L'opération ⊕ est le OU-exclusif bit-à-bit introduit précédemment.

Le principe du mode d'opération CBC est simple :

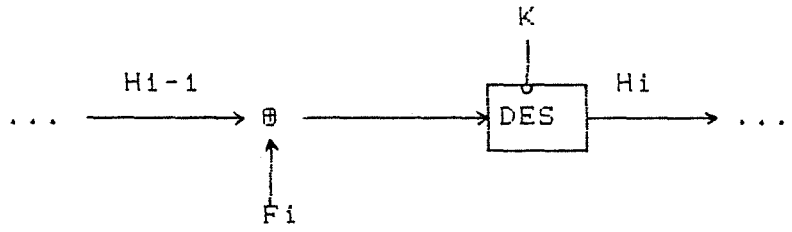
- Un bloc du fichier (disons Fi) est tout d'abord combiné au résultat intermédiaire précédent (Hi-1). Cette combinaison s'effectue par un OU-exclusif bit-à-bit. Elle a pour conséquence de lier au traitement du bloc Fi le résultat du traitement de tous les blocs qui l'ont précédé.

- Hi-1 ⊕ Fi est ensuite chiffré par le DES utilisant une clé K.

- Le résultat de cette opération, Hi, constitue le nouveau résultat intermédiaire.

- Le principe est 'itéré' de nouveau pour le bloc suivant.

FIGURE 2.2 : Le mode d'opération CBC



Afin de pouvoir manipuler plus facilement ce mode d'opération, nous utilisons la notation suivante :

$$Hi = E_K(Hi-1 \oplus Fi).$$

Pour que la description soit complète, il nous faut spécifier le début et la fin du processus d'authentification :

- Le premier bloc ne possède bien sûr pas de 'résultat intermédiaire précédent'. Celui-ci est remplacé par une valeur d'initialisation (VI) de 64 bits. Nous obtenons donc, pour le premier bloc F1, la situation suivante :

  $$H1 = E_K(VI \oplus Fi).$$

  Il est possible de garder secret le contenu de cette variable afin de compliquer le travail d'un ennemi. Cependant, l'objectif de VI est de prévenir une attaque sur les premiers blocs dans le cas où le CBC est utilisé à des fins de chiffrement <DAVIES, 84, 93-94). Dans le cadre de l'authentification, ce problème ne se pose pas. Il est donc plus simple de considérer VI comme une constante (la plupart du temps, la valeur zéro est choisie).

- le CAF est dérivé du dernier résultat (Hn) en prenant les b bits les plus significatifs. Si b est inférieur à 64, le stockage des CAF nécessite moins de place mémoire. Il est peut-être plus étonnant d'affirmer qu'un CAF ainsi écourté offre dans certaines circonstances une sécurité supérieure à un autre de 64 bits. Expliciter ceci n'entre toutefois pas dans le cadre de ce chapitre, mais nous y reviendrons dans la troisième partie.

Cette fonction d'authentification est très efficace dans son rôle de détectrice d'erreurs:

"*Because of the well established forward error propagating properties of the Cipher Block Chaining*

mode of operation, the change of even so much as a single bit in the (file) would cause an unpredictable change to every bit in the (CAF) with a probability of 50 % for each bit" <JUENEMAN, 84>.

Par contre, elle nécéssite absolument le secret de la clé <ANSI, 82>. Si ce n'est pas le cas, un ennemi peut facilement construire un fichier frauduleux ayant le même CAF que n'importe quel fichier authentique. Nous montrerons comment dans la troisième partie.

3.5.2. UNE FONCTION DUE A RABIN <RABIN, 78>

Cette fonction d'authentification se différencie du mode CBC par le fait que les blocs du fichier (F1, ..., Fn) sont vus par la fonction de chiffrement E comme des clés, et non pas comme des données.

Il s'agit en fait d'utiliser les n blocs en tant que clés successives pour chiffrer n fois une valeur initiale VI. Cette valeur peut être choisie aléatoirement et gardée secrète afin de compliquer la tâche de l'attaquant. Elle peut également être fixée une fois pour toutes par souci de standardisation. L'algorithme est donc :

$$H1 = E_{F1}(V1),$$

$$H2 = E_{F2}(H1),$$

$$\vdots$$

$$Hn = E_{Fn}(Hn-1).$$

Le CAF est obtenu en prenant les s bits les plus significatifs de Hn.

Si le DES est utilisé, chaque bloc Fi possède 56 bits (taille d'une clé pour le DES), tandis que Hn a lui une longueur de 64 bits.

4. LE RENFORCEMENT DU CHAINAGE

4.1. LE BESOIN DE TECHNIQUES SUPPLEMENTAIRES

Les techniques de chaînage des blocs décrites précédemment ne produisent pas nécéssairement des fonctions d'authentifications sûres. Nous verrons, dans les parties 3 et 4 comment des attaques peuvent en venir à bout. Nous nous contentons ici d'un petit exemple intuitif présentant le besoin de techniques supplémentaires permettant de construire de meilleures fonctions d'authentification.

Soit le mode CBC (présenté au chapitre précédent) utilisé pour authentifier le fichier F (F1, F2, ..., Fn). Si une modification apparaît dans le bloc Fi, le résultat intermédiaire

$$Hi = E_K(Hi-1 \oplus Fi)$$

est modifié. De par le principe de propagation des erreurs, cette modification est répercutée sur Hi+1, Hi+2, et ainsi de suite jusqu'à Hn.

Mais si Fi+1 est également altéré, il est possible que

$$Hi+1 = E_K(Hi \oplus Fi+1)$$

retrouve sa valeur initiale (nous montrerons dans la troisième partie que c'est le cas si un ennemi choisit habilement la nouvelle valeur de Fi+1). Dans ce cas, puisque Hi+1 est inchangé, le mécanisme de propagation des erreurs est arrêté et Hn ne subit aucune modification.

Pour pallier cette faiblesse, il est intéressant d'ajouter aux fonctions d'authentifications des mécanismes qui améliorent leur capacité de propagation des erreurs et qui les rendent donc plus sûres. Parmi les plus connus, nous allons évoquer l'utilisation de Codes Detecteurs de Manipulations (ou CDM) et la combinaison de plusieurs fonctions d'authentification.

4.2. LES CODES DETECTEURS DE MANIPULATION

4.2.1. LE PRINCIPE

Les codes détecteurs de manipulation sont des codes détecteurs d'erreurs qui travaillent au niveau d'un fichier. Ils trouvent leur origine dans le besoin de redondance nécessaire pour l'authentification des fichiers par chiffrement <DAVIES, 84, 134> et doivent posséder les qualités suivantes:

"*The (CDM) must be of sufficient power to detect all manipulation or corruption of the (protected) information, but should also be simple enough to have only a slight impact upon performance*" <JUENEMAN, 84>.

Ces deux caractéristiques leur permettent d'être avantageusement utilisables pour améliorer les fonctions d'authentification. Pour illustrer ceci, reprenons l'exemple du mode CBC:

1) Le CDM d'un fichier F est calculé.

2) Le mode CBC appliqué à F:

$$Hi = E_K (Hi-1 \oplus Fi) \qquad (i = 1, \ldots, n)$$

génère un résultat final Hn.

3) Une nouvelle itération est réalisée avec le CDM comme dernier bloc de F:

$$Hn+1 = E_K (Hn \oplus CDM).$$

(Ceci vaut pour le cas où le CDM est de taille inférieure à 64 bits. Autant de nouvelles itérations sont en fait nécéssaires par tranche de 64 bits du CDM).

4) Le CAF est maintenant dérivé directement de Hn+1 (par exemple en prenant les s bits les plus significatifs, avec $0 < s < 65$).

On devine aisément l'avantage d'une telle technique par rapport au mode CBC classique:

- Si un ennemi parvient à réaliser une modification des blocs de F telle que Hn n'est pas altéré (comme suggéré en début de ce chapitre), il est toutefois presque certain que cette modification va affecter le CDM et donc que le nouveau Hn+1 va être différent du précédent.

- Si l'ennemi modifie les blocs du fichier de façon telle que le CDM ne change pas, il y a de fortes chances que Hn soit quant à lui altéré et donc que Hn+1 soit tout de même modifié.

Cet exemple illustre la grande force d'une telle construction pour une fonction d'authentification, à savoir que la moindre modification du fichier a des répercutions A PLUS D'UN ENDROIT dans le calcul du CAF. Ceci oblige l'attaquant à résoudre une série d'équations extrêmement ardues. Cette résolution requiert souvent une telle quantité de ressources

qu'elle devient impossible à réaliser dans des conditions pratiques. Nous verrons pourtant dans la quatrième partie comment des cryptanalystes de talent ont trouvé des attaques contre ce type de construction.

4.2.2. TROIS EXEMPLES <CAMPANA, 88>

La première proposition en matière de CDM, décrite notamment dans <JUENEMAN, 83>, fut d'appliquer un OU-exclusif bit-à-bit entre tous les blocs du fichier:

$$CDM = F1 \oplus \ldots \oplus Fn.$$

Cette technique a l'avantage d'être extrêmement rapide, mais elle s'est révélée peu sûre pour de nombreuses fonctions d'authentification (et notamment pour le mode CBC) <JUENEMAN, 84>, <AKL, 84>.

Afin d'éviter les faiblesses évoquées ci-dessus, d'autres CDM ont été proposés, parmis lesquels nous citerons:

- L'addition linéaire modulo 2^K des différents blocs du fichier (avec K étant au moins égal à 128 pour éviter toute une série d'attaques) <JUENEMAN, 84>, <AKL, 84>.

- Le "Quadratic Congruential" CDM décrit dans <JUENEMAN, 83> et défini comme suit:

$$Z0 = VI \qquad \text{(VI est une quelconque valeur initiale)}$$

$$Zi = (Zi\text{-}1 + Fi)^2 \text{ modulo } M \qquad (i = 1, 2, \ldots, n)$$

$$CDM = Zn.$$

La valeur proposée pour M ($2^{31} - 1$) fournit un MDC de 31 bits qui est trop court pour résister à certaines attaques. C'est pourquoi, dans <JUENEMAN, 85>, on propose d'itérer quatre fois le calcul avec le résultat d'une itération servant de valeur initiale à la suivante, ce qui permet d'obtenir un CDM plus long (124 bits). Malheureusement, dans <JUENEMAN, 87> on démontre que cette technique est sensible à une attaque subtile décrite dans <COPPERSMITH, 86>. Un nouveau schéma utilisant quatre modules distincts est alors proposé.

4.2.3. CONCLUSION

Les CDM apportent sans conteste un 'plus' à la sécurité des fonctions d'authentification. Il semble toutefois que pour qu'ils puissent rendre toute attaque quasiment impossible, une relative complexité de conception soit requise. Ceci entraîne inévitablement une perte de performances. A titre d'exemple, celle-ci se chiffre entre 20 et 40 % pour le dernier CDM proposé au paragraphe précédent utilisé en combinaison avec le mode CBC <JUENEMAN, 87>.

Dans ces conditions, il est parfois utile de se tourner vers d'autres techniques telles celles présentées ci-dessous.

4.3. LA COMBINAISON DE FONCTIONS D'AUTHENTIFICATION

L'idée d'utiliser des MDC est intéressante, mais il est possible d'aller plus loin. Plutôt que d'employer une fonction génératrice de MDC, certes rapide, mais cryptologiquement parlant assez faible, pourquoi ne pas utiliser une deuxième fonction d'authentification? Ceci est relativement coûteux en termes de performances, mais ouvre de nombreuses perspectives en matière de structuration de fonctions d'authentification.

La grande majorité de ces constructions utilise séparément les deux fonctions (par exemple H et H'):

$$CAF1 = H(F),$$

$$CAF2 = H'(F),$$

et forme le CAF de F à partir de CAF1 et de CAF2. Pour réaliser ceci, plusieurs solutions sont envisageables:

- **Une première**, décrite dans <AKL, 84>, considère CAF1 comme un CDM. Dans le cas particulier du mode CBC, cela se traduit par l'utilisation de deux clés différentes (K1 et K2) et le calcul suivant:

$$Hi = E_{K1}(Hi-1 \oplus Fi) \qquad (i = 1, \ldots, n)$$

$$CAF1 = Hn$$

$$Hi' = E_{K2}(Hi-1' \oplus Fi) \qquad (i = 1, \ldots, n)$$

$$CAF2 = Hn'$$

$$CAF = E_{K2}(CAF2 \oplus CAF1)$$

- **Une seconde solution**, inspirée de <MEYER, 82, 399> consiste à concaténer les deux CAF:

  $$CAF = CAF1 \ll CAF2.$$

  Cette solution possède l'avantage de produire un CAF de grande taille (128 bits si le mode CBC est employé), ce qui pour beaucoup d'auteurs (notamment <JUENEMAN, 87> et <MEYER, 88>) est une nécéssité pour des applications où l'authentification se doit d'être extrêmement sûre.

  De plus, cette construction augmente considérablement la capacité de propagation des erreurs puisque le moindre changement dans un bloc se répercute DIFFEREMMENT dans les deux fonctions H et H'. Il est donc presque impossible qu'un ennemi parvienne à modifier un des blocs suivant pour que la propagation s'arrête A LA FOIS dans H et dans H'.

5. LES QUALITES ET LES ELEMENTS DE STRUCTURE D'UNE BONNE FONCTION D'AUTHENTIFICATION

Ce résumé de la deuxième partie constitue une synthèse de notre modèle de fonction d'authentification.

5.1. LES QUALITES

Une bonne fonction d'authentification H possède les qualités suivantes:

- Elle génère à partir de n'importe quel fichier F un et un seul CAF de longueur fixe et significativement plus courte que celle de F.
- Elle est rapide et pratique à utiliser.
- Elle joue le rôle d'une excellente fonction détectrice d'erreurs.
- Elle est telle qu'étant donné F, il est impossible, dans des conditions pratiques, de trouver un fichier F' $\neq$ F tel que H(F') = H(F).

5.2. LES ELEMENTS DE STRUCTURE

Une bonne fonction d'authentification travaille sur les blocs successifs d'un fichier. Elle leur applique des transformations profondes et imprévisibles et des mécanismes de chaînage efficaces. Ces derniers sont éventuellement complétés de CDM et/ou combinés entre eux.

Tout ceci nécéssite bien entendu un harmonieux dosage afin d'implémenter les qualités décrites à la section précédente. Il faut se méfier d'une complication irréfléchie qui, en plus de dégrader les performances, pourrait donner une fausse impression de sécurité. La construction d'une bonne fonction d'authentification doit résulter d'une analyse approfondie. La multitude de fonctions qui se sont révélées peu sûres (cfr notamment <AKL, 84>, <WINTERNITZ,84a> et <CAMPANA, 88>) sont là pour tempérer les ardeurs des plus optimistes.

5.3. CONCLUSION

Les qualités et élements de structure énoncés ci-dessus ne constituent pas une analyse complète des caractéristiques possibles d'une fonction d'authentification. Ils sont plutôt à considérer comme un ensemble de principes et de

techniques qui forment les bases de la construction de bonnes fonctions d'authentification.

De plus, cette étude nous a permis d'introduire bon nombre de concepts, d'exemples et de notations qui seront souvent repris dans la suite de ce mémoire.

Nous venons d'examiner dans cette partie les principes et techniques à la base des fonctions d'authentification. Nous décrirons dans la quatrième partie bon nombre de fonctions qui les mettent en oeuvre. Mais notre étude ne pourrait être complète sans l'examen d'un autre aspect de la protection des fichiers: celui des attaques. Ceci sera l'objet de la partie suivante.

Troisième Partie

DES ATTAQUES CONTRE LES FONCTIONS D'AUTHENTIFICATION

1. INTRODUCTION

"I believe that some solid experience in (cryptanalysis) is a prerequisite to the successful pratice of (cryptography); one cannot devise strong systems without some idea of the manner of attacks to which such systems are likely to be subject" <COPPERSMITH, 87>.

Suivant le conseil de COPPERSMITH, nous passerons en revue les différentes méthodes d'attaques à l'encontre des fichiers avant d'étudier la manière de les protéger.

Dans **les chapitres 2 et 3**, nous étudierons les grands principes d'attaque contre un système d'authentification. Ils nous permettront de dégager les critères de sécurité auxquels devront répondre les fonctions d'authentification que nous analyserons dans la partie suivante.

Dans **le chapitre 4**, nous tenterons de déterminer si les différentes attaques proposées sont réalistes ou pas, et ce en fonction des moyens techniques et financiers dont peuvent disposer les fraudeurs et les défenseurs.

2. LES ATTAQUES EXHAUSTIVES ET LES ANALYSES STATISTIQUES

Les attaques exhaustives (ou attaques brutales) sont des attaques où, afin de découvrir une information, l'ennemi essaie toutes les combinaisons possibles que peut prendre cette information, et cela jusqu'au moment où une combinaison valable est trouvée. Ceci nécessite énormément de temps et de moyens de calcul. Une analyse statistique de la fonction d'authentification peut aider l'attaquant à accélérer ses recherches. Les attaques ne sont alors plus exhaustives, mais guidées, c'est-à-dire que certaines combinaisons sont préférées à d'autres.

2.1. QUELQUES NOTIONS DE MATHEMATIQUES

Pour la bonne compréhension de ce qui va suivre, quelques notions élémentaires de mathématiques sont nécessaires:

- Si un CAF possède une longueur de 3 bits, il existe $2^3 = 8$ CAF différents possibles (000, 001, 010, 011, 100, 101, 110 et 111). De manière générale, un CAF de m bits peut prendre 2^m valeurs différentes.

- Si une fonction d'authentification génère des CAF de m bits de telle sorte que chacun des CAF ait la même probabilité d'apparaître, cette probabilité est de 2^{-m}. Corolairement, avec cette même fonction d'authentification, il faudra essayer 2^m fichiers différents pour être sûr d'en trouver au moins un qui corresponde à un CAF particulier.

2.2. LES ATTAQUES EXHAUSTIVES

2.2.1. L'ATTAQUE EXHAUSTIVE SUR UNE PAIRE FICHIER-CAF

Soit une fonction d'authentification H qui associe à un fichier F de b bits un CAF de m bits, avec m significativement plus petit que b. Dans ces conditions,

"The CAF is (...) a many-to-one function of the data to be checked. For that reason, with only m bits available for checking purposes, there is always the chance that the correct (CAF) was generated by a lucky accident even if changes took place in the (file)"<JUENEMAN, 84>.

On peut toutefois prendre des dispositions pour rendre pratiquement impossible la découverte d'un tel fichier.

Faisons l'hypothèse que la fonction possède de très bonnes propriétés de générateur pseudo-aléatoire. Cela signifie que tous les CAF ont la même probabilité de correspondre à un fichier donné. Comme il existe 2^m CAF différents possibles, étant donné F, un ennemi devra essayer 2^m fichiers frauduleux pour espérer en trouver un (disons F') qui soit tel que H(F) = H(F') avec F ≠ F'.

Si m est fixé à 64, l'attaquant doit alors construire 2^{64} (18 milliards de milliards!) fichiers et, pour chacun, calculer le CAF. Si un calcul de ce type peut être fait en 1 ns (nanoseconde) - ce qui est beaucoup plus rapide que la plus performante fonction d'authentification connue - l'ensemble de l'attaque prendra plus de 580 ans ! A ce niveau, on peut affirmer que l'attaque est impossible.

Comme le fait remarquer AKL :

"a large value of m is desirable to increase the chances of fraud detection. On the other hand, for large values of m, H(F) is more expensive to compute and to store. This is a typical example of the classical tradeoff between security and efficiency" <AKL, 1984>.

C'est pourquoi, dans un système d'authentification pratique, 128 bits sont considérés comme la limite supérieure pour la taille du CAF. Avec un CAF de 64 bits, on garde une très bonne sécurité, et dans bon nombre de situations, un CAF de 32 bits peut suffire.

2.2.2. L'ATTAQUE EXHAUSTIVE SUR PLUSIEURS PAIRES FICHIER-CAF

Si plusieurs paires (disons M) fichier-CAF sont disponibles, la tâche de l'attaquant est plus facile. Il ne doit essayer que $(2^m) / M$ fichiers frauduleux F' pour trouver une correspondance H(F') = H(F) avec F étant l'un des M fichiers authentiques. Il remplace alors F par F' et la fraude n'est pas détectée.

Une solution à ce problème est décrite par <AKL, 1984>, et est attribuée à R.C. MERKLE. L'idée consiste à utiliser une fonction d'authentification différente pour chaque fichier. Ceci met l'ennemi devant deux alternatives :

- soit il ne s'occupe que d'une seule fonction, et n'attaque donc que l'unique paire fichier-CAF associée à cette fonction (les autres paires étant obtenues par une fonction

différente); il se retrouve alors dans le cas traité au paragraphe précédent.

- soit il s'attaque aux M paires, ce qui l'oblige à utiliser les M fonctions. En effet, pour vérifier si un de ses fichiers frauduleux correspond aux M CAF visés, il doit calculer les M paires fichiers frauduleux-CAF frauduleux à l'aide des M fonctions. Le fait de devoir traiter les (2^n) / M fichiers frauduleux lui demande donc ((2^n) / M * M =) 2^n applications d'une fonction d'authentification.

Quel que soit son choix, l'attaquant devra faire une recherche exhaustive comprenant 2^n essais.

2.2.3. L'ATTAQUE EXHAUSTIVE SUR LA CLE

Nous avons émis l'hypothèse que la clé était connue de l'ennemi. Dans le cas contraire, une attaque exhaustive peut être lancée pour déterminer cette clé.

De façon étonnante, on peut montrer que, si la clé possède k bits, il faudra parfois plus de 2^k tentatives pour la découvrir.

Prenons l'exemple du mode CBC ("Cipher Block Chaining") défini pour le DES (pour rappel, revoir le paragraphe 3.5.1. de la deuxième partie). Cette fonction d'authentification utilise une clé de 56 bits et produit un résultat de 64 bits. Le CAF en est déduit en conservant les b bits les plus significatifs.

Examinons la situation où l'ennemi connaît un fichier authentique et son CAF, ainsi que la fonction d'authentification.

CAS 1 : CAF de 64 bits (pas de troncature)

Si le résultat de la fonction d'authentification n'est pas tronqué, le CAF est constitué de 64 bits. L'ennemi applique au fichier authentique la fonction d'authentification avec les 2^{56} clés possibles. 2^{56} résultats de 64 bits sont obtenus. Si on fait l'hypothèse réaliste que ceux-ci sont aléatoirement distribués dans l'espace des résultats, il est presque certain qu'un et un seul résultat corresponde au CAF authentique. La clé utilisée pour générer ce résultat est donc la clé d'authentification.

CAS 2 : CAF de 32 bits (par troncature des 32 bits les moins significatifs)

Comme dans le premier cas, l'ennemi essaie toutes les clés et obtient ainsi 2^{56} résultats de 64 bits aléatoirement distribués. Malheureusement pour lui, parmi ces résultats, il y en a ($2^{56} * 2^{-32} =$) 2^{24} qui ont leurs 32 bits les plus significatifs identiques aux 32 bits du CAF authentique. L'attaquant est donc dans l'impossibilité de découvrir la clé d'authentification sans une paire authentique fichier-CAF supplémentaire.

Ceci nous montre que révéler 32 bits au lieu de 64 est intéressant pour se protéger contre un ennemi ne possédant pas la clé d'authentification. Par contre, si elle est en sa possession, raccourcir le CAF ne peut que lui faciliter la tâche. En effet, il ne doit plus alors effectuer que 2^{32} essais au lieu de 2^{64} pour réaliser une attaque exhaustive sur le CAF.

Puisque nous avons supposé que l'ennemi connaît tout du système d'authentification, nous abandonnerons l'idée d'une troncature du code d'authentification.

2.2.4. L'ATTAQUE EXHAUSTIVE GUIDEE

La section suivante traite des analyses statistiques. Celles-ci visent à découvrir les faiblesses de la fonction d'authentification comme par exemple une probabilité supérieure à la normale de rencontrer telle caractéristique dans le CAF si telle condition est remplie dans le fichier. Ces propriétés peuvent être utilisées pour guider une attaque exhaustive, c'est-à-dire pour diminuer le nombre théorique d'essais à réaliser.

2.3. LES ANALYSES STATISTIQUES

Les analyses statistiques du comportement des fonctions d'authentification constituent un domaine complexe qui nécessite énormément de temps et de moyens de calcul.

Le résultat d'une fonction d'authentification idéale doit apparaître comme étant totalement aléatoire, c'est-à-dire indépendant du fichier et de la clé utilisés en entrée. Cette caractéristique oblige l'ennemi à réaliser des attaques longues et coûteuses telles celles décrites ci-dessus.

Mais s'il existe des régularités dans le comportement de la fonction, le processus de recherche exhaustif de l'ennemi peut être considérablement accéléré.

Supposons par exemple qu'une fonction génère un CAF de m

bits ayant une majorité de bits égaux à '1' quand les 64 derniers bits du fichier à authentifier sont égaux à '0'. Dans ce cas, la recherche d'un fichier frauduleux correspond à un CAF donné ayant la particularité d'avoir une majorité de bits égaux à '1' ne nécessitera pas 2^n essais. En choisissant des fichiers frauduleux ayant leurs 64 derniers bits à '0', l'ennemi peut espérer en trouver un bon assez rapidement.

Ci-dessous, nous donnons quelques-uns des comportements à vérifier:

- La génération des CAF est-elle aléatoirement distribuée? (ou en d'autres termes, certains CAF apparaissent-ils plus souvent que d'autres?)

- Les CAF sont-ils indépendants de la taille et/ou du contenu des fichiers en entrée.

- Les CAF sont-ils indépendants des clés utilisées?

- ...

Pour répondre à ces questions, il existe de nombreuses techniques d'analyse parmi lesquelles:

- Le test de fréquence (frequency test) qui est utilisé pour vérifier que le nombre de 0 et de 1 dans le CAF est plus ou moins le même quels que soient les fichiers et les clés en entrées

- Le test séquentiel (serial test) qui est utilisé pour vérifier que chaque bit du CAF est indépendant du bit qui le précède

Nous ne pouvons aller plus loin dans ce domaine sans introduire de nombreuses notions mathématiques relativement complexes. Nous invitons toutefois le lecteur intéressé par de plus amples renseignements à consulter <ISO, 85> et <BEKER, 82, 169-174>.

3. LES ATTAQUES SUR LES BLOCS

Les attaques décrites ci-dessous ont toutes pour particularité d'être axées sur la façon dont les fonctions d'authentification traitent le chaînage des blocs de fichiers. La fonction de chiffrement d'un bloc est ici considérée comme une 'boite noire'.

Rappelons que par hypothèse, l'ennemi connaît à la fois la fonction d'authentification et la clé. Les notations qui seront employées sont celles introduites dans la partie précédente et résumées en fin de ce mémoire sous le titre 'notations et abréviations'. Nous ne ferons pas mention de la clé; le chiffrement du bloc X par la fonction E s'exprimera par E (X) et le déchiffrement par D (X). Le mode d'opération CBC défini pour le DES, introduit au paragraphe 3.5.1. de la précédente partie, sera employé à titre d'illustration.

3.1. L'INSERTION, LA SUPPRESSION ET LA MODIFICATION

F est un fichier authentique. Ce fichier est constitué de n blocs de t bits : F1, F2, ..., Fn.

H est une fonction d'authentification. Faisons l'hypothèse que dans tout ce qui va suivre, H utilise toujours la même clé. En employant cette clé, H génère un CAF à partir du fichier F.

Un ennemi peut désirer :

- soit insérer un ou plusieurs blocs frauduleux de longueur t parmi les n blocs authentiques,
- soit supprimer ou modifier un ou plusieurs blocs authentiques,
- soit faire une combinaison de ces opérations.

Le problème qu'il doit résoudre est toujours le même, à savoir : comment faire pour que le fichier frauduleux obtenu conserve un CAF identique à celui du fichier authentique? En d'autres termes, comment doit-il manipuler judicieusement F afin d'obtenir F' de telle manière que H(F) = H(F')?

Il n'existe pas de solution 'universelle' à ce problème. Il y a même des cas (nous en verrons dans la quatrième partie) où ce type de manipulation n'est pas faisable. Nous avons donc choisi d'illustrer ces attaques en utilisant un algorithme qui s'y prête particulièrement bien: le mode d'opération CBC. Ceci nous donnera l'occasion de parfaire notre connaissance des mécanismes de base de l'authentification.

3.1.1. L'INSERTION D'UN BLOC FRAUDULEUX

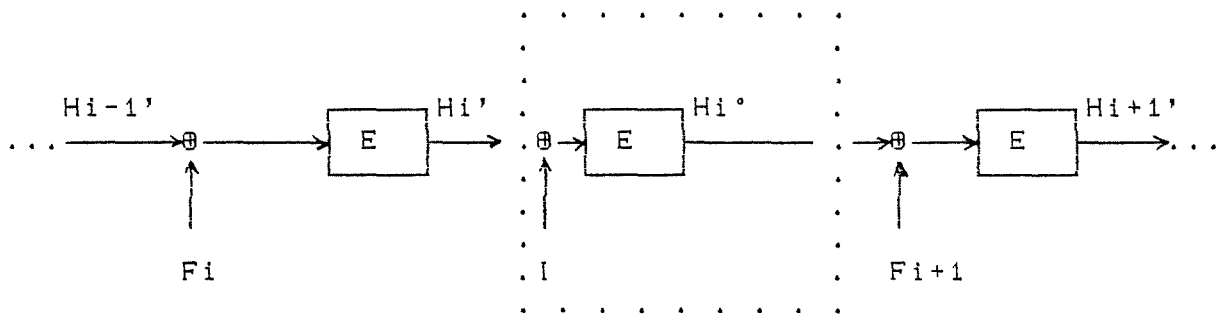
Nous examinerons ci-dessous comment un bloc frauduleux I peut être inséré entre deux blocs authentiques. Nous généraliserons ensuite cette technique à l'insertion de plusieurs blocs.

L'insertion du bloc I entre les blocs Fi et Fi+1 peut être illustrée par la figure 3.1.

FIGURE 3.1 : Insertion de I entre Fi et Fi+1

AVANT

APRES



On avait avant (Fichier F)

$$Hi = E(Hi-1 \oplus Fi),$$
$$Hi+1 = E(Hi \oplus Fi+1). \quad (1)$$

On obtient maintenant (Fichier F')

$$Hi' = E(Hi-1' \oplus Fi),$$
$$Hi° = E(Hi' \oplus I), \quad (2)$$
$$Hi+1' = E(Hi° \oplus Fi+1). \quad (3)$$

Deux cas peuvent être envisagés. Le premier correspond à la situation où le bloc I est choisi pour sa signification frauduleuse et sans se soucier de la fonction d'authentification utilisée. Nous

parlerons alors de **manipulation choisie**. Le deuxième cas est celui où le bloc I est choisi non pas pour sa sémantique, mais parce qu'il permet de déjouer la fonction d'authentification. Puisque le choix de l'ennemi lui est dicté par les circonstances et non par son bon vouloir, nous parlerons de **manipulation imposée**.

Nous allons maintenant étudier ces deux cas en commençant par la manipulation choisie. Avec cette hypothèse, nous pouvons tenir le raisonnement suivant:

- Puisque rien n'a changé dans le fichier avant le bloc Fi, on a Hi-1' = Hi-1 et donc aussi Hi' = Hi (4).

- Si nous examinons la figure 3.1., nous nous apercevons que l'insertion de I peut être représentée par l'ajout du carré pointillé. Ce carré a pour conséquence de produire, avant que n'intervienne le bloc Fi+1, un résultat Hi°. Celui-ci est calculé par (2) à partir de Hi' et de I. Puisque I est choisi sans précaution particulière, Hi° est certainement différent de Hi'. En appliquant (4), nous obtenons :

  $$Hi^{\circ} \neq Hi.$$

  A partir de là, le phénomène de propagation des erreurs (introduit à la section 3.3. de la partie précédente) va entrer en jeu: puisque Hi° ≠ Hi, on a également Hi+1' ≠ Hi+1, Hi+2' ≠ Hi+2, et ainsi de suite jusqu'à Hn' =/ Hn.

- Si le CAF est défini comme étant égal au dernier résultat intermédiaire, puisque Hn' ≠ Hn, nous avons CAF' ≠ CAF. Autrement dit, en choisissant d'insérer un bloc frauduleux sans prendre aucune précaution, cette modification du fichier original est détectée (CAF ≠ CAF').

Après avoir étudié le cas d'une modification choisie, nous allons examiner celui d'une manipulation imposée. Notre nouvelle hypothèse est donc:

Le bloc I à insérer est choisi sur base de la fonction d'authentification employée.

Nous pouvons alors tenir le raisonnement suivant :

- L'objectif de l'ennemi est d'obtenir CAF' = CAF, ce qui équivaut à obtenir Hn' = Hn.

- Puisque Fi+1, Fi+2, ..., Fn ne sont pas modifiés, pour obtenir Hn' = Hn, il suffit d'avoir Hi+1' = Hi+1.

- Si nous nous référons aux équations (1) et (3), Hi+1 = Hi+1' équivaut à Hi = Hi'.

- Cette dernière égalité se traduit par :

$$E(Hi-1 \oplus Fi) = E(Hi' \oplus I),$$

ou encore :

$$Hi-1 \oplus Fi = Hi' \oplus I,$$

ce qui revient à prendre :

$$I = Hi-1 \oplus Fi \oplus Hi'.$$

En se référant à l'équation (4) (spécifiant que Hi' = Hi), nous obtenons une formule qui permet de construire un bloc I pouvant être inséré entre n'importe quels blocs successifs Fi et Fi+1 d'un fichier protégé par le mode CBC :

FORMULE 3.1.

$$I = Hi-1 \oplus Fi \oplus Hi$$

Ce bloc frauduleux sera sans conséquence sur le CAF. Le fichier frauduleux obtenu par insertion de I ne sera donc pas détecté par le système d'authentification.

On peut constater que I est formé par un OU-exclusif entre Hi-1 (le résultat intermédiaire après le traitement du bloc Fi-1), Hi (le résultat après le traitement du bloc Fi) et le bloc Fi. Si Fi peut avoir une signification, il n'en est rien de Hi et Hi-1. Ces deux valeurs peuvent être considérées comme résultat d'une fonction pseudo-aléatoire. I n'a donc certainement aucune sémantique. Nous verrons dans le chapitre 4 de cette partie si la menace de l'insertion d'un bloc sans signification est grave ou simplement gênante.

3.1.2. L'INSERTION DE PLUSIEURS BLOCS FRAUDULEUX

Nous avons vu à la section précédente qu'un ennemi

désirant insérer un bloc frauduleux parmi les blocs authentiques d'un fichier protégé par le mode CBC est placé devant une alternative :

- ou bien il choisit un bloc à la sémantique frauduleuse, mais alors ce bloc est automatiquement détecté puisque son insertion provoque une modification du CAF,

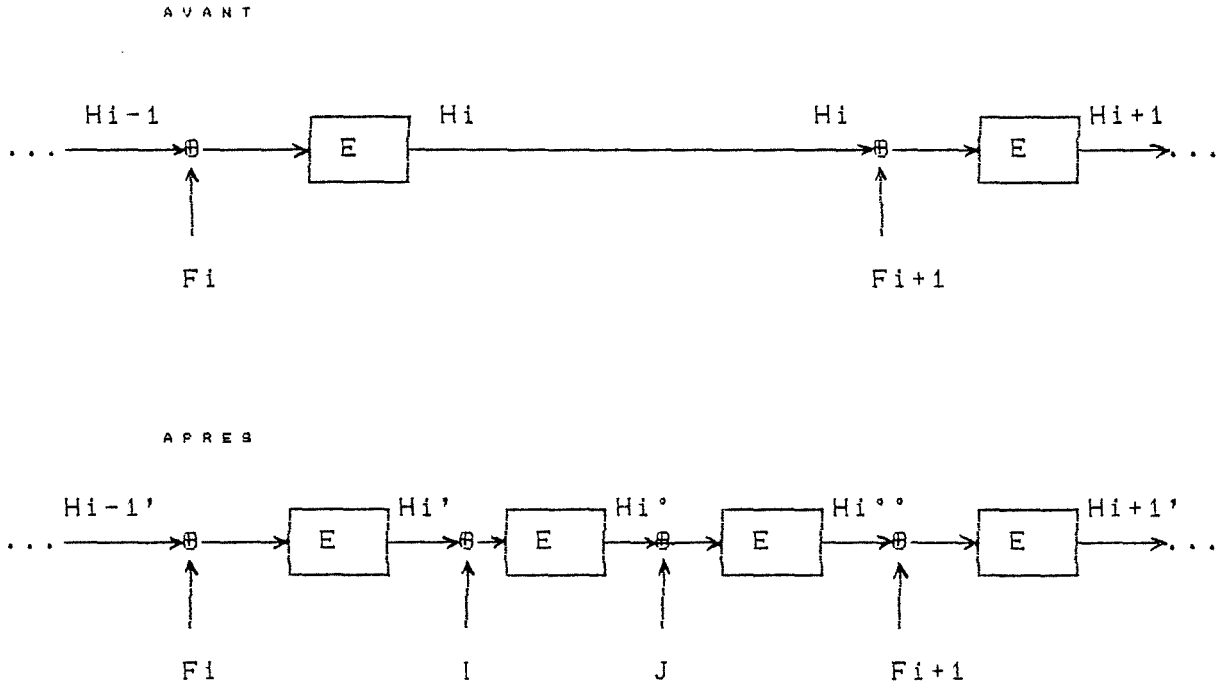
- ou bien il choisit un bloc qui n'affecte pas le CAF, mais alors ce bloc n'a aucune signification.

Cet ennemi peut également désirer insérer, non pas un seul, mais plusieurs blocs frauduleux. La question est maintenant de savoir si cette insertion est également liée à l'alternative décrite ci-dessus. La réponse est non.

Il est en effet possible à l'attaquant d'insérer un ou plusieurs blocs significatifs. Mais afin d'éviter la détection, un autre bloc, imposé cette fois, est placé à leur suite. Ce bloc sans sémantique a pour effet d'annuler les erreurs introduites dans les résultats intermédiaires par les insertions qui le précédent.

Afin de rester simple, nous envisagerons l'insertion d'un bloc choisi, suivie de l'insertion d'un bloc imposé. Ceci peut être illustré par la figure 3.2.

FIGURE 3.2 : insertion de I choisi et de J imposé.



I et J ont chacun un objectif bien différent :

- celui de I est d'opérer une fonction frauduleuse quelconque,

- celui de J est de compenser l'erreur introduite par I dans la suite de résultats intermédiaires.

Comment construire un tel J ? Dans le cas du mode CBC, c'est très simple :

- Il est clair que J doit être tel que :

$$Hi^{\circ\circ} = Hi.$$

- J doit donc être choisi de telle sorte que :

$$E(Hi\text{-}1 \oplus Fi) = E(Hi^{\circ} \oplus J).$$

- Pour cela il suffit de prendre :

$$Hi\text{-}1 \oplus Fi = Hi^{\circ} \oplus J,$$

ou encore

FORMULE 3.2.

$$J = Hi^{\circ} \oplus Fi \oplus Hi\text{-}1.$$

On peut commenter cette formule comme suit :

Si le bloc J doit être introduit avant le bloc Fi+1, il doit être composé d'un OU-exclusif entre

- le résultat intermédiaire qui précède J,

- le bloc Fi, et

- le résultat intermédiaire qui précéde Fi.

La formule 3.2. reste bien entendu valable pour l'insertion de plus d'un bloc choisi. Il est intéressant de noter que dans le cas particulier où aucun bloc choisi n'est inséré, nous retrouvons la formule 3.1.

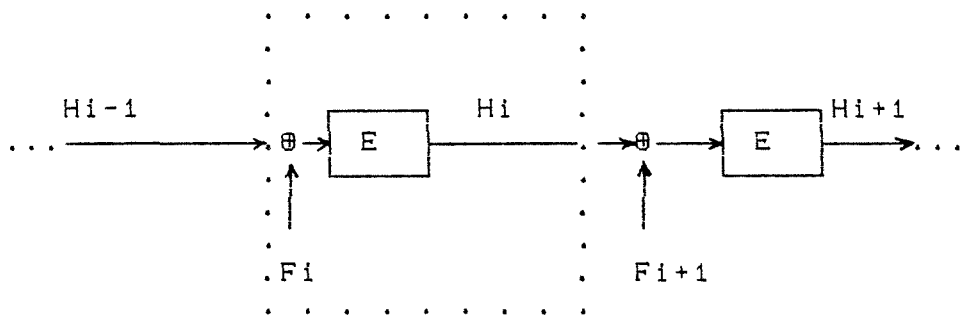
3.1.3. SUPPRESSION D'UN BLOC AUTHENTIQUE

Examinons maintenant quelle peut être la conséquence

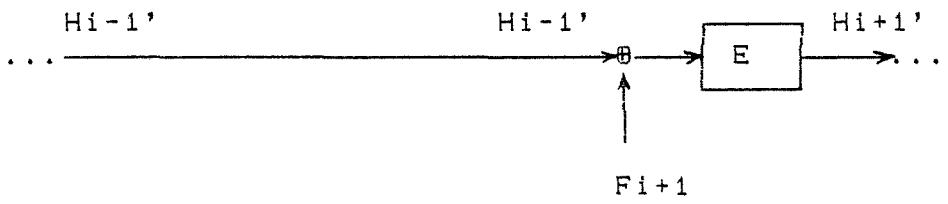
de la suppression d'un bloc (Fi par exemple) appartenant au fichier authentique F. La suppression de plusieurs blocs s'obtient facilement en généralisant cette technique. Cette suppression peut être illustrée par la figure 3.3.

FIGURE 3.3 : Suppression de Fi.

AVANT



APRES



Nous n'avons pas besoin de tenir un raisonnement rigoureux pour découvrir la condition nécessaire à la non-détection de cette manipulation. Examinons tout simplement la figure 3.3, et plus particulièrement le carré pointillé. La suppression de Fi correspond à la disparition de ce carré. Pour que ceci n'ait pas de conséquence sur Hi+1, et dès lors laisse le CAF inchangé, il est nécessaire d'avoir :

$$Hi = Hi-1.$$

Malheureusement (pour l'attaquant), il est illusoire de croire que dans un système réel, deux résultats intermédiaires successifs puissent être identiques. En effet, si les blocs d'un fichier ne sont pas choisis de façon spécifique, les résultats intermédiaires sont indépendants les uns des autres. A titre d'exemple, si le DES est utilisé pour la fonction de chiffrement E, une telle coïncidence

peut avoir lieu avec une probabilité de 2^{-64}, ce qui correspond en fait à une impossibilité.

Alors, doit-on en conclure qu'il est impossible d'enlever un bloc authentique sans que cette manipulation ne soit décelée? Bien sûr que non! Mais pour éviter la détection, la suppression doit être conjuguée à une insertion imposée satisfaisant la formule 3.2.

A titre d'exemple, examinons le cas de la suppression des blocs authentiques Fi-1 et Fi. Un bloc J est inséré à leur place. Puisque Fi-1 et Fi sont supprimés, le résultat intermédiaire qui précède J est celui généré par Fi-2 c'est-à-dire Hi-2.

La formule 3.2. impose donc à l'ennemi de choisir J = Hi-2 ⊕ Fi ⊕ Hi-1. Le fichier frauduleux F' constitué des blocs :

F1, F2,..., Fi-2, J, Fi+1,..., Fn,

possède le même CAF que le fichier authentique F et n'est donc pas détecté.

3.1.4. LA MODIFICATION D'UN OU PLUSIEURS BLOCS AUTHENTIQUES

Dans les trois derniers paragraphes, nous avons examiné l'insertion et la suppression d'un ou plusieurs blocs. Or il est clair que modifier un bloc est équivalent à sa suppression suivie par l'insertion de sa nouvelle version. Ce concept de modification est toutefois intéressant pour l'attaquant car une des précautions élémentaires qui est prise par un concepteur de système d'authentification est la vérification de la longueur d'un fichier à protéger.

Essayons de trouver comment une modification (choisie!) peut-être perpétrée sur un bloc du fichier F. La modification de plusieurs blocs s'obtient facilement en généralisant cette technique.

A titre d'exemple, examinons le cas de la modification de Fi-1 en Fi-1'. Par un raisonnement semblable à ceux tenus dans les paragraphes précédents, nous pouvons montrer que si Fi n'est pas modifié de façon adéquate, le CAF du nouveau fichier sera différent du CAF du fichier original.

En appliquant la formule 3.2., nous obtenons la nouvelle valeur (Fi') imposée à Fi, à savoir :

Fi' = Hi-1' ⊕ Fi ⊕ Hi-1.

Donc, si en plus de la modification 'choisie' sur Fi-1, l'ennemi perpétue la modification 'imposée' sur Fi, le CAF du fichier frauduleux ainsi obtenu restera identique au CAF du fichier original.

Un cas particulier de la modification de plusieurs blocs d'un fichier est le remplacement de ce fichier par un autre entièrement choisi par l'ennemi. En fait, 'entièrement' n'est pas tout à fait correct car, pour éviter la détection, le dernier bloc de ce fichier frauduleux doit satisfaire la formule 3.2. Il est en conséquence certainement sans signification.

3.2. L'ATTAQUE " WORKING-BACKWARDS "

Nous avons vu au paragraphe précédent une technique d'attaque qui, dans le cas du mode d'opération CBC, permet de construire un fichier frauduleux F' tel que, étant donné un fichier authentique F, on ait H(F') = H(F). Cette technique autorise le fraudeur à donner à F' la forme qu'il désire, sauf en ce qui concerne le dernier bloc qui lui, est imposé et sans signification.

Il existe une technique semblable mais qui place le bloc imposé en début de fichier. Cette technique travaille à partir du CAF visé, et calcule la suite des résultats intermédiaires 'à reculons' : Hn, Hn-1, ..., H2, H1. Le premier bloc est alors calculé de façon à produire le premier résultat intermédiaire H1. Ceci résume le principe de l'attaque "working-backwards" que nous allons détailler dans le cas du mode CBC défini pour le DES.

- Supposons que le système d'authentification contienne la paire F - H(F) où :

  * F est un fichier authentique constitué de n blocs F1, F2, ..., Fn,

  * H(F) est le CAF obtenu par application du mode CBC au fichier F.

  L'ennemi désire trouver un fichier frauduleux F', entièrement choisi par lui (sauf le premier bloc F1') tel que H(F') = H(F).

- Il construit F' (F1', F2', ..., Fn') sans se soucier du premier bloc. Ce fichier contient le même nombre de blocs que F pour éviter une détection par vérification de la longueur. Si cette détection n'est pas implémentée par le système d'authentification, la taille de F' peut être quelconque.

- Quoiqu'il arrive, il faut que le dernier résultat intermédiaire soit égal au CAF du fichier authentique. On a donc :

$Hn' = H(F)$.

- Le bloc Fn' est alors examiné. Pour qu'il génére le dernier résultat intermédiaire attendu (Hn'), il faut que l'équation suivante soit vérifiée :

  $Hn' = E\ (Hn\text{-}1' \oplus Fn')$.

  Puisque Hn' et Fn' sont fixés, il faut que :

  $D\ (Hn') = D\ (E\ (Hn\text{-}1' \oplus Fn'))$,

  $D\ (Hn') = Hn\text{-}1' \oplus Fn'$,

  ou encore

  $Hn\text{-}1' = D\ (Hn') \oplus Fn'$.

- En continuant le raisonnement 'à reculons' , on calcule la suite des résultats intermédiaires Hn-2', Hn-3', ... H1'. Ce H1' exprime la valeur que doit prendre le premier résultat intermédiaire pour qu'avec les blocs F2',...Fn', le CAF obtenu soit égal à H(F).

- L'ennemi calcule alors la valeur que doit prendre F1' pour que H1' soit généré. Il obtient facilement cette valeur (sans aucune signification) par

  $H1' = E\ (IV \oplus F1')$,

  et donc

  $F1' = D\ (H1') \oplus IV$.

Une fois de plus, il est important de rappeler que ce résultat est celui obtenu par étude du mode CBC, et qu'il ne peut en aucun cas être généralisé aux autres fonctions d'authentification. La plupart des fonctions qui seront analysées dans la quatrième partie ne sont d'ailleurs pas attaquables de la sorte.

3.3. L'ATTAQUE "MEET-IN-THE-MIDDLE"

Aux sections 3.1 et 3.2 de ce chapitre, toute manipulation des blocs d'un fichier authentique imposait à l'ennemi l'usage d'un bloc sans aucune sémantique. Nous nous poserons plus tard la question de savoir si un fichier frauduleux contenant un tel bloc est réellement dangereux. Mais sans entrer dans un grand raisonnement, il paraît évident que cette présence est gênante pour l'attaquant.

Nous avons déjà vu comment, par recherche exhaustive, le fraudeur pouvait trouver un fichier entièrement choisi par lui-même. Toutefois, le prix à payer pour ce fichier à la sémantique 100% frauduleuse est bien lourd, à savoir un nombre astronomique d'essais. Une première solution a été

proposée sous la forme d'analyses statistiques qui peuvent diminuer le nombre de tentatives infructueuses. Toutefois, cette solution se base sur le fait que la fonction d'authentification possède des faiblesses (les CAF non aléatoirement distribués, régularités internes, etc...), ce qui n'est pas toujours le cas.

Cela ne signifie pas pour autant qu'un ennemi désirant trouver un fichier entièrement choisi qui corresponde à un CAF donné doive en essayer 2^n. Une attaque connue sous le nom de "meet-in-the-middle" va lui servir à trouver un tel fichier avec seulement $2^{n/2}$ essais.

Afin de mieux situer le gain réalisé, il suffit de calculer le temps des deux attaques pour m = 64. Si on suppose qu'un essai (c'est-à-dire le calcul du CAF d'un fichier essayé) prend une microseconde, l'attaque exhaustive prendra 5849 siècles, tandis que l'attaque "meet-in-the-middle" ne prendra qu'une heure et douze minutes. Ce qui était impossible passe maintenant dans le domaine du possible.

Avant de décrire l'attaque proprement dite, nous exposerons sa base théorique. Celle-ci peut se résumer par le paradoxe de l'anniversaire : un problème classique de la probabilité qui possède un résultat aussi intéressant qu'inattendu.

3.3.1. LE PARADOXE DE L'ANNIVERSAIRE

Le paradoxe de l'anniversaire, aussi appelé problème de l'anniversaire, s'énonce ainsi :

Combien doit-il y avoir de personnes dans un local pour qu'il soit plus que probable qu'au moins deux d'entre elles possèdent la même date d'anniversaire?

'pour qu'il soit plus que probable' doit ici se comprendre par 'pour qu'il y ait une probabilité supérieure à 0,5'.

Si on fait l'hypothèse que les dates d'anniversaires sont choisies aléatoirement parmi 365 possibilités, la réponse est 23. Ce résultat peut paraître très petit à première vue, d'où l'appellation de paradoxe.

La formule donnant la solution exacte de ce problème n'a que peu d'intérêt. Il est beaucoup plus important d'avoir l'intuition du mécanisme sous-jacent. Pour cela, nous allons chercher une formule approchant la solution exacte. Ces deux formules ainsi que le raisonnement qui va suivre sont extraits de <DAVIES, 84, 116-117>.

Reformulons le problème en termes plus généraux.

Des nombres sont choisis aléatoirement parmi les valeurs 1, 2, ..., n. Après le choix de r de ces nombres, quelle est la probabilité que deux ou plus d'entre eux soient égaux? Nous pouvons donner une approximation, valable quand r^2 est petit par rapport à n, en nous demandant combien de comparaisons peuvent être faites parmi les r nombres. Il y a r(r-1) / 2 manières différentes de sélectionner et de comparer deux nombres parmi les r choisis. Chaque comparaison a une probabilité 1 / n de donner une égalité. La probabilité d'une égalité parmi les r nombres choisis est donc approximativement r(r-1) / 2n. Ceci montre que nous devons comparer n avec r^2, et non r, ce qui aide à disciper le paradoxe apparent.

Ce paradoxe de l'anniversaire peut être utilisé pour de nombreuses applications, et notamment pour mettre au point une attaque connue sous le nom de "meet-in-the-middle".

3.3.2. L'ATTAQUE

Il existe plusieurs variantes de l'attaque "meet-in-the-middle" (<COPPERSMITH, 86>, <AKL, 84>). Leur point commun est qu'elles exploitent toutes le paradoxe de l'anniversaire pour rechercher des coïncidences entre deux ensembles plutôt que de s'intéresser à des valeurs spécifiques. Afin de rendre le raisonnement plus clair, nous illustrerons une fois de plus l'attaque par le mode CBC défini pour le DES.

- Supposons que le système d'authentification contienne la paire F - H(F) et qu'un ennemi désire trouver un fichier frauduleux F', entièrement choisi par lui, tel que H(F') = H(F).

- Il construit deux tables contenant chacune 2^{32} demi-fichiers frauduleux. La table 1 contient des éléments du type

  F1'« F2'«...« Fj' (où j est proche de n / 2),

  tandis que la table 2 contient des éléments du type :

  Fj+1'« Fj+2'« ...« Fn'.

  Ces deux tables doivent être telles qu'en concaténant n'importe quel élément de la première et n'importe quel élément de la seconde, un fichier frauduleux intéressant pour

l'ennemi soit obtenu.

- Pour chacun des éléments de la table 1, l'ennemi calcule le dernier résultat intermédiaire Hm :

$$H1 = E(VI \oplus F1'),$$

$$H2 = E(H1 \oplus F2'),$$

$$\vdots$$

$$Hj = E(Hj-1 \oplus Fj').$$

Ces 2^{32} valeurs Hj (une valeur par demi-fichier) sont triées et stockées.

- Pour chacun des éléments de la table 2, il calcule la valeur initiale (Hj°) qui donnera le CAF visé. Pour cela, il est obligé de travailler 'à reculons ':

$$Hn = \text{CAF visé} = H(F),$$

$$Hn-1 = D(Hn) \oplus Fn',$$

$$\vdots$$

$$Hj^\circ = D(Hj+1) \oplus Fj+1'.$$

- Chacune des 2^{32} valeurs Hj° ainsi obtenue est comparée à la liste triée de Hj. Nous trouvons ici les conditions de base pour qu'intervienne le paradoxe de l'anniversaire: au lieu de rechercher une valeur spécifique (le CAF d'un fichier présent dans le système d'authentification), il est plus intéressant de chercher une coïncidence entre les valeurs de deux ensembles de beaucoup plus petite taille.

  Dans les termes employés au paragraphe précédent, nous avons $n=2^{64}$ et $r=2^{32}$, ce qui nous donne une probabilité de coïncidence d'environ ($r^2/2n$ =) 0,5. Dans <MEYER, 88>, il est montré que la probabilité exacte est de 0,63.

  Les deux demi-fichiers pour lesquels ces Hm et Hm° ont été générés sont alors mis bout à bout pour former un fichier F' (F1'«... Fj'« Fj+1'« ...« Fn') tel que :

$$H(F') = H(F).$$

4. UNE EVALUATION DE FAISABILITE ET DE RENTABILITE

Dans les chapitres 2 et 3 de cette partie, nous avons décrit bon nombre d'attaques. Il est légitime de se poser la question de savoir si celles-ci sont à considérer comme des exercices de style pour mathématiciens, ou si elles représentent un véritable danger pour la sécurité des données.

Nous nous proposons dans ce chapitre d'évaluer leur faisabilité et leur rentabilité. En effet, dans le cas où elles sont techniquement impossibles, elles ne représentent bien entendu aucun danger. De même, si elles ne rapportent pas plus à l'ennemi qu'elles ne lui coûtent, nous pouvons également considérer que le risque est faible.

4.1. LA FAISABILITE DES ATTAQUES

Il est souvent bien difficile de dire si une attaque est possible ou pas. Cela dépend des moyens techniques dont dispose l'ennemi et du niveau de performance de cette technologie. Or, ce niveau est en continuelle progression. C'est pourquoi nous avons pris le parti de baser notre analyse sur l'affirmation suivante :

"The algorithms used must be able to successfully resist cryptanalysis on today's most powerful super computers by the most knowledgeable experts, or on whichever super computers are likely to be available during the expected life times of the (...) algorithms" <FERREIRA, NP>.

Nous ne prendrons pas en compte ici l'aspect financier des choses. Celui-ci sera traité dans la section suivante.

Une fois de plus, le mode CBC défini pour le DES sera pris comme référence.

4.1.1. UNE QUANTIFICATION DES ELEMENTS INTERVENANTS DANS L'EVALUATION

En ce qui concerne les performances du mode CBC, nous pouvons citer comme références <FERREIRA, NP> :

- 1,25 Mo/s pour une version programmée tournant sur un CRAY 1.
- 11 Mo/s pour la plus rapide version matérielle connue.

Nous utiliserons, pour la présente évaluation, une estimation optimiste de la vitesse du mode CBC sur un CRAY XX, sans doute l'ordinateur le plus rapide prévu pour les années 1990. En supposant que de gros progrès soient réalisés dans l'optimisation

du DES, nous pouvons nous attendre à une vitesse de **25 Go/s <FERREIRA, NP>**. Dans ces conditions, un bloc de 64 bits est traité en **0.32 ns** (nanosecondes).

Nous avons choisi de ne pas prendre en considération les différents temps d'accès aux moyens de stockage. Le lecteur intéressé par une telle analyse peut consulter **<JUENEMAN, 87>**. Quant au problème du volume mémoire nécessaire pour les différentes attaques, nous le traiterons au point 4.1.5. ci-après.

4.1.2. LA FAISABILITE D'UNE ATTAQUE EXHAUSTIVE

Nous avons montré que dans sa forme la plus simple, une attaque exhaustive sur une paire fichier-CAF nécessitait la construction de 2^m fichiers et, pour chacun d'eux, le calcul du CAF (m étant la longueur en nombre de bits du CAF). Si nous considérons des fichiers frauduleux d'un Ko, le calcul d'un CAF prendra 40 ns, et la recherche dans son entièreté nécessitera environ:

130 jours pour m=48

91 ans pour m=56

L'attaque semble donc être impossible pour m>50.

Il existe toutefois un moyen d'accélérer ce processus. Imaginons en effet que les fichiers frauduleux construits par l'ennemi soient tous identiques, sauf en ce qui concerne les 128 derniers bits. En modifiant ceux-ci, il peut construire facilement jusqu'à 2^{128} fichiers différents. Il suffit alors de calculer une fois pour toutes le résultat intermédiaire du mode CBC avant ces deux derniers blocs, et de le considérer comme la valeur initiale VI dans le calcul du CAF d'un fichier de 128 bits. Le calcul d'un CAF prendra alors (2 * 0,32 =)0,64 ns contre 40 ns précédemment. Dans ces conditions, une recherche exhaustive nécessitera environ:

50 heures pour m=48

533 jours pour m=56

374 ans pour m=64

L'attaque semble donc impossible pour m>56.

Il semble donc que m = 56 soit une limite de sécurité acceptable. Toutefois, ce résultat nécessite quelques nuances:

- Il faut toujours garder à l'esprit que le temps d'une attaque exhaustive doit être réduit en proportion du nombre de paires fichier-CAF disponibles pour l'ennemi (cfr à cet effet le point 3.1.2. de la présente partie).

- Nous n'avons pas traité le cas de l'attaque exhaustive guidée. Le gain de performances dû à la découverte de certaines régularités dans une fonction d'authentification est en effet difficilement quantifiable. Ce phénomène doit toutefois être analysé avec toute l'attention qu'il mérite.

En conclusion, nous nous rangerons à un résultat largement accepté dans la littérature (<AKL, 84>, <CAMPANA, 88>) à savoir que pour résister à une attaque exhaustive, un CAF doit avoir une longueur minimale de 64 bits.

4.1.3. LA FAISABILITE DES ATTAQUES PAR MANIPULATION DE BLOCS

Nous traitons ici des attaques décrites aux points 3.1. et 3.2. de la présente partie.

Contrairement aux énumérations exhaustives, la faisabilité de ces attaques est conditionnée par la structure de la fonction d'authentification employée. Nous avons montré comment elles pouvaient être très facilement réalisées pour le mode CBC, mais nous rencontrerons dans la partie suivante des fonctions pour lesquelles elles sont irréalisables.

Quoiqu'il en soit, si la structure de la fonction d'authentification est telle que ces attaques soient possibles, leur réalisation ne pose aucune difficulté particulière.

4.1.4. LA FAISABILITE D'UNE ATTAQUE "MEET-IN-THE-MIDDLE"

Nous traitons ici de l'attaque "Meet-in-the-middle" décrite au point 3.3. de la présente partie.

Comme ci-dessus, la faisabilité de l'attaque dépend de la structure de la fonction d'authentification utilisée. Nous rencontrerons dans la partie suivante des fonctions pour lesquelles elles sont irréalisables. Dans la suite de ce paragraphe, nous travaillerons avec le mode CBC qui est sensible à une telle attaque.

Une attaque "Meet-in-the-middle" nécessite la construction de deux fois 2^{32} demi-fichiers, le

calcul de deux fois 2^{32} résultats intermédiaires et le tri de 2^{32} valeurs de 64 bits.

- Si nous considérons des fichiers frauduleux de 1 Ko, le calcul des résultats intermédiaires nécessitera environ 3 minutes. Ce temps est directement fonction de la taille des fichiers manipulés. Une astuce très semblable à celle décrite en 4.1.2. et que l'on peut trouver notamment dans <COPPERSMITH, 86> permet de réduire ce temps de recherche à 3 secondes et ce, quelle que soit la taille des fichiers frauduleux.

  Rappelons que ces temps ne prennent pas en compte les accès aux mémoires secondaires. Cela nécessite de multiplier les résultats obtenus par un facteur 2 ou 3 <JUENEMAN, 87>. Quoi qu'il en soit, ces valeurs restent nettement dans le domaine du faisable.

- Dans <DAVIES, 84, 278>, on montre que le tri de 2^{32} valeurs nécessite de l'ordre de 2^{38} étapes de calcul. Sur un CRAY XX qui pourra réaliser près de 30 milliards d'opérations à la seconde, ce travail ne devrait demander que quelques minutes.

Nous sommes donc obligés d'affirmer qu'une attaque "meet-in-the-middle" est techniquement réalisable si les résultats intermédiaires ont une taille de 64 bits ou moins. Par contre, tout danger semble écarté si les résultats intermédiaires ont 128 bits ou plus. Cette affirmation est généralement celle adoptée dans la littérature (<JUENEMAN, 87>, <CAMPANA, 88>, <MEYER,88>).

4.1.5. COMMENT CONSTRUIRE ET STOCKER UN GRAND NOMBRE DE FICHIERS FRAUDULEUX?

L'attaque "meet-in-the-middle" nécessite 2^{32} fichiers frauduleux. L'attaque exhaustive en demande 2^{64}. Nous examinerons ci-dessous s'il est possible de construire et de stocker une telle quantité de fichiers.

L'astuce décrite précedemment au point 4.1.2. nous donne un moyen de construire sans trop de difficultés jusqu'à 2^{218} fichiers différents. Remarquons toutefois que les fichiers ainsi formés possèdent 128 bits générés aléatoirement, et donc sans aucune signification.

Une technique décrite dans <DAVIES, 84, 278> permet de générer systématiquement de très grandes quantités de variations entièrement significatives

d'un même fichier. Cette méthode est basée sur des modifications de syntaxe, des permutations, des utilisations de synonymes, ... A titre d'exemple, on trouvera en annexe 3 un texte qui peut servir de base à la construction de 2^{37} messages frauduleux. Les messages ne doivent pas être stockés, un mot de 37 bits étant suffisant pour identifier et donc reconstruire une des 2^{37} variantes. Cette technique est facilement généralisable à tous les types de fichiers.

Pour l'attaque exhaustive, les fichiers ne doivent pas être stockés. On en crée un et on calcule son CAF. S'il ne convient pas, on l' abandonne et on passe à un autre.

L'attaque "meet-in-the-middle" nécessite la construction d'une table de 2^{32} éléments; chacun d'entre eux étant composé des 64 bits de résultat intermédiaire et des 32 bits identifiant le demi-fichier auquel correspond ce résultat. Le volume de stockage nécessaire est de 51,5 Go, ce qui est faisable au vu de la technologie actuelle.

4.1.6. LES BLOCS SANS AUCUNE SIGNIFICATION

Dans ce mémoire, nous ne nous intéresserons pas à la sémantique des fichiers frauduleux. Nous nous contenterons d'évoquer un phénomène qui apparaît dans les attaques décrites aux points 3.1. et 3.2. de la présente partie, à savoir l'obligation pour les fraudeurs de placer un bloc de 64 bits sans aucune signification dans leurs fichiers frauduleux.

Nous ne croyons pas que cette contrainte puisse empêcher un ennemi de perpétrer ces attaques. En effet, la construction d'un programme (ou d'un fichier de données) frauduleux qui contient un tel bloc, mais qui reste quand même exécutable (ou utilisable pour une application quelconque), ne devrait pas poser de difficultés majeures à un bon spécialiste de la programmation <FERREIRA, 88>.

4.2. LA RENTABILITE DES DEFENSES ET DES ATTAQUES

Les attaques techniquement possibles ne représentent que peu de dangers si elles ne sont pas rentables pour l'ennemi (c'est-à-dire si leur coût est inférieur aux bénéfices espérés).

Ce prix de la fraude est souvent lié au niveau des mesures de sécurité prises. C'est pourquoi il est d'abord nécessaire de se poser la question de la rentabilité des défenses, car elles ne seront mises en oeuvre que si leur

coût est inférieur aux bénéfices (en termes de pertes évitées) estimés.

4.2.1. L'ANALYSE COUTS/BENEFICES DU POINT DE VUE DE LA DEFENSE

Dans les coûts de la défense, il faut principalement compter (<**FERNANDEZ, 81, 47**>):

- Une perte de performances.
- Un accroissement de complexité (qui peut entaîner un taux d'erreur plus élevé).
- Une perte de flexibilité du système informatique.
- Un accroissement des ressources afin d'administrer et de maintenir le système de sécurité.
- (éventuellement) un besoin accru en matériel.

A cela, il faut ajouter les frais de mise en place de la sécurité qui comprennent:

- Le développement des procédures et des programmes.
- L'acquisition de logiciels et de matériels.
- La formation du personnel.

Les bénéfices de la défense incluent principalement les pertes évitées. Celles-ci sont difficilement quantifiables (<**FERNANDEZ, 81, 48**>), surtout si l'on sait qu'en plus des pertes 'directes' (comme par exemple le montant des gains de l'attaquant dans le cadre d'une fraude bancaire), bon nombre de pertes 'indirectes' sont à prendre en compte. Dans <**PARKER, 86**>, on en cite 25, parmi lesquelles:

"legal fees and penalties, special audits, extra staff time, loss of valuable employees, embarrassment to management, and lost image of integrity, which in turn results in the loss of customers and a reduction in the value of the company's stock".

4.2.2. L'ANALYSE COUTS/BENEFICES DU POINT DE VUE DE L'ATTAQUE

Les coûts d' une attaque se chiffrent bien entendu en fontion du type d'attaque. Ils comprennent:

- Le prix du (ou des) processeurs.

- Le coût des moyens de stockage.

- Le montant des différentes analyses nécessaires à la mise en oeuvre de l'attaque.

- Le chiffrement du risque encouru par l'attaquant.

Notons également que l'acquisition d'un 'super ordinateur' à des fins frauduleuses pose quelques problèmes de discrétion <**FERREIRA, NP**>. Toutefois, il ne faut pas sous-estimer le risque qu'une de ces machines, appartenant par exemple à une banque ou à un centre de recherche, soit épisodiquement utilisée pour commettre une des attaques décrites dans cette partie.

Les bénéfices d'une attaque peuvent être considérables :

"Plus de 100 milliards de $ sont transférés quotidiennement à travers le monde par des moyens électroniques, et on estime que de 5 à 15 milliards de $ sont perdus annuellement pour cause de fraude" <**FERREIRA, 87**>.

"Aux Etats-Unis, un hold-up traditionnel rapporte en moyenne 3500$, lorsque l'auteur ne finit pas derrière les barreaux. Le délit informatique y est 140 fois plus lucratif, mais est aussi 100 fois plus délicat à démasquer" <**HARROIS-MONIN, 87**>.

Les bénéfices d'une attaque peuvent parfois se chiffrer non pas en gains réalisés, mais en pertes causées à la victime (cfr le point 4.2.1. de cette partie). Cette fraude, qui tient du terrorisme, est surtout à craindre dans les milieux industriels, militaires et commerciaux.

Au vu de ces quelques renseignements, nous pouvons affirmer qu'il existe, dans le monde informatisé qui est le nôtre, suffisamment d'opportunités de fraudes fructueuses pour justifier le coût d'une attaque, aussi important soit-il.

4.3. LA SYNTHESE DE L'EVALUATION

Il ressort de notre évaluation que des attaques exhaustives ou "meet-in-the-middle" sont faisables sous certaines conditions. Elles nécessitent des moyens et du temps dans de très grandes proportions. Il est bien évident que ce type de fraude n'est pas à la portée du premier venu. Toutefois, certains fichiers renferment des informations

d'une telle valeur qu'il est réaliste de craindre des attaques provenant de grandes organisations criminelles (comme par exemple la MAFIA) ou d'agences gouvernementales **<SUGARMAN,79>, <DESMEDT, 88>.**

Les attaques décrites aux points 3.2. et 3.3. de la troisième partie requièrent quant à elles peu de moyens et sont donc 'à la portée du premier venu', bien que cette affirmation soit à modérer en fonction des caractéristiques du système d'authentification à considérer. N'oublions pas en effet que nous avons supposé que la fonction d'authentification, la ou les clés et les paires fichiers-CAF étaient connues de l'ennemi. Dans un système réel, des mesures sont prises pour les garder secrètes autant que faire se peut.

Une fonction d'authentification possédant un CAF de 64 bits et une stucture empêchant toute attaque autre qu'exhaustive semble être un bon critère de sécurité. Si de plus les résultats intermédiaires ont 128 bits, la sécurité devient, semble-t-il, parfaite et ce au vu de la technologie prévue pour les quelques années à venir.

5. CONCLUSION

Dans cette partie, nous avons décrit différentes attaques à l'encontre d'une fonction d'authentification.

Les attaques exhaustives sont des attaques coûteuses et requérant énormément de temps. Nous avons montré que si le CAF possède une longueur d'au moins 64 bits, elles sont, si pas infaisable, au moins certainement non rentables. Une analyse statistique de la fonction d'authentification peut aider à diminuer le volume de travail nécessaire.

Contrairement aux attaques exhaustives, les attaques sur les blocs ne sont pas toujours possibles. L'objectif du concepteur d'un système d'authentification est donc de découvrir la fonction idéale, c'est-à-dire qui ne soit pas sensible à ce type d'attaque tout en étant raisonnablement rapide. Nous ferons une étude de ce type dans la partie suivante.

Avant de conclure, nous voudrions attirer l'attention du lecteur sur le fait que la présente énumération d'attaques n'est pas complète. De plus, il convient de maîtriser le contexte dans lequel évolue le système d'authentification. En effet, un ennemi ne dépensera pas des sommes excessives pour une attaque mathématique s'il peut par exemple courcircuiter le système d'authentification, ou même simplement soudoyer un programmeur, un opérateur ou un responsable de la sécurité.

Quatrième Partie

L'ANALYSE DE QUELQUES FONCTIONS D'AUTHENTIFICATION

1. INTRODUCTION

Dans le monde pourtant confiné de la cryptologie, il ne se passe pas une semaine sans qu'un nouvel algorithme ne soit publié. Devant cette multitude, il est bien difficile de faire un choix. On peut bien entendu prendre comme critère la renommée de l'auteur, mais même dans ce cas, des surprises ne sont pas à exclure. De grands spécialistes comme DAVIES, PRICE, MEYER, MATYAS, ... ont proposés des algorithmes qui n'ont pas résistés aux experts en cryptanalyse. D'autres algorithmes qui semblent intéressants du point de vue de la sécurité, se révèlent être totalement inapplicables pour des raisons de performances.

Nous nous proposons ci-dessous de passer en revue quelques algorithmes de fonction d'authentification présents dans la littérature, et de les classer en deux catégories.

La première sera l'objet du **chapitre 2.** Elle contiendra les algorithmes abandonnés, c'est-à-dire ceux qui, pour une raison ou une autre, ne remplissent pas les propriétés d'une bonne fonction d'authentification.

La deuxième catégorie sera l'objet du **chapitre 3.** Elle reprendra les algorithmes qui paraissent à première vue acceptables et qui seront analysés beaucoup plus en profondeur dans le **chapitre 4.**

La plupart de ces algorithmes seront décrits avec les notations introduites dans la deuxième partie de ce mémoire et résumées en fin de mémoire sous le titre 'notations et abréviations'.

Sauf mention contraire, le DES sera utilisé si une fonction de chiffrement est requise. Nous supposerons celui-ci sans défauts. Cela signifie qu'aucune fraude ne pourra se baser sur une propriété qui lui soit spécifique. Les attaques décrites resteront donc valables pour toutes les autres fonctions de chiffrement. Une telle hypothèse est à rapprocher du 'modèle de la boîte noire' que l'on utilise dans <WINTERNITZ, 84b>.

2. QUELQUES FONCTIONS A ABANDONNER

Nous énumérerons ci-dessous quelques-unes des fonctions d'authentification publiées dans la littérature et dont nous déconseillons l'emploi pour l'implémentation d'un système d'authentification de fichier.

2.1. LE MODE CBC DEFINI POUR LE DES <NBS, 80>

Le mode CBC défini pour le DES et décrit au point 3.5.1. de la seconde partie peut s'exprimer par le schéma suivant:

$$H1 = E_K(F1 \oplus VI)$$

$$Hi = E_K(Fi \oplus Hi-1) \quad \text{pour i allant de 2 à n}$$

CAF = les s bits les plus significatifs de Hn (avec s pouvant varier de 0 à 64)

Dans la troisième partie de ce mémoire, nous avons montré comment cette fonction se comportait face à diverses attaques. Il s'est avéré que dans un contexte où la clé d'authentification était connue de l'attaquant, cette fonction n'offrait aucune sécurité.

Il est vrai qu'à l'origine, le mode CBC a été conçu pour être utilisé pour l'authentification de messages financiers; application où il est courant de supposer que l'ennemi n'est pas en possession de la clé <ANSI, 82>.

"Toutefois, ce schéma est sensible à un type d'attaque lorsque le fraudeur a le loisir d'obtenir les (CAF) de tous les messages qu'il désire sans pour autant connaître la clé" <CAMPANA, 88>.

En fait, l'attaque n'est possible que si s = 64, c'est-à-dire si le dernier résultat Hn n'est pas tronqué pour former le CAF. Dans ces conditions, l'attaque fonctionne comme suit:

- Authentifier le fichier F1«F2«...«Fn-1. On obtient alors Hn-1.
- Construire un fichier frauduleux F1'«F2'«...«Fn-1'.
- Authentifier ce fichier. On obtient ainsi Hn-1'.
- Calculer Fn'de telle sorte que:

$$Fn' \oplus Hn-1' = Fn \oplus Hn-1.$$

Pour cela, il suffit de prendre:

$$Fn' = Fn \oplus Hn-1 \oplus Hn-1'.$$

Le fichier frauduleux F1'«F2'«...«Fn-1'«Fn' aura le même CAF que le fichier authentique F.

L'attaque décrite ci-dessus peut être contrée en tronquant le dernier résultat Hn. Cette solution est retenue dans le standard X9.9 <ANSI, 82>, lequel préconise de ne révéler que 32 bits des 64 bits terminaux.

2.2. LE DSA (Decimal Shift and Add) <DAVIES, 84, 127-130>

Le DSA est une fonction rudimentaire. Elle a été conçue pour faire de l'authentification 'manuelle' à l'aide d'une calculatrice de poche. Dans <CAMPANA, 88>, on affirme qu'elle est cassée. Cet avis est partagé par Mr QUISQUATER <QUISQUATER, 87>.

2.3. UNE PREMIERE FONCTION DUE A MEYER ET MATYAS <MEYER, 82, 399>

La fonction décrite ci-dessus fait usage d'un Code Détecteur de Manipulation (CDM) tel que décrit au point 4.2. de la seconde partie.

Si le CDM est calculé par un OU-exclusif bit-à-bit entre tous les blocs du fichier à authentifier, la fonction peut s'exprimer par le schéma suivant:

$$CDM = F1 \oplus \ldots \oplus Fn$$

$$H1 = E_K(F1 \oplus VI)$$

$$Hi = E_K(Fi \oplus Hi-1) \quad \text{pour } i \text{ allant de } 2 \text{ à } n$$

$$CAF = E_K(CDM \oplus Hn)$$

Il est intéressant de remarquer que malgré ce surplus de sécurité, la fonction reste très faible. Il est en effet montré dans <AKL, 84> que l'on peut:

- Utiliser une attaque "meet-in-the-middle"
- Modifier des blocs (la sémantique de ces modifications n'est toutefois pas choisie par l'ennemi)
- Insérer de nouveaux blocs (à la sémantique non choisie)

JUENEMAN propose un CDM de 128 bits qui permettrait de rendre cette fonction plus sûre <JUENEMAN, 87>. Ce CDM

étant relativement complexe et entraînant une perte sensible de performances (cfr le point 4.2.3. de la seconde partie), nous abandonnerons cette fonction.

2.4. UNE FONCTION DUE A RABIN <RABIN, 78>

La présente fonction, décrite au point 3.5.2. de la seconde partie, se caractérise par le fait que les blocs du fichier sont vus par la fonction de chiffrement E comme des clés et non pas comme des blocs. Ceci est exprimé par le schéma suivant:

$$H1 = E_{F1}(VI)$$

$$Hi = E_{Fi}(Hi-1) \quad \text{pour i allant de 2 à n}$$

$$CAF = Hn$$

Notons que si le DES est utilisé, les blocs Fi doivent avoir 56 bits contre 64 dans la plupart des autres fonctions. Les performances en sont donc réduites de 12,5 %.

Fait plus grave encore, il est montré dans <AKL, 84> et dans <WINTERNITZ, 84a> que cette fonction est sensible à une attaque "meet-in-the-middle".

2.5. UNE FONCTION DUE A BITZER <DAVIES, 83>

Une fonction d'authentification due à BITZER essaie de prévenir l'attaque "meet-in-the-middle" sur la fonction de RABIN. Pour cela, une sélection (sel) de 56 bits du résultat intermédiaire précédent est combiné à la clé courante, comme l'exprime le schéma suivant.

$$H1 = E_{F1 \oplus VI}(VI)$$

$$Hi = E_{Fi \oplus sel(Hi-1)}(Hi-1) \quad \text{pour i allant de 2 à n}$$

$$CAF = Hn$$

De façon étonnante, il s'est avéré que cette fonction est quand même attaquable par "meet-in-the-middle" <AKL, 84>, <WINTERNITZ, 84a>.

2.6. UNE FONCTION DUE A DAVIES ET PRICE <AKL, 84>

Afin d'empêcher une attaque "meet-in-the-middle", DAVIES et

PRICE ont proposé une fonction où le processus d'authentification est réalisé deux fois, ce qu'illustre le schéma suivant:

$$H1 = E_{Fi}(VI)$$

$$Hi = E_{Fi}(Hi-1) \quad \text{pour i allant de 2 à n}$$

$$Hn+i = E_{Fi}(Hn+i-1) \quad \text{pour i allant de 1 à n}$$

$$CAF = H2n$$

Remarquons tout d'abord que cette construction est deux fois moins performante que la fonction due à RABIN. De plus, contrairement à ce qui est affirmé dans <AKL, 84> et dans <WINTERNITZ, 84a>, cette fonction est sensible à une forme particulière d'attaque "meet-in-the-middle" que l'on peut trouver dans <COPPERSMITH, 86>.

2.7. UNE FONCTION DUE A DAVIES <WINTERNITZ, 84a>

Le schéma qui suit décrit une fonction d'authentification due à DAVIES et qui est présentée comme sûre dans <WINTERNITZ, 84a> et <CAMPANA, 88>.

$$H1 = VI \oplus E_{F1}(VI)$$

$$Hi = Hi-1 \oplus E_{Fi}(Hi-1) \quad \text{pour i allant de 2 à n}$$

$$CAF = Hn$$

La fonction évite l'attaque "meet-in-the-middle" en empêchant de travailler 'à reculons'. En effet, étant donné une valeur de Y, trouver n'importe quelle paire X,K telle que:

$$E_K(X) \oplus X = Y$$

semble nécessiter au moins de l'ordre de 2^{64} étapes de calcul.

Malheureusement, dans <WINTERNITZ, 84b>, il est montré qu'une astuce de calcul pouvait réduire ce nombre d'étapes proportionnellement à la quantité et à la taille des fichiers disponibles pour l'attaque. Dans le même article, une variation de cette fonction est proposée: un 'numéro de compte' est ajouté à chaque bloc et est ensuite incrémenté.

Cette technique, si elle élimine le danger de l'attaque précitée, possède néanmoins plusieurs désavantages:

- Elle requiert des blocs de fichiers plus petits, ce qui dégrade les performances.

- Elle nécessite la gestion et la sécurité du numéro de compte, ce qui augmente le risque de fraude par manipulation de la fonction d'authentification.

- Elle se limite à l'authentification de petits fichiers (une borne supérieure étant requise pour le numéro de compte).

Pour ces différentes raisons, nous abandonnons cette fonction.

2.8. QUELQUES FONCTIONS UTILISANT DES OPERATIONS MODULO

Il apparaît que la première proposition en matière de fonction utilisant des opérations modulo soit due à DAVIES et PRICE <AKL, 84>. Elle utilise un algorithme de chiffrement à clé publique: le RSA (cfr le point 4.3.2. de la première partie). Le schéma suivant décrit la fonction. Kd est la clé secrète et M le module public. Les blocs Fi ont la même taille que M, à savoir 512 bits.

$$Hi = (F1 + V1)^{Kd} \bmod M$$

$$Hi = (Fi + Hi-1)^{Kd} \bmod M \quad \text{pour i allant de 2 à n}$$

$$CAF = Hn$$

Cette fonction a l'avantage de combiner à la fois la compression et la signature. En effet, seul le possesseur de Kd a pu calculer les Hi consécutifs. Une tierce personne peut uniquement, grâce à la clé publique, vérifier que CAF est bien le code d'authentification de F.

Elle possède toutefois des performances médiocres et elle est sensible à l'attaque 'working-backwards' décrite au point 3.2. de la troisième partie <AKL, 84>.

Une autre proposition de DAVIES et PRICE vise à accélérer la fonction précédente. Pour cela, l'exposant Kd est remplacé par un simple carré, ce qui, en moyenne, augmente les performances d'un facteur de 750 <FERREIRA, 87>. La longueur des blocs est réduite à 448 bits. Ceci revient à imposer une redondance de 64 bits, tous égaux à zéro, en tête de chaque bloc. Celle-ci a pour but de rendre inefficace l'attaque 'working-backwards' puisque le bloc 'corrigé' nécessaire à cette dernière n'aurait à priori qu'une chance sur 2^{64} de satisfaire la règle de redondance <CAMPANA, 88>.

Cette fonction s'étant avérée sensible à une attaque décrite dans <GIRAULT, AP>, d'autres propositions ont été faites. Il semble toutefois qu'aucune ne se soit révélée à la fois assez sûre et assez rapide pour concurrencer les

fonctions que nous présenterons dans le chapitre suivant (<CAMPANA, 88>, <QUISQUATER, 88>).

Notons toutefois qu'une telle technique d'authentification est à l'heure actuelle en cours d'examen à l'ISO <CAMPANA, 88>. Si elle obtient le statut de standard international, le jugement plutôt négatif exprimé ci-dessus devra être revu en raison de l'intérêt généralement suscité par ce type d'officialisation <FERREIRA, 88>.

2.9. UNE FONCTION DUE A GILBERT, MAC WILLIAMS ET SLOANE <GILBERT, 74>

Cette méthode semble être irréprochable du point de vue de la sécurité. Elle est malheureusement inutilisable en pratique car elle nécéssite l'emploi d'un CAF de même ordre de grandeur que le fichier à authentifier.

2.10. UNE FONCTION DUE A WEGMAN ET CARTER <WEGMAN, 81>

Cette fonction offre l'avantage appréciable d'être sûre avec une probabilité choisie par le constructeur. Mais tout comme la fonction précédente, elle est inutilisable en pratique. Pour un CAF d'une longueur m et un fichier de longueur b, on doit utiliser une clé de longueur

$$4 \log(b) (m + \log(\log(b))).$$

Si par exemple m = 16 octets et b = 1 Mo, la clé devra faire plus de 176000 octets.

3. QUELQUES FONCTIONS INTERESSANTES

Nous énumérerons ci-dessous quatre fonctions qui nous semblent intéressantes pour l'implémentation d'un système d'authentification.

Nous voulons insister sur le fait qu'elles ne sont pas sans reproche. En effet, il n'existe aucune preuve fiable de leur sécurité. C'est pourquoi nous conseillons à toute personne désirant utiliser l'une d'elles de consulter régulièrement la littérature afin d'être informée des derniers progrès en matière de cryptanalyse.

3.1. UNE DEUXIEME FONCTION DUE A MEYER ET MATYAS <MEYER, 82, 399>

Une deuxième fonction due à MEYER et MATYAS est reconnue comme étant extrêmement sûre <AKL, 84>, <JUENEMAN, 84>. Elle utilise deux clés différentes (K1 et K2), le mode d'opération CBC et un code détecteur de manipulation (CDM). Elle peut être exprimée par le schéma suivant:

$$CDM = F1 \oplus \ldots \oplus Fn$$

$$H1 = E_{K1}(F1 \oplus VI)$$

$$Hi = E_{K1}(Fi \oplus Hi-1) \quad \text{pour i allant de 2 à n}$$

$$CAF1 = E_{K1}(CDM \oplus Hn)$$

$$H1^{\circ} = E_{K2}(F1 \oplus VI)$$

$$Hi^{\circ} = E_{K2}(Fi \oplus Hi-1^{\circ}) \quad \text{pour i allant de 2 à n}$$

$$CAF2 = E_{K2}(CDM \oplus Hn^{\circ})$$

$$CAF = CAF1 \ll CAF2$$

Dans la description de l'algorithme faite par les auteurs, la fonction de chiffrement E est le DES, ce qui correspond à une clé totale de 112 bits et à un CAF de 128 bits.

Pour différencier cette fonction d'authentification des trois autres décrites dans la suite de ce chapitre, nous la noterons MM82.

3.2. UNE FONCTION DUE A MEYER ET SCHILLING

En 1988, MEYER et SCHILLING ont proposé une fonction conçue pour résister aux attaques d'un ennemi en possession de toutes les informations qu'il désire. La description suivante, inspirée de **<MEYER, 88>**, est mieux comprise en se référant à la figure 4.1.

Dans cette fonction d'authentification, les fichiers sont découpés en blocs Fi de 64 bits afin d'être utilisés en tant que données par le DES.

Le processus de génération d'un CAF est initialisé par deux clés de 64 bits: K1(1) et K2(1). Elles peuvent être standardisées ou générées aléatoirement.

La brique de base de la fonction est la transformation:

$$f(Kj,X) = X \oplus E_{Kj}(X) \quad \text{avec } j = 1 \text{ ou } 2$$

Les clés K1 et K2 sont modifiées à chaque étape de calcul. Comme nous le verrons dans le chapitre suivant où est analysé cette fonction, il est indésirable d'obtenir certaines valeurs de clés, et ce par mesure de sécurité. Ces valeurs sont donc éliminées par une modification systématique de K1 et K2. La transformation f est donc en fait:

$$f(Kj,X) = X \oplus E_{mod(Kj)}(X) \quad \text{avec } j = 1 \text{ ou } 2$$

Sans entrer dans les détails, nous dirons que mod(Kj) supprime les 8 bits de parité de Kj, et ensuite impose à deux bits de prendre la valeur '10' si j = 1 et '01' si j = 2. Remarquons que ceci a pour conséquence de réduire l'espace d'une clé à 54 bits.

Le schéma proprement dit de la fonction est le suivant:

A = f(K1(i),Fi)

AL et AR sont générés en scindant A en 2 blocs de 32 bits : A = AL∗AR

B = f(K2(i),Fi)

B = BL∗BR

Les deux nouvelles clés sont obtenues à partir de A

et B par une 'référence croisée' ('cross coupling'):

- K1(i+1) = AL‖BR (la moitié gauche de A est concaténée à la moitié droite de B)

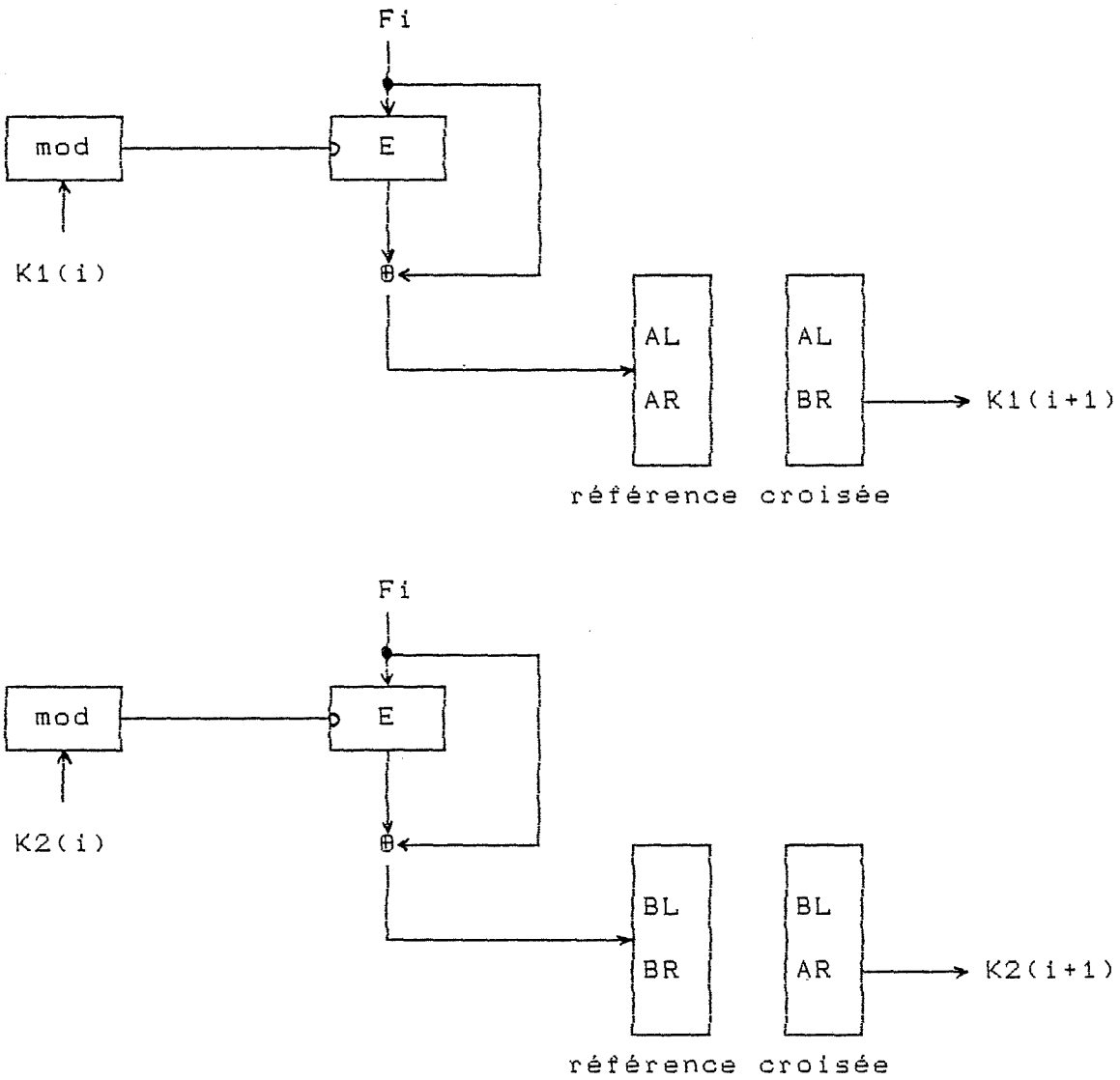
- K2(i+1) = BL‖AR (la moitié droite de B est concaténée à la moitié gauche de A)

Ce schéma est itéré pour i allant de 1 à n, et le CAF est obtenu en mettant bout-à-bout K1(n) et K2(n):

CAF = K1(n)‖K2(n)

Nous noterons cette fonction d'authentification MS88.

FIGURE 4.1: Une troisième fonction d'authentification due à MEYER et SCHILLING



Pour une plus grande sécurité, MEYER et SCHILLING proposent une autre fonction où le DES n'est pas employé deux fois mais quatre fois par itération. Cette fonction peut être facilement construite en utilisant deux fois MS88. Bien qu'elle semble plus sûre que le MS88, nous ne la retiendrons pas pour des raisons évidentes de performances.

3.3. UNE FONCTION DUE A QUISQUATER <QUISQUATER, 88>

La dernière fonction présentée ci-dessous utilise quatre chiffrements par le DES pour une itération, c'est-à-dire pour le traitement de 64 bits de fichier. La justification de ce surplus d'efforts est un accroissement de la sécurité **<MEYER, 88>**.

Or,"*à l'heure actuelle, il n'existe aucune preuve indiquant qu'une fonction de condensation à sens unique sûre nécessite absolument plus d'un chiffrement par itération*" **<QUISQUATER, 88>**.

Pour appuyer cette affirmation, Mr QUISQUATER propose la fonction suivante que nous noterons QU88:

Un fichier à authentifier est divisé en blocs de 56 bits qui seront transformés en entrant par la porte de la clé du DES. Afin d'obtenir un CAF de 128 bits, deux flux parallèles sont définis. Chaque bloc de fichier va directement influencer l'un des flux selon que le numéro courant de l'itération est pair ou impair. L'autre flux reste inchangé:

$$H_i = H_{i-1}$$

$$\text{et } H_i^\circ = E_{F_i}(H_{i-1}) \oplus H_{i-1}^\circ \qquad \text{(si i est pair)}$$

$$H_i = E_{F_i}(H_{i-1}^\circ) \oplus H_{i-1}$$

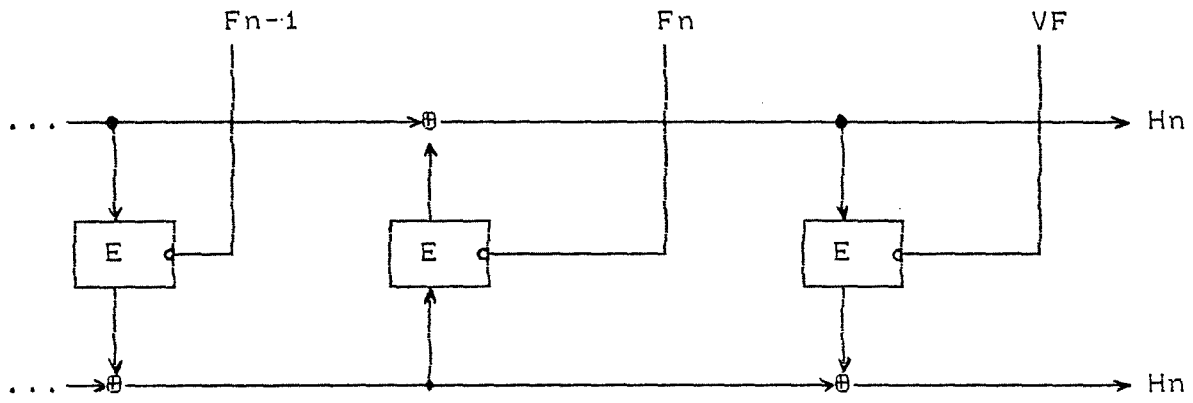
$$\text{et } H_i^\circ = H_{i-1}^\circ \qquad \text{(si i est impair)}$$

Ce schéma est itéré pour i allant de 1 à n.

On peut remarquer que les deux flux de calcul subissent l'influence des n-1 premiers blocs Fi, alors que Fn n'agit que sur un seul. Afin de remédier à cette faiblesse, un bloc final (VF) de valeur nulle (ou n'importe quelle autre constante) est ajouté en fin de fichier.

Ce schéma correspond au principe de base de la fonction d'authentification. Il est illustré par la figure 4.2.

FIGURE 4.2: Une fonction d'authentification due à QUISQUATER: principe de base



Comme nous le montrerons dans le chapitre suivant où le QU88 sera analysée, le schéma de base présenté ci-dessus n'est pas totalement satisfaisant. Afin d'augmenter la propagation des erreurs, des croisements entre les deux flux de calcul sont ajoutés. Le schéma complet, illustré par la figure 4.3 devient alors le suivant:

$$Hi = Hi\text{-}1 \oplus Hi\text{-}2^{\circ}$$

(si i est pair)

$$et\ Hi^{\circ} = E_{Fi}(Hi\text{-}1) \oplus Hi\text{-}1^{\circ}$$

$$Hi = E_{Fi}(Hi\text{-}1^{\circ}) \oplus Hi\text{-}1$$

(si i est impair)

$$Hi^{\circ} = Hi\text{-}1^{\circ} \oplus Hi\text{-}2$$

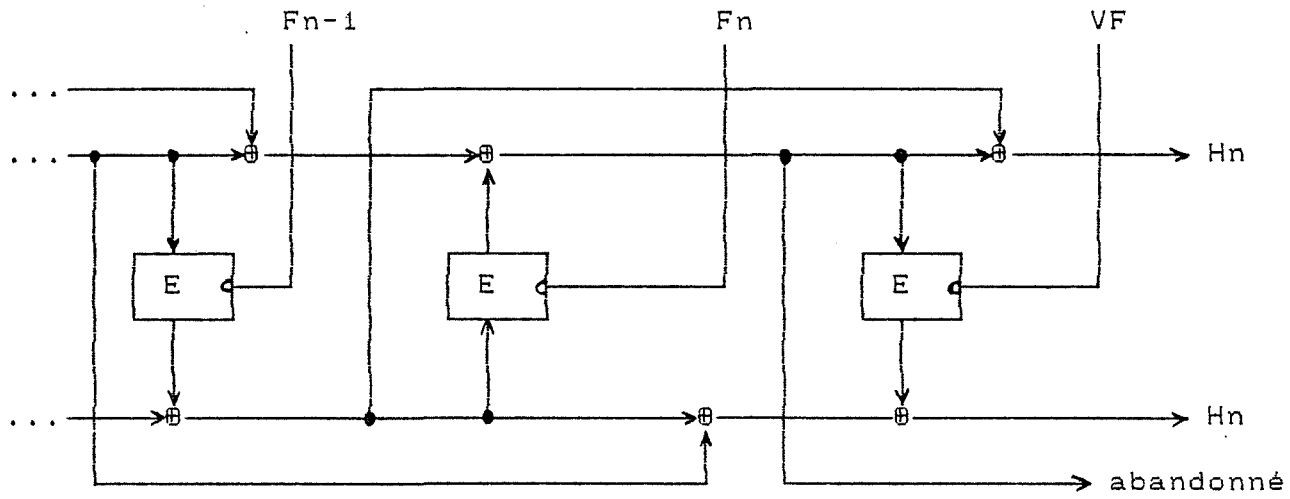
Ce schéma est itéré pour i allant de 1 à n, une dernière étape étant utilisée pour traiter le bloc VF.

Trois valeurs d'initialisation doivent être définies. Elles sont utilisées pour remplacer les résultats intermédiaires H0, H0° et H-1 nécessaires au traitement des deux blocs F1 et F2.

Le code d'authentification est obtenu par concaténation des résultats finaux des deux flux :

$$CAF = Hn \ll Hn^{\circ}.$$

FIGURE 4.3: Une fonction d'authentification due à QUISQUATER: schéma complet



3.4. LE "MESSAGE AUTHENTICATOR ALGORITHM" (MAA) <ISO, 86c>

Nous présentons ci-dessous une fonction d'authentification due à DAVIES et CLAYDEN : le "Message Authenticator Algorithm" ou MAA **<DAVIES, 85>**.

L'algorithme est conçu pour être utilisé dans un contexte où la clé est gardée secrète. Si on ajoute à cela le fait qu'il n'est composé que de quelques opérations élémentaires et que son CAF est limité à 32 bits, le MAA ne peut en aucun cas espérer tenir la comparaison avec les fonctions décrites aux points 3.1., 3.2. et 3.3. du présent chapitre.

Le MAA possède pourtant d'autres atouts:

- Les fonctions précédentes utilisent le DES qui n'est pas très bien adapté pour une implémentation logicielle **<MADRYGA, 84>**. Le MAA, lui, convient particulièrement bien aux opérations arithmétiques et logiques sur 32 bits que l'on trouve sur la plupart des processeurs classiques.

- Il est conçu pour offrir une authentification rapide.

- Il a été normalisé par l'ISO **<ISO, 86c>**.

Nous avons donc pris le parti de classer le MAA parmi les algorithmes 'intéressants', même si nous ne conseillerions pas son utilisation pour l'implémentation d'un système d'authentification nécessitant une haute sécurité. Il nous semble toutefois qu'il peut rendre de très grands services pour l'authentification 'à petite échelle', c'est-à-dire dans un contexte centralisé et de faible importance

financière. La description qui suit est inspirée de **<DAVIES, 84, 130-132>**.

Le MAA est conçu pour donner une combinaison de vitesse et de sécurité pour tout processeur pouvant rapidement produire le résultat en double longueur de la multiplication de deux nombres non signés de 32 bits.

La figure 4.4 montre la boucle principale ('main-loop') du calcul du CAF. Les opérations sont montrées en colonnes pour mettre en évidence leur parallélisme, mais elles doivent être lues de gauche à droite et de bas en haut. Tous les opérandes sont des nombres de 32 bits non signés. CYC est une rotation de un bit sur la gauche. MUL1 est la multiplication modulo $(2^{32} - 1)$ et MUL2A la multiplication modulo $(2^{32} - 2)$. A, B, C, D sont des constantes qui aident à conditionner les multiplicateurs F et G. ADD est l'addition modulo 2^{32}. XOR, OR et AND sont les opérations logiques de même nom travaillant en parallèle sur les 32 bits de leurs opérandes. Chacun des deux multiplicateurs F et G est dérivé de l'autre flux de calcul en employant les constantes A, B, C, D et la valeur E, elle même dérivée directemment de la clé.

FIGURE 4.4 : 'main-loop' du MAA

```
        V = CYC(V)                      !  V et W sont dérivés
        E = XOR(V,W)                    !  de la clé.

X = XOR(X,Fi)    Y = XOR(Y,Fi)          !  Fi est un bloc de
F = ADD(E,Y)     G = ADD(E,X)           !  fichier.
F = OR(F,A)      G = OR(G,B)            !  A, B, C et D sont
F = AND(F,C)     G = AND(G,D)           !  des constantes.
X = MUL1(X,F)    Y = MUL2A(Y,G)         !  MUL1 et MUL2A sont
                                        !  des multiplications
                                        !  avec module
```

Le CAF est dérivé des deux flux de calcul par un ⊕ appliqué aux valeurs finales de X et Y.

Les clés qui contrôlent le processus d'authentification sont deux nombres non signés de 32 bits. Elles sont utilisées dans une phase dite 'de prélude'. Cette première étape est une procédure complexe, employant les multiplications avec module décrites ci-dessus et générant six nombres de 32 bits. Le prélude n'est exécuté qu'une seule fois pour chaque paire de clés. Les six valeurs sont alors stockées dans un environnement sûr et employées à chaque nouvelle itération du MAA. Deux de ces nombres sont les valeurs d'initialisation pour X et Y; deux autres, V et W, sont utilisés pour modifier E; et les deux derniers sont employés comme blocs finaux et donc ajoutés à la fin du fichier à authentifier.

4. L'ANALYSE DE QUATRE FONCTIONS

Ci-dessous, nous analyserons les quatre fonctions d'authentification proposées au chapitre précédent: le MM82, le MS88, le QU88 et le MAA.

L'analyse portera sur la sécurité et les performances. Ces dernières seront données en termes de comparaison par rapport à la vitesse du DES qui est généralement bien connue pour la plupart des processeurs. Cela permettra donc d'évaluer rapidement la vitesse des fonctions sur ces processeurs. A titre de synthèse, nous reprendrons ces évaluations dans un tableau comparatif.

En raison de leur grande rapidité, les opérations logiques et celles d'affectation pourront être abandonnées sans trop nuire à la précision des estimations de performances.

4.1. L'ANALYSE DU MM82

4.1.1. L'ANALYSE DE LA SECURITE

La sécurité du MM82, comme celle des deux fonctions suivantes, repose sur un double flux de calcul. Celui-ci est obtenu en utilisant deux fois une première fonction due à MEYER et MATYAS décrite au point 2.3 de la présente partie. Deux clés différentes K1 et K2 étant employés, deux CAF de 64 bits sont obtenus , respectivement CAF1 et CAF2.

Au même point 2.3., nous avons montré qu'il était possible de trouver un fichier F'≠ F tel que H(F') = CAF1. Toutefois, ce même fichier F', authentifié avec la clé K2 ≠ K1, possède une chance infime de produire les 64 bits de CAF2. La combinaison de deux fonctions non-sûres semble donc produire une nouvelle fonction beaucoup plus solide.

Nous ne pouvons donner la preuve de la sécurité du MM82. Précisons toutefois qu'il résiste depuis plus de six ans aux attaques des spécialistes de la cryptanalyse. Si l'on en croit certains auteurs, ceci constitue la meilleure garantie de robustesse qui soit (<WINTERNITZ, 84b>, <JUENEMAN,84>).

4.1.2. L'ANALYSE DES PERFORMANCES

Un rapide examen de l'algorithme du MM82 présent au point 3.1. du chapitre précédent nous permet de donner l'approximation suivante:

vitesse du MM82 = vitesse du DES à clé constante / 2

Remarque: L'expression 'à clé constante' est ici très importante. Il faut en effet savoir qu'un rapport du simple au quintuple peut exister entre la vitesse du DES à clé constante et à clé variable. Ceci est dû au fait que dans le premier cas, la partie de l'algorithme destinée à générer 16 sous-clés est effectuée une fois pour toutes. Dans le second cas, ce travail doit être réalisé pour chaque nouveau bloc de données **<QUISQUATER, 87>**. Pour rappel, signalons que la structure du DES est détaillée dans l'annexe 2.

4.2. L'ANALYSE DU MS88

Les deux analyses présentées ci-dessous sont inspirées de **<MEYER, 88>**.

4.2.1. L'ANALYSE DE LA SECURITE

La brique de base de l'algorithme est la fonction à deux variables:

$$f(K,X) = X \oplus E_K(X)$$

Soit une paire (K,X) telle que f(K,X) = v. La tâche consistant à trouver une valeur X' ≠ X telle que f(K,X') = f(K,X) = v, requiert apparemment 2^{64} essais, sauf si K est l'une des 'clés faibles' ('weak keys') ou 'semi-faibles' ('semi-weak keys') du DES (*).

K1 et K2 étant modifiés à chaque itération de l'algorithme, il est important d'empêcher qu'elles ne prennent la valeur d'une des clés 'faibles' ou 'semi-faibles'. Ceci est réalisé en forçant systématiquement la valeur de deux bits avant que les clés ne soient utilisées par le DES.

Plusieurs attaques ont été essayées par les auteurs, mais aucune ne semble s'être montrée satisfaisante.

Le MS88 a été soumis à un grand spécialiste de la cryptanalyse: le Dr Don COPPERSMITH de 'IBM Research'. Celui-ci a diagnostiqué les résultats suivants: étant donné une paire F-CAF, un ennemi ne peut espérer trouver un fichier frauduleux F' ayant même CAF que F que s'il réalise une attaque nécessitant:

(*) Ces clés sont au nombre de 16. Leurs propriétés sont présentées dans **<MEYER, 82, 147-153>**.

- 2^{55} utilisations du DES. A 1 *Ms* (microseconde) l'itération, l'ensemble de la recherche prend plus de 1000 ans).

- 2^{55} stockages de mots de 128 bits, ou encore 2^{29} Go.

Ceci est bien au delà des possibilités techniques actuelles. Toutefois, l'algorithme étant récent (1988), il n'est pas possible de certifier qu'il est sans failles. Etant donné le nombre de fonctions qui se sont révélées peu sûres ces dernières années, il est plus prudent d'attendre que le monde de la recherche se soit penché sur sa structure avant de lui attribuer des qualités de robustesse.

4.2.2. L'ANALYSE DES PERFORMANCES

Un rapide examen de la figure 4.1 nous permet de donner l'approximation suivante:

vitesse du MS88 = vitesse du DES à clé variable / 2

Comme nous l'avons déjà fait remarquer, le fait de travailler 'à clé variable' nuit fortement aux performances de cette fonction.

Dans <MEYER,88>, on cite les résultats suivants:

- Implémentation logicielles sur IBM PC / XT: 80 octets/s.

- Implémentation matérielle sur IBM 5977 (*): 5 Ko/s.

4.3. L'ANALYSE DU QU88

4.3.1. L'ANALYSE DE LA SECURITE

La sécurité de la fonction réside dans la difficulté de résoudre l'équation:

$$E_X(A) = B$$

Pour A et B connus, la solution requiert en effet en moyenne 2^{55} applications du DES.

(*) Il s'agit d'un "terminal de chiffrement programmable que l'on peut attacher à divers ordinateurs via une communication en série et qui dispose d'une implémentation matérielle du DES" <MEYER,88>

Nous avons pourtant rencontré dans le chapitre 2 de la présente partie des fonctions employant la même équation et qui sont sensibles à des attaques (cfr notamment les points 2.4 à 2.7 de ce chapitre).

La faiblesse de ces fonctions réside dans le mode de chaînage qui permet d'utiliser l'une ou l'autre attaque sur les blocs présentée au chapitre 3 de la troisième partie.

Grâce au double flux de calcul, schématisé par la figure 4.2, ces attaques sont impossibles. L'ennemi est obligé d'utiliser en moyenne ($2 * 2^{55}$ =) 2^{56} applications du DES s'il désire manipuler un fichier. Il doit en effet résoudre deux fois l'équation citée ci-dessus pour arrêter la propagation des erreurs dans le premier flux et ensuite dans le deuxième.

Une telle quantité de travail est difficilement conciliable avec les réalités techniques et surtout financières actuelles. On peut toutefois craindre que dans un avenir proche, ce type d'attaque devienne faisable.

C'est sans doute pourquoi les références croisées entre flux de calcul ont été ajoutées à la figure 4.2 pour obtenir la figure 4.3. Avec un tel chaînage, il ne semble plus possible de compenser dans les deux flux à la fois les erreurs introduites par la modification frauduleuse d'un bloc de fichier.

Nous devons toutefois faire remarquer que le QU88 est la fonction d'authentification la plus récente de toutes celles présentées. De ce fait, elle n'a pas encore été examinée par le monde scientifique. Il est donc nécessaire d'émettre les réserves d'usage quant à sa sécurité.

4.3.2. L'ANALYSE DES PERFORMANCES

Un rapide examen de la figure 4.3 nous permet de donner l'approximation suivante:

vitesse du QU88 = vitesse du DES à clé variable * 7/8

Le coefficient 7/8 provient du fait que seuls 56 bits de données sont traités en une itération contre 64 pour le DES.

4.4. L'ANALYSE DU MAA

4.4.1. L'ANALYSE DE LA SECURITE

Le MAA est une pure fonction d'authentification au sens où elle n'est basée sur aucun algorithme de chiffrement connu.

Le CAF ne fait que 32 bits, ce qui est nettement insuffisant pour résister à une attaque exhaustive. Remarquons toutefois que par une modification minime de l'algorithme, il pourrait très bien être plus long, et ce jusque 64 bits.

La boucle principale est non-réversible <CAMPANA, 88>. Ceci empêche toute attaque sur les blocs comme celles décrites aux points 4.3. de la partie précédente.

Une expertise menée au nom de l'ISO a montré que <ISO, 85> :

- Le MAA opère suffisamment bien comme fonction pseudo-aléatoire de la clé et du fichier.
- Il contient certaines régularités internes qui sont observées lors de l'utilisation de longs fichiers.
- Il possèderait "une probabilité raisonnable d'être susceptible de succomber à une attaque" lorsqu'il est utilisé pour l'authentification de longs fichiers.

Les précautions prises pour parler d'une éventuelle faiblesse et le fait qu'aucune indication ne soit donnée pour montrer comment une attaque pourrait réussir nous invitent à croire qu'il s'agit seulement de présomptions de faiblesse.

Toutefois, il est plus prudent de suivre le conseil suivant:

"Il ne faut pas employer le MAA dans le cas de fichiers longs et redondants, ou périodiques, et ce quelle que soit la clé" <CAMPANA, 88>.

4.4.2. L'ANALYSE DES PERFORMANCES

Contrairement aux trois premières fonctions étudiées, les performances du MAA ne peuvent se calculer sur base de celles du DES. C'est pourquoi, dans le cadre d'un stage au Centre de Technologies Informatiques (CTI) de Fontenay aux

Roses (France), nous avons réalisé plusieurs implémentations de cette fonction (*) en langage C afin d'en évaluer la vitesse. Chaque version a été conçue pour utiliser au mieux les possibilités du processeur sur lequel elle doit s'exécuter. L'optimisation a été axée sur la vitesse.

Les résultats les plus intéressants sont les suivants:

- sur P3200 (processeur Intel 80286) : 7,3 Ko/s

- sur un processeur Intel 80386 : 57 Ko/s

Le gain de performance entre le processeur du P3200 (16 bits) et l'Intel 80386 (32 bits) n'est pas seulement dû à la différence d'horloge (respectivement 6 et 16 MHz), mais aussi au fait que le MAA est presque exclusivement formé d'instructions sur 32 bits.

Notons enfin que le langage C ne permet pas d'accéder au résultat sur 64 bits de la multiplication de deux nombres de 32 bits. Une nouvelle version en C intégrant quelques instructions assembleurs de l'Intel 80386 nous a permis d'utiliser cette opération, et ainsi de dépasser la vitesse de 100 Ko/s.

4.5. LE RETOUR A LA NOTION DE FONCTION "COLLISION FREE"

Jusqu'à présent, nous avons toujours travaillé avec l'hypothèse selon laquelle une attaque frauduleuse portait sur une ou plusieurs paires fichier-CAF. Dans ces conditions, l'ennemi tente de trouver un fichier frauduleux ayant le même CAF qu'un des fichiers considérés comme authentiques. Cette approche de la sécurité des fichiers n'est pas unique.

Certains auteurs <QUISQUATER, 88>, <CAMPANA,88>, <MEYER, 88> considèrent un autre danger, appelé parfois 'attaque de l'intérieur' <MEYER, 88>. Bien que cette fraude soit mentionnée la plupart du temps dans la littérature relative à la communication électronique, elle reste aussi applicable dans le cadre de la protection de fichiers.

L'attaque, déjà mentionnée au point 2.3. de la seconde partie fonctionne ainsi:

- considérons une personne 'de l'intérieur', c'est-à-dire "who have access to the system (...) and who can

(*) Les détails relatifs à ces implémentations sont disponibles auprès de Mr FERREIRA <FERREIRA, 87>

have one (file) authenticated on his behalf" <MEYER,88>. Par la suite, nous prendrons comme exemple un programmeur félon.

- Ce programmeur est chargé de construire un quelconque fichier qui sera protégé par un système d'authentification.

- Il sait que son travail sera examiné par des experts et qu'aucune fraude ne leur échappera.

- Il construit le fichier correctement, mais d'un autre côté, il en prépare une version frauduleuse.

- Selon une technique présentée au point 4.1.5. de la troisième partie, il réalise $2^{m/2}$ variantes des deux fichiers (m étant la longueur en bits du CAF).

- Pour chacun des deux ensembles de $2^{m/2}$ fichiers ainsi formés, il calcule le CAF.

- Selon le paradoxe de l'anniversaire présenté au point 3.3.1. de la troisième partie, il existe une grande probabilité qu'une correspondance existe entre un CAF du premier ensemble et un CAF du second.

- Le programmeur présente alors le fichier correct pour lequel la correspondance a eu lieu.

- Plus tard, il remplace ce fichier par la variante frauduleuse qui possède le même CAF.

Cette attaque peut sembler rocambolesque, mais il est des situations où elle représente une réelle menace. Remarquons qu'il ne s'agit pas, comme nous l'avons supposé tout au long de ce mémoire, d'une fraude à l'encontre d'un ensemble donné de fichiers, mais visant un nouveau fichier.

Afin de nous prémunir contre ce risque, il nous faut être plus sévère quant aux propriétés d'une bonne fonction d'authentification. Elle doit être telle qu'il soit impossible de trouver deux fichiers F et F' tels que H(F) = H(F'). Une fonction répondant à un tel critère est appelée "collision free" <DAMGARD, AP>.

Dans <MEYER, 88>, une expertise du MS88 due au Dr COPPERSMITH montre qu'une attaque de l'intérieur nécessite 2^{54} applications du DES et un volume négligeable de stockage. Au vu de la technologie actuelle, ceci met le MM88 à l'abri de ce type d'attaque.

Nous ne savons pas comment se comportent le MM82 et le QU88 face au critère "collision free". Ils possèdent un CAF de 128 bits, ce qui est souvent considéré comme une condition suffisante pour résister à l'attaque brutale faisant jouer le principe de l'anniversaire <CAMPANA, 88>, <MEYER, 88>. Néanmoins, il existe peut-être des caractéristiques propres

à ces fonctions qui permettraient d'accélérer le processus de recherche. Seule une analyse appronfondie par un spécialiste en cryptanalyse pourrait nous renseigner.

Quant au MAA, il ne résiste pas à une attaque 'de l'intérieur'. Avec seulement 2^{17} applications de la fonction (quelques minutes sur un ordinateur individuel!), une paire frauduleuse de fichier peut-être trouvée.

4.6. LA SYNTHESE DE L'ANALYSE

La figure 4.5 synthétise les estimations de vitesse réalisées tout au long de cette analyse. Nous utiliserons pour ce faire les performances du DES dans une implémentation logicielle en C sur deux machines: le P3200 (processeur Intel 80286, 6MHz) et le VAX780 (VMS) **<DELESCAILLE, 88>**.

FIGURE 4.5: Synthèse des performances de 4 fonctions d'authentification

perf. / fct.	par rapport au DES	sur P3200	sur VAX780
MM82	vitesse du DES à clé constante / 2	1,6 Ko/s	3,4 Ko/s
MS88	vitesse du DES à clé variable / 2	338 octets/s	1,3 Ko/s
QU88	vitesse du DES à clé variable * 7/8	593 octets/s	2,3 Ko/s
MAA	/	7,3 Ko/s	?

Les performances ne doivent cependant pas être les seuls critères de choix d'une fonction d'authentification. Dans ce domaine, tout est une question de compromis entre la rapidité et la sécurité.

MS88 et QU88 semblent à première vue très solides, mais ils posent de réels problèmes quand ils sont utilisés pour authentifier de grands fichiers. A titre d'exemple, l'implémentation logicielle du QU88 sur VAX780 (VMS) demande 7 minutes et 45 secondes pour authentifier un fichier de 1 Mo. Dans de nombreux contextes, ce délai est inadmissible.

Afin de rendre attractives de telles fonctions, une implémentation matérielle performante est indispensable. Avec une vitesse de DES de 1,25 Mo/s, citée dans **<PAILLES, 86>**, l'authentification d'un fichier de 1 Mo requiert:

- 1,6 seconde pour MS88,

- 0,9 seconde pour QU88.

Le MAA est quant à lui performant dans une implémentation logicielle: 10 secondes pour authentifier un fichier d'un Mo sur un processeur Intel 80386 à 16 MHz.

Malheureusement, nous avons montré que la fonction n'était pas sûre pour l'authentification de très grands fichiers, et qu'elle ne résistait pas à une attaque 'de l'intérieur'.

Nous pensons toutefois que le MAA peut être avantageusement utilisé dans un contexte de faible envergure comme par exemple pour l'authentification de petits fichiers présents sur le disque dur d'un micro-ordinateur. Si la clé peut être gardée secrète, l'attaque 'de l'intérieur' n'est plus possible et le MAA redevient sûr.

Le compromis entre la rapidité et le sécurité serait-il atteint par le **MM82**? Nous ne sommes pas loin de le penser. Dans une implémentation logicielle, il est plus rapide que MM88 et QU88. Mais est-il suffisamment solide? Si MEYER a proposé le MS88 pour l'implémentation d'un système d'authentification de fichier, cela signifie qu'il estimait que le MM82 ne remplissaient pas toutes les exigences de robustesse.

5. CONCLUSION

Dans cette partie, nous avons évoqués quelques-unes des fonctions d'authentification publiées dans la littérature, et nous les avons critiquées notamment grâce aux éléments introduits dans les deux parties précédentes.

Cette évocation n'est pas complète. Bien d'autres fonctions existent; le lecteur intéressé en découvrira six autres dans <CAMPANA, 88>. De nouvelles fonctions seront sans aucun doute publiées. Nous ambitionnons cependant d'avoir donné les éléments nécessaires à une approche critique de ces fonctions.

La partie suivante fera l'hypothèse que le constructeur d'un système d'authentification de fichiers possède une fonction 'parfaite', c'est-à-dire sûre et rapide. Nous avons montré qu'une telle fonction n'existe probablement pas. Par contre, il semble raisonnable d'espérer trouver une fonction qui réponde aux besoins de sécurité et de vitesse d'une application particulière. C'est dans cette optique que nous faisons les propositions suivantes :

- Pour une authentification sûre, utiliser le MS88 ou le QU88 tout en étant à l'écoute du monde de la cryptanalyse qui ne devrait pas tarder à réagir à la publication de ces deux nouvelles fonctions.

- Pour une authentification peut-être un peu moins sûre mais plus rapide, utiliser le MM82.

- Pour une authentification 'de petite envergure', utiliser le MAA.

Cinquième Partie

UNE PROPOSITION DE SYSTEME D'AUTHENTIFICATION

1. INTRODUCTION

Dans la première partie de ce mémoire, nous avons étudié le besoin de protection de l'intégrité des fichiers et nous avons choisi d'y apporter une solution par l'intermédiaire des fonctions d'authentification. Ces fonctions ont été analysées en profondeur dans les parties deux, trois et quatre. Pour ce qui va suivre, nous supposerons que nous disposons d'une fonction d'authentification 'parfaite', c'est-à-dire sûre et rapide.

Grâce à cette hypothèse, nous ébaucherons un système d'authentification permettant de protéger l'intégrité des fichiers présents dans un système informatique 'standard' que nous décrirons dans le **chapitre 2**.

La construction de ce système se fera de manière progressive dans le **chapitre 3**. Nous partirons d'hypothèses très favorables au constructeur, hypothèses que nous lèverons par après.

Ensuite, nous montrerons que dans les conditions les plus avantageuses pour l'attaquant, aucun système d'authentification sûr ne peut être défini sans un certain nombre d'outils présentés au **chapitre 4**.

Enfin, dans le **chapitre 5**, le système d'authentification proposé sera critiqué afin de définir son champ d'application.

Pour cette dernière partie, nous supposerons qu'il existe une technique de signature numérique qui soit beaucoup plus difficile à casser que la fonction d'authentification 'parfaite' citée ci-dessus. Cette hypothèse a pour but de reporter toute notre attention sur les fonctions d'authentification qui forment le coeur de ce mémoire.

Dans le système d'authentification qui sera ébauché ci-après, les fonctions de signature seront utilisées pour la protection de toutes les données (et notamment les CAF) transférées d'un endroit à un autre. Des canaux de communication matériellement sûrs, c'est-à-dire garantissant l'intégrité des données sans employer la cryptologie, permettraient bien entendu de simplifier les procédures de transferts <MEYER, 88>. Toutefois, dans une configuration informatique réelle, il est rare de rencontrer de tels canaux <FERREIRA, 87>. Pour cette raison, nous ne retiendrons pas cette hypothèse.

2. LA DESCRIPTION D'UN SYSTEME INFORMATIQUE

Afin d'illuster les propos qui vont suivre, nous présenterons un système informatique relativement simple. Il nous permettra de préciser les risques potentiels et d'indiquer où et quand il est possible (ou nécessaire) de vérifier les fichiers.

2.1. LE SCHEMA D'UN SYSTEME INFORMATIQUE

Dans les années 1970, le schéma type d'un système informatique était constitué d'un ordinateur que se partageaient un grand nombre d'utilisateurs à l'aide de multiples terminaux non intelligents. A l'heure actuelle, il est courant de rencontrer des installations contenant plusieurs ordinateurs de grande et moyenne taille, ainsi qu'un nombre important de terminaux intelligents (comme par exemple des micro-ordinateurs) <CARLSON, 86>. Nous schématiserons ce type de système informatique par la figure 5.1.

Le système est composé de deux sites distincts possédant chacun deux machines : un ordinateur et un micro-ordinateur. Chaque machine dispose d'une mémoire primaire et d'une mémoire secondaire. Un fichier en mémoire secondaire doit être transféré dans la mémoire primaire d'une machine pour pouvoir être utilisé par le processeur de cette machine.

Un ordinateur est directement accessible par l'intermédiaire de deux terminaux : le premier est le micro-ordinateur situé sur le même site, le second est un terminal non intelligent.

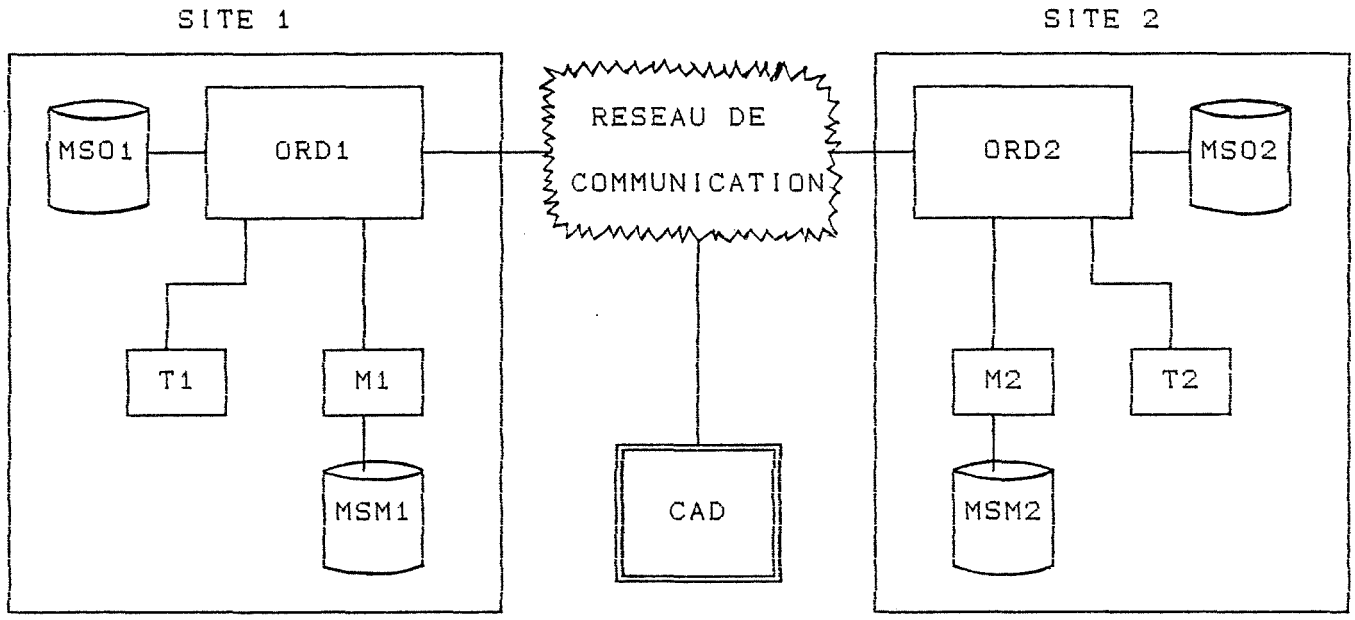
Les deux ordinateurs sont interconnectés, de sorte qu'il soit possible d'accéder aux quatre machines à partir de n'importe quel terminal.

Afin de nous conformer à l'hypothèse posée dans la première partie selon laquelle les nouveaux fichiers sont exempts de toutes erreurs ou manipulations frauduleuses, nous utiliserons le concept de Centre d'Achat et de Développement (CAD). Ce centre est chargé de fournir au système les fichiers nécessaires à son bon fonctionnement. Ceux-ci sont soit achetés, soit développés sur place. Dans les deux cas, ils sont examinés de manière à les certifier sans erreurs ni fraudes. Ils peuvent alors être installés dans l'un ou l'autre site. Dans la figure 5.1, nous avons choisi de représenter le CAD par un site indépendant des deux sites d'utilisation des fichiers. Cette solution est sans conteste la plus sûre qui soit. Toutefois, le CAD peut prendre d'autres formes.

Nous sommes conscients des limites de ce schéma. En particulier, la dichotomie 'mémoire primaire'- 'mémoire secondaire' est un peu simpliste. Où se situent en effet

les mémoires étendues, les mémoires statiques, les mémoires virtuelles et autres mémoires caches? Nous les négligerons et nous nous contenterons de placer la mémoire centrale dans la catégorie 'primaire' et les disques et bandes magnétiques dans la catégorie 'secondaire'. Une analyse de la sécurité des fichiers au sein d'une hiérarchie de mémoire plus élaborée est l'une des extensions intéressantes qui pourrait être réalisée à partir de ce mémoire.

FIGURE 5.1 : Le schéma d'un système informatique



ORD = ORDinateur
T = Terminal non intelligent
M = Micro-ordinateur
MSO = Mémoire Secondaire d'un Ordinateur
MSM = Mémoire Secondaire d'un Micro-ordinateur
CAD = Centre d'Achat et de Développement

2.2. OU SONT LES FICHIERS?

Dans un système informatique tel que décrit au point 2.1. ci-dessus, les fichiers peuvent être situés à différents endroits:

- En mémoire secondaire des ordinateurs ou des micro-ordinateurs.
- En mémoire primaire des mêmes machines.
- En transit entre * les deux ordinateurs,
  * le CAD et un ordinateur,

* un ordinateur et un micro-ordinateur,

* la mémoire secondaire et la mémoire primaire associées à une même machine.

2.3. OU ET QUAND VERIFIE-T-ON LES FICHIERS?

Nous pourrions nous baser sur le fait que les deux sites sont sûrs. Dans ces conditions, une attaque ne peut porter que sur un fichier transitant par le système de communication. Nous en revenons ainsi à la protection des messages traitée abondamment dans la littérature (cfr notamment <DAVIES, 84> et <MEYER, 82>). Dans ces conditions, il suffit de vérifier les fichiers à chaque fois qu'ils arrivent dans un des sites. Dans ce qui va suivre, nous abandonnerons cette hypothèse, trop restrictive, et considérerons donc que des ennemis peuvent frauder à l'intérieur d'un site.

Supposons maintenant qu'un fichier est en sécurité lorsqu'il se trouve dans la mémoire primaire d'une machine. Un ennemi peut le modifier partout ailleurs. Dans ces conditions, il est possible de spécifier plusieurs modalités de vérification :

- Elle peut se dérouler au moment où le fichier est chargé dans une mémoire primaire <MEYER, 88>. Avec l'hypothèse de sécurité posée ci-dessus, ce choix semble judicieux. En effet, un fichier peut alors résider un long moment en mémoire primaire et être utilisé plusieurs fois sans qu'il ne soit nécessaire de le vérifier de nouveau.

- Elle peut avoir lieu en mémoire primaire juste avant que le fichier ne soit traité par un processeur <POZZO, 86>. Cette option est moins performante que celle décrite précédemment car elle nécessite une vérification à chaque utilisation du fichier. Ceci est inutile dans un contexte où la mémoire primaire est sûre. Ce n'est malheureusement pas toujours vrai.

- On peut imaginer de placer les mesures de protection des fichiers encore plus près du processeur, plus exactement entre la mémoire primaire et le décodeur d'instruction du processeur. Cette mesure vise à empêcher la manipulation frauduleuse d'un fichier en raccourcissant le temps entre le moment où il est vérifié et le moment où il est effectivement utilisé <WHITE, 87>. Si l'idée est séduisante, elle n'est toutefois pas applicable dans le cas de l'emploi de CAF, ceux-ci se calculant sur le fichier en entier.

Ci-dessus, la vérification de l'intégrité des fichiers était supposée systématique. Il est pourtant des situations où des critères de performances l'emportent sur ceux de sécurité. Dans ce cas, on peut envisager de prendre certaines mesures parmi lesquelles :

- Ne pas faire d'authentification quand cela n'est pas strictement nécessaire. Nous pensons notammment au transfert d'un fichier d'un endroit à un autre, ou à sa destruction.

- Utiliser un système de vérification par morceaux. Il s'agit de diviser un fichier authentique en morceaux et de calculer un CAF pour chacun d'eux. Au moment de l'authentification, plutôt que de traiter le fichier dans son entièreté, on n'en vérifie qu'une partie choisie aléatoirement. L'objectif de cette découpe est de diminuer le temps d'une vérification.

  Il existe cependant le risque qu'un ennemi modifie une partie d'un fichier qui ne soit pas vérifiée avant une exécution, et qu'il remette ensuite le fichier dans sa version originale.

  Une autre attaque consiste à permuter plusieurs morceaux. Cette fraude peut être évitée en numérotant les morceaux et en incluant le numéro dans le calcul du CAF. Elle peut être encore plus facilement contrée en découpant le fichier en parties de taille différente.

- Vérifier les fichiers en entier, mais pas systématiquement. On peut très bien imaginer une vérification à un moment où la gêne est minimale pour l'utilisateur, comme par exemple lors du lancement du système.

  Afin de diminuer le risque d'une modification 'temporaire' de fichiers, cette mesure peut très bien être combinée à une authentification d'un certain nombre de fichiers sélectionnés aléatoirement et à des moments choisis tout aussi au hasard. Un schéma de priorité peut être défini afin de vérifier plus souvent certains fichiers plus critiques.

Les deux dernières suggestions ci-dessus possèdent le même défaut : celui de laisser des 'trous' sans vérification. Pour peu qu'un ennemi remette en état un fichier après l'avoir modifié, la fraude peut passer inaperçue si elle profite d'un de ces 'trous'. Par contre, avec ces deux techniques, le gain en terme d'efficacité est évident. Nous retrouvons ici une fois de plus le problème de l'équilibre à trouver entre la sécurité et les performances.

Dans ce qui va suivre, nous nous intéresserons principalement à la sécurité, c'est pourquoi nous retiendrons la solution consistant à authentifier un fichier juste avant son utilisation par un processeur.

3. LA DEFINITION PROGRESSIVE D'UN SYSTEME D'AUTHENTIFICATION

Ci-dessous, nous construirons un système d'authentification relativement général en enrichissant un premier système basé sur les hypothèses suivantes:

- Il existe une seule fonction d'authentification : H.
- Il existe une seule paire de fonction de signature et de vérification : S et V.
- Tous les fichiers sont protégés.
- Le système d'exploitation des quatre machines est sûr.

Cette dernière affirmation sous-entend que les fichiers sont en sécurité une fois présents dans une mémoire primaire, et que l'intégrité de H et de V est assurée. En corollaire, nous pouvons affirmer que dans un tel système, les attaques ne peuvent avoir lieu que pendant le stockage ou le transfert d'un fichier. Tous ces éléments forment le cas n°1.

La première hypothèse, quant à elle, sera conservée tout au long de ce chapitre. Nous avons recherché des fonctions d'authentification qui restent sûres dans un contexte où l'ennemi en connaît tous les détails. Il n'y a donc aucun problème de confidentialité, et la même fonction d'authentifiation peut être facilement distribuée dans tout le système. Une version matérielle performante peut par exemple être conçue et installée sur les différentes machines.

Pour tous les cas décrits ci-dessous, la distribution d'un nouveau fichier F se déroule toujours de la même façon. Le CAD calcule un CAF et il le signe. Ensuite, F et S(CAF) sont envoyés vers l'une ou l'autre machine où ils sont stockés en mémoire secondaire. Une version de F est conservée en toute sécurité au niveau du CAD.

L'association d'un fichier F et de son S(CAF) peut prendre plusieurs formes, parmi lesquelles:

- La concaténation de S(CAF) au début ou à la fin de F <MEYER, 88>. Cette solution a toutefois le désavantage de modifier le fichier. Or, l'un des critères pour lesquels nous avions choisi les codes d'authentification était justement le respect de l'intégrité des fichiers.
- La création d'un autre fichier contenant un identifiant de F, S(CAF) et éventuellement d'autres renseignements tels que la date de création, le propriétaire, le numéro de version, ... Nous appelerons ce fichier 'fichier d'accompagnement'.

3.1. LE CAS N°1

Dans une situation caractérisée par les hypothèses décrites ci-dessus, nous proposons le protocole d'authentification suivant :

- Lorsque F doit être utilisé, il est transféré, de même que S(CAF), de la mémoire secondaire à la mémoire primaire.

- Le CAF de F est calculé et est comparé à V(S(CAF)).

- En cas de différence entre ces deux valeurs, une nouvelle paire F - S(CAF) est demandée au CAD et une enquête est menée pour déterminer l'origine de la modification.

Remarquons qu'une paire F - S(CAF) peut sans problème être transférée d'un endroit à un autre comme ce sera fréquemment le cas entre un ordinateur et son micro associé, opération connue sous le terme de 'téléchargement'. Sur la machine de réception, le fichier transféré sera authentifié selon le protocole déjà exposé ci-dessous.

Notons enfin que dans le cas n°1, un ennemi ne peut installer un nouveau fichier. La fonction de signature lui étant inconnue, il ne pourra pas de ce fait créer un S(CAF) valable pour le fichier en question.

3.2. LE CAS N°2

Dans le cas n°1, nous avons supposé que tous les fichiers du système informatique étaient protégés. Cependant, dans la réalité, cela n'est pas toujours possible, comme on l'explique dans <POZZO,86>:

- Tout système informatique ne possède pas un CAD indépendant et sûr. Cela signifie que les logiciels en phase de développement encourent les mêmes risques que les fichiers installés. Protéger ces logiciels par authentification n'est toutefois pas une bonne solution, car leur 'mise au point' nécessite de nombreuses modifications. Une vérification systématique de l'intégrité serait ici trop coûteuse en temps. Nous pensons qu'un développement 'sans protection' suivi d'une analyse du résultat par une autorité de contrôle semble dans ce cas mieux adapté.

- Dans la plupart des systèmes informatiques, les utilisateurs peuvent créer leurs propres fichiers. Il semble déraisonnable d'imposer que ceux-ci soient systématiquement soumis aux mêmes contraintes que les fichiers 'officiels' du système. N'oublions pas en effet que tout fichier protégé est sensé être

authentique. Cela implique un examen minutieux par les responsables du CAD. Exiger un tel travail pour tous les fichiers du système peut être soit impossible, soit excessif au vu du degré de sécurité requis par le système.

Si nous permettons la cohabitation entre fichiers protégés et fichiers non protégés, nous ouvrons la porte à une fraude évidente : un ennemi peut faire en sorte que le fichier qu'il vient de modifier prenne le statut 'non protégé'. Puisque dans ce cas aucune vérification n'est effectuée avant l'exécution, la fraude ne sera pas détectée.

Un moyen de se prémunir contre une telle attaque est d'enregistrer toutes les informations relatives aux fichiers à protéger dans une liste qui est distribuée par le CAD <POZZO, 86>. Cette liste est protégée par un CAF signé. Elle est vérifiée une fois arrivée en mémoire primaire où elle reste bloquée si elle n'a subi aucune modification durant son transfert.

Dans ces conditions, le protocole d'authentification d'un fichier devient le suivant :

- Vérifier si le fichier appartient à la liste.
- Si non, ne rien faire.
- Si oui, appliquer le protocole décrit dans le cas n°1.

Quand le CAD ajoute, supprime ou modifie un fichier qui doit être protégé, il met à jour la liste et la distribue à toutes les machines.

Bien entendu, avec un tel protocole, rien n'empêche un ennemi de créer et d'utiliser un nouveau fichier. Rappelons toutefois que l'objectif de ce mémoire n'était pas de combattre ce type de fraude.

L'admission de fichiers non protégés pose de gros problèmes et, dans la mesure du possible, il faudra l'éviter. De plus, l'hypothèse de sécurité des fichiers en mémoire primaire est difficilement conciliable avec l'exécution d'instuctions éventuellement frauduleuses.

3.3. LE CAS N°3

Dans les deux premiers cas, nous avons supposé qu'il n'existait qu'une seule paire S - V. Cette hypothèse peut parfois être trop contraignante. Nous pensons tout particulièrement à un système réparti où il existe plusieurs CAD. L'obligation de centraliser l'examen et la génération des CAF signés peut entraîner des lourdeurs inadmissibles dans le processus de distribution des fichiers.

Dans ces conditions, plusieurs CAD devraient avoir la possibilité de générer des CAF signés. Deux optiques se présentent alors :

- Soit tous les CAD utilisent la même fonction de signature, ce qui entraîne la nécessité de la distribuer de manière sûre (rappelons que la confidentialité de la fonction de signature est une condition nécessaire à la sécurité du système). Une fois le problème de la distribution résolu, nous en revenons au cas n°2.

- Soit chaque CAD utilise sa propre fonction de signature <POZZO,86>.

Dans la seconde optique, il est nécessaire qu'un système d'authentification sache quelle fonction de vérification employer. On peut très bien imaginer résoudre ce problème en ajoutant un identifiant du signataire au 'fichier d'accompagnement' évoqué au point 3.1. de ce chapitre. En plus de cela, une liste des paires (identifiant du signataire - fonction de vérification) doit être conçue par une personne de confiance qui ensuite la protége et la distribue dans tout le système. Quand la liste arrive dans la mémoire primaire d'une machine, elle y est bloquée si la vérification d'intégrité est positive.

Le protocole d'authentification d'un fichier devient maintenant le suivant :

- accéder à la liste des fichiers à protéger. Si le fichier ne s'y trouve pas, l'authentification est terminée. Dans le cas contraire, passer à l'étape suivante.

- accéder au fichier d'accompagnement du fichier et y extraire l'identifiant du signataire et S(CAF).

- accéder à la liste des fonctions de vérification et extraire celle qui correspond à l'identifiant obtenu à l'étape précédente. Si elle ne s'y trouve pas, l'authentification échoue. Si elle s'y trouve, appliquer le protocole décrit au cas n°1.

Le protocole décrit ci-dessus ne résoud pas entièrement le problème de la centralisation. En effet, il est obligatoire de concentrer en un seul endroit la mise à jour de la liste des fonctions de vérification acceptables et celle de la liste des fichiers à protéger. La première mise à jour ne devrait pas poser trop de problèmes car, en principe, l'ensemble des entités autorisées à émettre des CAF signés ne varie qu'épisodiquement. Par contre, la liste des fichiers à protéger risque d'être modifiée souvent dans un système de grande importance.

Nous tirerons deux enseignements de cette constatation. Le premier est qu'il est beaucoup plus facile de protéger un système de petite taille dont tous les paramètres peuvent

être connus et gérés de manière centralisée. Le deuxième, déjà évoqué précédemment, est qu'il est très délicat de devoir gérer une liste des fichiers à protéger. Un système où tous les fichiers sont authentifiés est infiniment plus simple à concevoir.

3.4. LE CAS N°4

Nous allons maintenant lever l'hypothèse de la sécurité des systèmes d'exploitation. Ceci est très contraignant, car maintenant nous ne pouvons plus nous reposer sur un environnement sûr pour effectuer la vérification des fichiers.

Mais devons-nous forcément nous passer de cette hypothèse si pratique? Nous avons affirmé que oui dans la première partie en nous basant notamment sur l'exemple des modifications dues aux virus. La récente attaque informatique contre le réseau NASA-SPANet durant laquelle plusieurs systèmes d'exploitation VMS ont été altérés ne peut que confirmer nos craintes <GLISS, 88>.

Nous ne trancherons pourtant pas la question de la sécurité ou non des systèmes d'exploitation. Nous nous contenterons de citer un document publié par le "Department of Defense" des Etats-Unis qui fixe des critères d'évaluation de la sécurité des systèmes informatiques. Ces critères définissent quatre niveaux de sécurité ordonnés de manière décroissante : A, B, C, D <DoD, 83> .

Un système d'exploitation correspondant à une marque A peut sans doute être considéré comme un environnement sûr. Nous laissons à l'appréciation du constructeur d'un système d'authentification le soin de décider si les niveaux B ou même C méritent une telle confiance. Pour le niveau D, il est fortement conseillé d'être plus que prudent.

Dans la suite de cette section, nous examinerons les conséquences introduites par la levée de l'hypothèse de sécurité des systèmes d'exploitation. Dans ce cas, nous ne pouvons plus supposer que les fichiers sont en sécurité dans la mémoire primaire d'une machine, ni que l'intégrité de H et de V est assurée. Les protocoles décrits précédemment doivent dès lors être revus :

- Comme nous l'avons montré, la liste des fichiers à protéger doit être maintenue intègre. Une solution consiste à authentifier cette liste avant chaque consultation <POZZO, 86>. Si elle a subi des manipulations, toute authentification de fichier est refusée jusqu'à la correction de la liste. Une enquête est menée pour déterminer l'origine de la modification.

- La liste des fonctions de vérification doit également être préservée de toute altération. Le principe de protection décrit ci-dessus pour la liste des fichiers

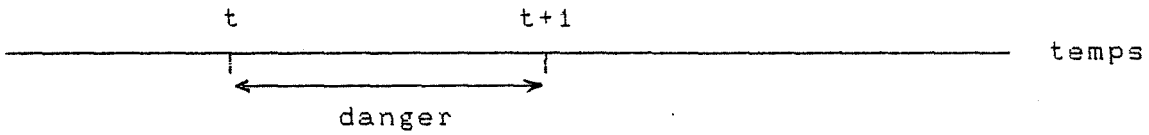
à protéger peut lui être appliqué <POZZO, 86>.

Pour ces deux contrôles d'intégrité, la même fonction de vérification (propre à l'autorité 'de confiance' gérant les listes) est utilisée par toutes les machines du système.

Malgré ces précautions, un tel système d'authentification reste sensible à un certain nombre d'attaques :

La première d'entre elles, sans doute la plus difficile à contrer, se base sur une constatation très simple, à savoir que l'authentification d'un fichier n'apporte qu'un résultat ponctuel : nous pouvons être certains à un moment t correspondant à la fin de l'authentification qu'un fichier est dépourvu de toute modification. Ce fichier est alors employé à une tâche quelconque qui se termine au moment t+1. Le problème réside ici dans le fait qu'entre t et t+1, le fichier a peut-être subi des manipulations <QUISQUATER, 88>. Ceci est illustré par la figure 5.2.

FIGURE 5.2 : L'authentification et l'utilisation d'un fichier



t : fin de l'authentification d'un fichier
t+1 : fin de l'utilisation de ce même fichier

A titre d'exemple, dans notre description d'un système d'authentification, rien n'empêche qu'un ennemi modifie la liste des fichiers à protéger entre le moment où elle est déclarée intègre et celui où le processus de recherche d'un fichier dans cette liste se termine.

La seconde attaque, plus simple à concevoir, consiste à modifier directement la fonction d'authentification.

Nous pouvons donc dès à présent formuler une première conclusion:

S'il n'est pas possible de faire confiance à l'environnement dans lequel va se dérouler l'authentification et l'utilisation des fichiers, il est impossible de définir un protocole d'authentification de fichiers qui soit sûr.

Dans ces conditions, la construction d'un système d'authentification sûr ne peut passer que par l'utilisation de composants matériels résistant aux manipulations. Nous étudierons ceux-ci au chapitre suivant.

4. LES 'OUTILS' DONT DISPOSE LE CONSTRUCTEUR

Le constructeur d'un système d'authentification dispose de divers 'outils' qui lui permettent de pallier l'insuffisance de sécurité de certains systèmes informatiques. Ces outils peuvent tous être regroupés sous le terme générique 'Tamper-Resistant Modules' (TRM). Dans la littérature, on rencontre aussi la dénomination 'security modules' ou 'cryptographic security modules' <WOOD, 86>.

Un TRM est un composant matériel contenant des possibilités de stockage et de calcul, et qui protège ceux-ci des manipulations frauduleuses ou accidentelles provenant de l'extérieur <WOOD, 86>. Il constitue une unité sûre d'un point de vue physique : il n'y a pas moyen de connaître ce qui se trouve à l'intérieur, et toute tentative pour y pénétrer a pour conséquence la destruction totale du contenu. Il est également sûr d'un point de vue logique au sens où les applications qu'il renferme sont telles qu'elles ne fournissent aucun résultat qui permettrait de découvrir des informations contenues à l'intérieur <DAVIES, 84, 80>.

L'utilité des TRM est évidente : *"Firms use security modules principally to prevent employees and other potential system attackers from gaining direct access to, or from modifying security parameters or other important security data (and...) programs"* <WOOD, 86>.

Grâce au concept de TRM, le constructeur d'un système d'authentification dispose d'un endroit où il peut stocker des données et effectuer des opérations en toute sécurité.

Les TRM ont pourtant deux défauts : ils sont chers et de faible capacité <FERREIRA, 88>. De ce fait, leur usage est souvent restreint à la protection de la confidentialité de quelques paramètres secrets et de l'intégrité d'une application relative à la sécurité. Ils sont par exemple l'outil idéal pour assurer la protection de la procédure de génération des S(CAF). Un TRM contenant H et la fonction de signature peut être conçu pour accepter en paramètre un fichier et produire en résultat un CAF signé pour ce fichier. Une première phase d'identification permet d'assurer que seule une personne 'de confiance' ait accès à la procédure. La sécurité physique et logique du TRM garantit la protection de l'intégrité de H et de la confidentialité de S.

On peut également implémenter au sein d'un TRM la procédure de vérification des S(CAF). Le stockage sûr de la liste des fichiers à protéger et de la liste des fonctions de vérification valables est assuré. L'intégrité de la fonction d'authentification l'est autant.

Il existe au niveau expérimental des TRM de beaucoup plus grande capacité, dénommés 'protected processor' <WHITE, 87> ou 'secure processor' <MEYER, 88>. Il s'agit de TRM contenant un système informatique complet : un processeur, une horloge et un générateur de nombres pseudo-aléatoires. Ils disposent en plus de suffisamment de mémoire pour pouvoir stocker les applications

et leurs données durant l'exécution. Une mémoire non volatile permet de conserver en toute sécurité bon nombre de paramètres et autres données nécessaires aux applications, et ce même lorsque le système est hors tension <WHITE, 87>.

Il existe une différence fondamentale entre un TRM classique et un 'protected processor'. Le TRM est utilisé par un processeur comme un périphérique capable d'effectuer un certain nombre d'opérations nécessitant un environnement sûr. Le 'protected processor' quant à lui, peut fonctionner de manière indépendante. Il est possible d'y implémenter une procédure de vérification des fichiers et d'y faire s'exécuter les applications matérialisées par ces mêmes fichiers.

Grâce à sa nature fondamentalement 'périphérique', le TRM peut être ajouté à un système préexistant sans pour autant modifier ce dernier <CARLSON, 86>. Malheureusement, cette approche est insuffisante dans le cadre de l'authentification. Lors de la vérification d'intégrité, rien n'empêche en effet un ennemi :

- de présenter au TRM un fichier authentique, et de lui substituer ensuite un fichier frauduleux juste avant son utilisation par un processeur,

- de présenter au TRM un fichier frauduleux, et de changer le résultat 'fichier non intègre' en 'fichier intègre',

- ou encore beaucoup plus simplement de courcircuiter le passage par le TRM.

Ceci nous amène à conclure que :

Pour qu'un système d'authentification de fichier soit sûr, il faut que la décision d'intégrité et l'utilisation du fichier soient réalisées à l'intérieur d'un environnement protégé de toute manipulation frauduleuse <FERREIRA, 88>, <MEYER, 88>, <POZZO,86>. **Si cet environnement sûr n'existe pas, il peut être créé grâce au concept de 'secure processor' décrit ci-dessus.**

Pour illustrer ceci, revenons au système informatique décrit à la figure 5.1. Il sera courant de supposer que les deux ordinateurs sont sûrs. Par contre, les micro-ordinateurs possèdent de faibles défenses et constituent rarement un environnement exempt de tout soupçon <HIGHLAND, 85>. Dans ces conditions, si nous associons à chaque micro-ordinateur un 'secure processor', le protocole d'authentification décrit au point 3.3. de la présente partie peut être considéré comme sûr.

Un autre cas de figure correspond à la situation où les ordinateurs sont eux aussi suspects. On peut alors leur associer un 'secure processor', bien que cela risque d'avoir un effet très défavorable sur les performances.

Nous n'affirmons toutefois pas qu'il faut du jour au lendemain remplacer tous les processeurs par des processeurs sûrs. Comme nous le verrons dans le chapitre suivant, ce type de transformation et le protocole qui lui est associé entraînent bon nombre de contraintes qui limitent leur champ d'application.

5. LE CHAMP D'APPLICATION DU SYSTEME D'AUTHENTIFICATION PROPOSE

Nous sommes conscients des limites du système d'authentification de fichiers proposé dans ce mémoire. Dans sa version la plus complexe, c'est-à-dire celle décrite au point 3.4. de la présente partie, on peut lui faire les reproches suivants :

- Il nécessite une **nouvelle technologie**, celle du 'protected processor'. Il sera difficile pour les utilisateurs de délaisser les techniques standards actuelles qui ont fait leur preuve. Il existe peut-être un compromis consistant à sécuriser les environnements d'exécution existants. Telle est l'optique suivie par POZZO et GRAY qui proposent un protocole de protection d'un système d'exploitation. Celui-ci consiste à vérifier l'intégrité du système au moment de son lancement et à empêcher tout accès en lecture et en écriture durant son exécution <POZZO, 86>.

- Il constitue un **frein aux performances**, puisque toute utilisation d'un fichier protégé nécessite l'accès à plusieurs listes en plus de l'authentification proprement dit. Il paraît évident qu'un utilisateur ne tolèrera jamais de devoir attendre plusieurs dizaines de minutes avant l'exécution d'un programme alors que sans protection l'attente serait inférieure à une seconde. L'utilisation d'une version matérielle très performante des fonctions d'authentification et de vérification de signature peut aider à minimiser cet inconvénient.

- Il représente une **source de coûts**. Ceux-ci sont notamment dûs aux besoins de technologies cités ci-dessus, mais également aux frais nécessaires à la mise en oeuvre du protocole décrit au point 3.3. de la présente partie.

  Ces deux derniers points sont à évaluer en se référant à l'affirmation suivante : "En général, aucun utilisateur n'est prêt à accepter une perte de performance et un accroissement de coût de plus de 10% uniquement pour des mesures de sécurité" <FERREIRA, 88>.

- Enfin, il est soumis à de **nombreuses contraintes** parmi lesquelles :

  * L'obligation de posséder un ou plusieurs CAD ou entités équivalentes qui centralisent la production ou l'achat de nouveaux fichiers à protéger.

  * L'obligation de passer par un CAD afin d'apporter une modification 'légale' à un fichier protégé. Cette modification doit être contrôlée afin d'assurer qu'elle n'introduit aucune instruction

frauduleuse. Cette contrainte est très lourde si les fichiers sont régulièrement mis à jour.

* L'obligation de gérer des paramètres tels la liste des fichiers à protéger, ou la liste des fonctions de vérification valables, de même que l'obligation d'instaurer un contrôle des nouveaux fichiers et des fichiers modifiés. Cette gestion et ce contrôle nécessitent des personnes 'de confiance', car une corruption à ce niveau est l'une des manières les plus simples d'introduire des fichiers frauduleux dans le système informatique. Une vérification impliquant plusieurs personnes peut diminuer le risque de fraude, mais augmente les coûts de traitement.

Ces inconvénients ont pour conséquence de restreindre le champ d'application du système d'authentification proposé. Il nous semble raisonnable à l'heure actuelle de limiter son utilisation aux systèmes informatiques :

- où une politique de contrôle et de maintien de la sécurité peut être instaurée efficacement

- où la sécurité est aussi importante que les performances, et ce à cause des sommes mises en jeux

- où les fichiers sont en nombre limité et possèdent un taux de mise à jour faible

De tels systèmes existent dans le domaine financier et industriel, mais il nous est difficile à priori d'en donner de plus amples caractéristiques. Seule une analyse minutieuse pourrait préciser les modalités d'installation d'un système d'authentification dans une configuration informatique donnée.

CONCLUSION

Nous avons consacré ce mémoire à la protection de l'intégrité des fichiers. Nous avons choisi de l'aborder en utilisant le concept de fonction d'authentification.

La description d'un modèle de fonction d'authentification et de l'étude des attaques possibles nous ont permis de constituer le 'portrait robot' de la fonction idéale :

- Elle génère un code d'authentification d'au moins 128 bits.
- Elle possède des résultats intermédiaires de même longueur.
- Elle est structurée de manière à empêcher une attaque autre qu'exhaustive.
- Sa sécurité repose sur le moins d'éléments secrets possible.
- De plus, elle est très rapide.

Cette fonction sera encore plus attractive si elle est reconnue par les organes internationaux de standardisation.

A l'aide de ces critères, nous avons analysé quelques-uns des algorithmes de fonctions d'authentification publiés dans la littérature.

Nous avons mis en évidence le fait que bon nombre de ces fonctions ne pouvaient pas être qualifiées de 'bonne', soit parce qu'elles n'étaient pas assez sûres, soit parce qu'elles n'étaient pas assez performantes.

Quatre algorithmes prometteurs ont été étudiés davantage en profondeur. Nous pouvons considérer que trois d'entre eux (MM82, MS88 et QU88) possèdent un niveau de sécurité satisfaisant. Au vu des très nombreuses fonctions qui ont été cassées ces dernières années, nous conseillons toutefois au lecteur qui désirerait les utiliser de se maintenir au courant des derniers progrès en matière de cryptanalyse.

La quatrième fonction, le MAA, ne peut pas rivaliser avec la sécurité offerte par les trois autres précitées. Elle possède toutefois l'avantage d'être très rapide (100 Ko/s sur un processeur Intel 80386 à 16 MHz). De plus, notre expérience nous permet d'affirmer que sa mise au point sur la plupart des processeurs classiques ne pose pas de problèmes majeurs.

Nous la conseillons donc à toute personne désirant obtenir à moindre frais un système d'authentification pour les fichiers résidant sur le disque dur d'un micro-ordinateur. La fonction, la clé et les codes d'authentification doivent être placés sur une disquette qui sera protégée contre toute altération (l'usage d'un coffre fort peut notamment constituer une solution).

Le choix de ces quatre algorithmes, s'il n'est pas arbitraire, n'en reste pas moins limité. Nous aurions voulu en analyser d'autres, et tout particulièrement ceux qui favorisent la sécurité au détriment des performances. Une implémentation matérielle efficace pourrait peut-être les rendre plus attractives.

Pour toutes les fonctions non analysées et les nouvelles qui sans aucun doute seront publiées, nous ambitionnons d'avoir donné les éléments nécessaires à une approche critique de leur sécurité et de leurs performances.

De plus, nous espérons avoir fait comprendre au lecteur que si la fonction d'authentification parfaite n'existera probablement jamais, il est tout à fait possible d'en trouver une qui réponde aux besoins d'une application particulière. Le cas du MAA pour une 'authentification maison' sur micro-ordinateur est le meilleur exemple qui soit.

Enfin, rappelons que les fonctions d'authentification ont une autre utilisation que la protection des fichiers au sein d'un système informatique.

En effet, elles peuvent être employées pour la condensation de messages avant leur signature numérique. Cette technique offre l'avantage d'augmenter les performances du processus de génération de la signature et celui de réduire la taille du résultat. De plus, la condensation du message a la propriété d'augmenter la sécurité de ce même processus en empêchant certaines attaques <WINTERNITZ, 84a>.

Les fonctions nécessaires à la génération de ces condensats de message doivent respecter les mêmes critères que ceux que nous avons définis pour les fonctions d'authentification. C'est pourquoi les analyses, exemples et résultats présents dans les parties deux, trois et quatre restent valables pour ce type d'application.

Dans la cinquième partie de ce mémoire, nous avons proposé l'ébauche d'un système mettant en oeuvre une fonction d'authentification afin de protéger les fichiers d'un système informatique 'standard' que nous avons préalablement défini.

Nous n'avons pas la prétention d'avoir ainsi résolu de façon universelle le problème du besoin d'intégrité des fichiers. Tout au plus, avons-nous montré comment un système d'authentification pouvait être conçu en fonction des caractéristiques propres à un certain nombre de situations qu'il est possible de rencontrer dans l'informatique actuelle.

Notre proposition est loin d'être complète. Elle n'offre notamment aucune solution au problème de la protection de

l'intégrité des fichiers mis à jour régulièrement. A titre d'exemple, nous pouvons citer le cas d'un fichier contenant les transactions journalières d'une agence bancaire.

Un tel fichier doit être protégé. Mais puisqu'il est sans cesse modifié, les fonctions d'authentification ne peuvent lui apporter l'assurance d'intégrité souhaitée.

Toutefois, un bon niveau de sécurité peut être obtenu si les mesures suivantes sont prises :

- Assurer l'intégrité de tous les programmes qui sont impliqués dans la manipulation de ce fichier de transaction.

- Maintenir un environnement sûr de sorte que le fichier de transaction ne puisse être modifié durant son utilisation par un processeur.

- Si le fichier de transaction doit sortir du contexte sécurisé (pour être transféré ou stocké par exemple), générer un code d'authentification signé qui l'accompagnera et permettra de vérifier s'il a subi des modifications durant son exposition dans le milieu non sûr.

Cette solution possède malheureusement quelques inconvénients. Le principal est la nécessité de générer localement un code d'authentification signé. Or, nous avons montré que cette production doit être entourée de la plus grande protection :

- il faut éviter que la fonction d'authentification et celle de signature ne subisse des altérations.

- il faut en refuser l'accès à tout ennemi désirant obtenir un code d'authentification signé pour un fichier frauduleux.

Suite à cette réflexion, il est normal de se demander s'il n'est pas plus simple de retreindre la protection aux fichiers 'stables', et en particulier aux programmes. Tel est en tout cas l'avis émis par les quelques auteurs qui ont proposé l'utilisation de fonctions d'authentification <POZZO, 86>, <MEYER, 88>.

Cette optique restrictive est compréhensive. Les programmes sont en effet les éléments moteurs d'un système informatique. Falsifier un chèque peut rapporter beaucoup, mais modifier le programme qui génère ces chèques rapportera certainement beaucoup plus.

Rappelons également que les virus informatiques, probablement l'un des grands fléaux des années à venir, se propagent par infection de programmes. En empêchant l'exécution de tout

programme modifié, on endigue l'infection et on localise les virus, ce qui permet de les éliminer.

L'ébauche de système d'authentification proposée dans ce mémoire peut être également comprise comme une réflexion sur la sécurité des systèmes informatiques. A ce titre, nous pouvons en retirer plusieurs enseignements intéressants :

- Tout d'abord, il est illusoire de croire qu'avec une fonction d'authentification, fusse-t-elle parfaite, le problème de la sécurité des fichiers va être résolu du jour au lendemain. La sécurité est un concept complexe mettant en oeuvre des hommes, des machines, des programmes et des données. L'usage de fonctions d'authentification ne peut être qu'un maillon de la chaîne.

- Il est important d'assurer un environnement sûr à tout processus relevant de la sécurité. Cet environnement peut être constitué des protections classiques offertes par les systèmes d'exploitation. Cette solution est bien entendu la plus simple à mettre en oeuvre, mais elle ne peut être raisonnablement choisie que s'il est plus facile à l'ennemi de s'attaquer au processus de sécurité lui-même qu'au système d'exploitation.

  A l'autre extrémité, l'environnement sûr peut être créé de toute pièce en utilisant le concept de 'secure processor'. Cette solution possède néanmoins de nombreux désavantages dont la non standardisation, le coût élevé et les faibles performances.

- Enfin, il est beaucoup plus facile de mettre en oeuvre la protection d'un système informatique de taille réduite et dont on maîtrise tous les paramètres. A ce titre, une sécurité couvrant tous les aspects du système sera certainement plus efficace.

  Il suffit pour s'en convaincre de se rapporter aux difficultés introduites par le fait de ne pas authentifier systématiquement tous les fichiers. Ceci a malheureusement pour conséquence d'ôter à l'ensemble du système sa souplesse naturelle.

Cette réflexion nous permet de mettre une fois de plus l'accent sur la complexité de traitement introduite par l'ajout de mesures de sécurité. Nous comprenons alors peut-être mieux pourquoi bien souvent ces mesures sont négligées en faveur de l'efficacité opérationnelle.

Une position intermédiaire entre la protection totale et le refus de la sécurité peut sans doute être trouvée. Elle doit résulter d'un équilibre entre le niveau de sécurité et les

performances qui y sont associées. Et comme le fait remarquer Monsieur FERREIRA, *"the level of security appropriate can only be optimised in an application specific context"*.

C'est pourquoi une extension possible de ce mémoire pourrait être l'étude de la construction d'un système d'authentification de fichier dans un environnement typique - tel une station de travail ou un réseau local - et pour une application bien précise.

Par ce mémoire, nous espérons avoir contribué à faire prendre conscience au lecteur que la sécurité est à l'heure actuelle un problème majeur. L'authentification des fichiers peut aider à résoudre ce problème à condition d'être intégrée le plus tôt possible dans le processus de construction des nouveaux systèmes informatiques.

ANNEXES

ANNEXE 1

UN PSEUDO LANGAGE DE DESCRIPTION DE PROGRAMMES

Nous décrivons dans cette annexe un pseudo langage qui est utilisé au point 2.2.3. de la première partie pour décrire un exemple simpliste de virus. Ce pseudo langage est extrait de <COHEN, 87>.

- Le symbole := exprime une définition.
- Le symbole : exprime une étiquette.
- Le symbole ; termine toute instruction.
- Le symbole = est utilisé à la fois pour exprimer l'assignation ou la comparaison.
- Les symboles { et } délimitent des blocs d'instructions.
- Le symbole ... indique qu'une portion du code a été laissé implicite.

ANNEXE 2

LE DATA ENCRYPTION STANDARD

Dans cette annexe, nous reproduisons le texte de la publication des spécifications du Data Encryption Standard, mieux connu sous l'abréviation de DES.

Cet algorithme de chiffrement symétrique est d'une importance capitale pour la sécurité des données. Il s'agit en effet d'un des rares algorithmes qui soit à la fois rapide et sûr. Originellement prévu pour être utilisé jusqu'en 1988, le DES vient de recevoir l'aval des experts de l'administration américaine pour cinq nouvelles années, ce qui est un signe évident de qualité <FERREIRA, NP>.

Nous conseillons vivement à toute personne non initiée de parcourir cette spécification, ce qui l'aidera certainement à mieux comprendre quelques passages de ce mémoire.

Federal Information
Processing Standards Publication 46

1977 January 15

SPECIFICATIONS FOR THE

DATA ENCRYPTION STANDARD

The Data Encryption Standard (DES) shall consist of the following Data Encryption Algorithm to be implemented in special purpose electronic devices. These devices shall be designed in such a way that they may be used in a computer system or network to provide cryptographic protection to binary coded data. The method of implementation will depend on the application and environment. The devices shall be implemented in such a way that they may be tested and validated as accurately performing the transformations specified in the following algorithm.

DATA ENCRYPTION ALGORITHM

Introduction

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. The key-dependent computation can be simply defined in terms of a function f, called the cipher function, and a function KS, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function f is given in terms of primitive functions which are called the selection functions S_i and the permutation function P. S_i, P and KS of the algorithm are contained in the Appendix.

The following notation is convenient: Given two blocks L and R of bits, LR denotes the block consisting of the bits of L followed by the bits of R. Since

concatenation is associative $B_1B_2 \ldots B_8$, for example, denotes the block consisting of the bits of B_1 followed by the bits of B_2 . . . followed by the bits of B_8.

Enciphering

A sketch of the enciphering computation is given in figure 1.

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation *IP*:

IP

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

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described below. The output of that computation, called the preoutput, is then subjected to the following permutation which is the inverse of the initial permutation:

IP^{-1}

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

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

The computation which uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of a calculation that is described below in terms of the cipher function f which operates on two blocks, one of 32 bits and one of 48 bits, and produces a block of 32 bits.

Let the 64 bits of the input block to an iteration consist of a 32-bit block L followed by a 32-bit block R. Using the notation defined in the introduction, the input block is then LR.

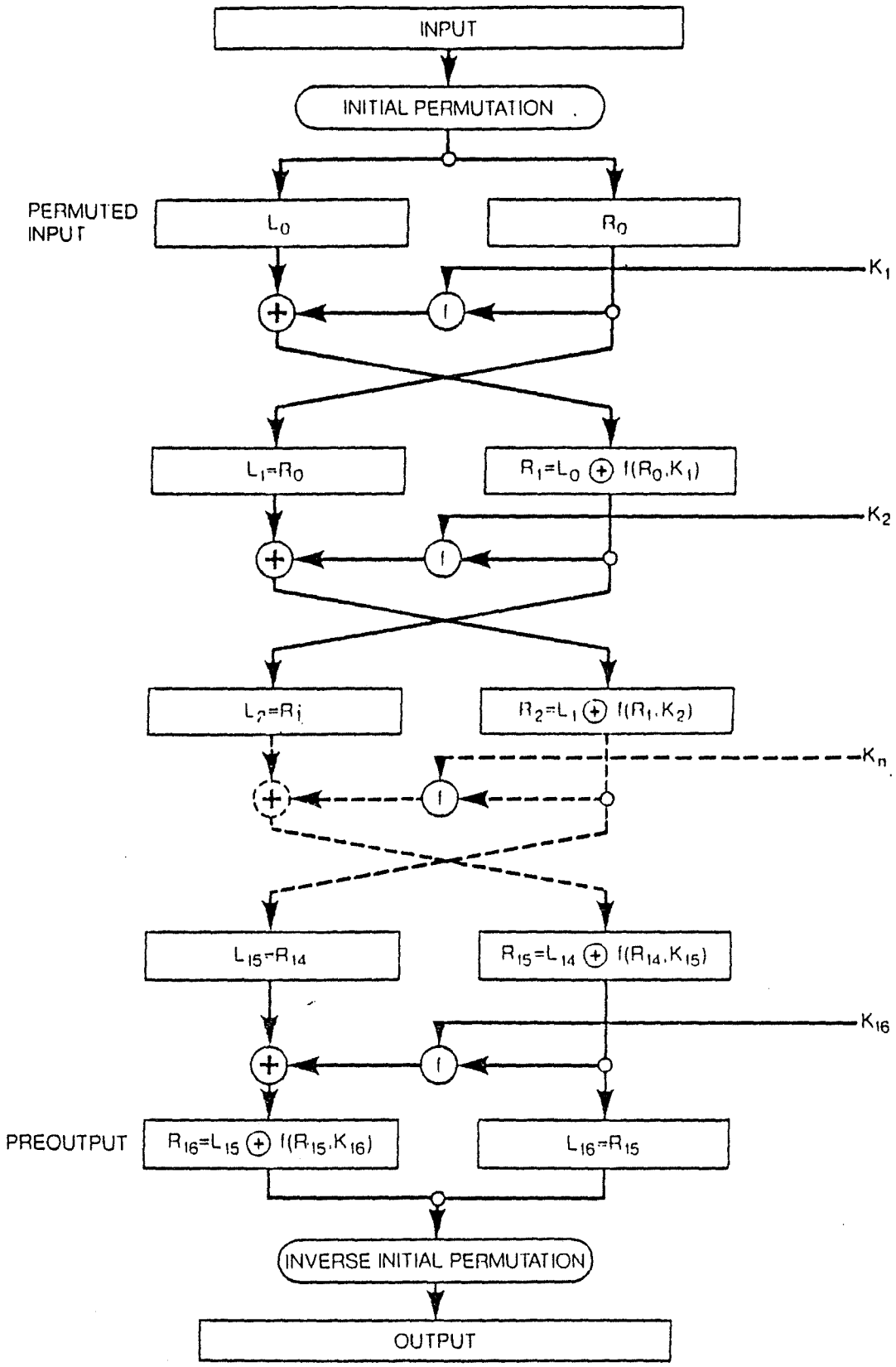


Figure 1 Enciphering computation

Let K be a block of 48 bits chosen from the 64-bit key. Then the output $L'R'$ of an iteration with input LR is defined by:

(1) $$L' = R$$
$$R' = L \oplus f(R, K)$$

where $\oplus$ denotes bit-by-bit addition modulo 2.

As remarked before, the input of the first iteration of the calculation is the permuted input block. If $L'R'$ is the output of the 16th iteration then $R'L'$ is the preoutput block. At each iteration a different block K of key bits is chosen from the 64-bit key designated by KEY.

With more notation we can describe the iterations of the computation in more detail. Let KS be a function which takes an integer n in the range from 1 to 16 and a 64-bit block KEY as input and yields as output a 48-bit block K_n which is a permuted selection of bits from KEY. That is

(2) $$K_n = KS(n, KEY)$$

with K_n determined by the bits in 48 distinct bit positions of KEY. KS is called the key schedule because the block K used in the n'th iteration of (1) is the block K_n determined by (2).

As before, let the permuted input block be LR. Finally, let L_0 and R_0 be respectively L and R and let L_n and R_n be respectively L' and R' of (1) when L and R are respectively L_{n-1} and R_{n-1} and K is K_n; that is, when n is in the range from 1 to 16,

(3) $$L_n = R_{n-1}$$
$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

The preoutput block is then $R_{16}L_{16}$.

The key schedule KS of the algorithm is described in detail in the Appendix. The key schedule produces the 16 K_n which are required for the algorithm.

Deciphering

The permutation IP^{-1} applied to the preoutput block is the inverse of the initial permutation IP applied to the input. Further, from (1) it follows that:

(4) $$R = L'$$
$$L = R' \oplus f(L', K)$$

Consequently, to *decipher* it is only necessary to apply the *very same algorithm to an enciphered message block*, taking care that at each iteration of the computation *the same block of key bits* K *is used* during decipherment as was used during the encipherment of the block. Using the notation of the previous section, this can be expressed by the equations:

(5) $$R_{n-1} = L_n$$
$$L_{n-1} = R_n \oplus f(L_n, K_n)$$

where now $R_{16}L_{16}$ is the permuted input block for the deciphering calculation and L_0R_0 is the preoutput block. That is, for the decipherment calculation with $R_{16}L_{16}$ as the permuted input, K_{16} is used in the first iteration, K_{15} in the second, and so on, with K_1 used in the 16th iteration.

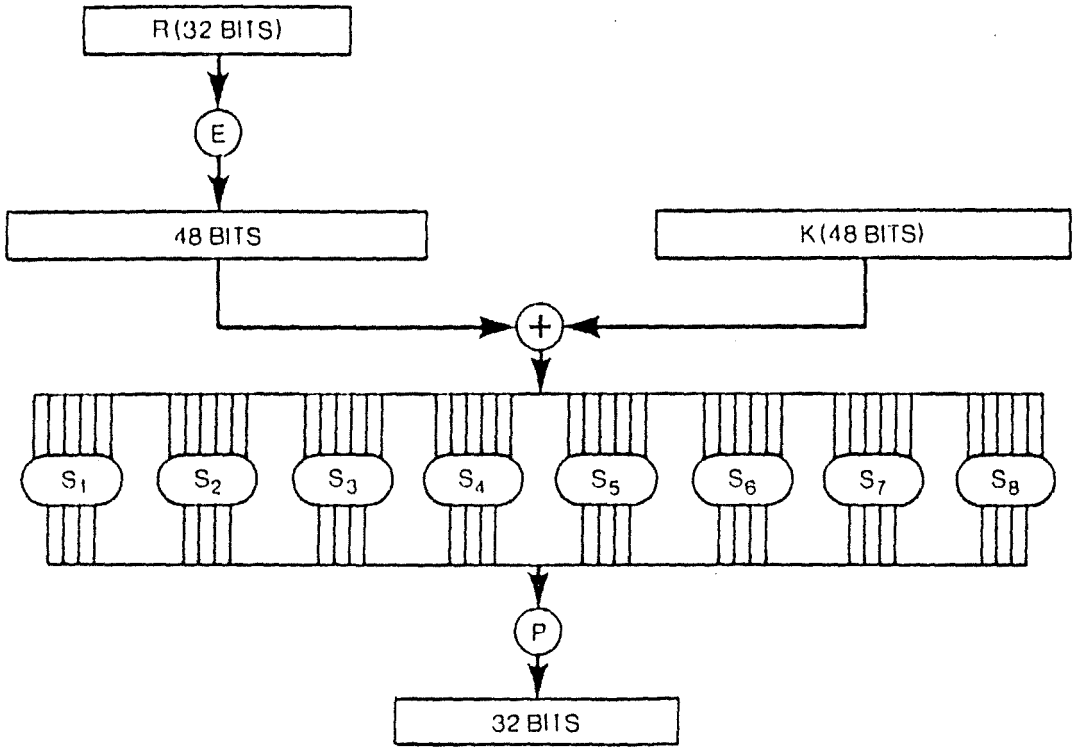


Figure 2 Calculation of $f(R, K)$

The Cipher Function f

A sketch of the calculation of $f(R, K)$ is given in figure 2.

Let E denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

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

Thus the first three bits of $E(R)$ are the bits in positions 32, 1 and 2 of R while the last 2 bits of $E(R)$ are the bits in positions 32 and 1.

Each of the unique selection functions $S_1, S_2, \ldots, S_8$, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using a table containing the recommended S_1:

S_1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B represent in base 2 a number in the range 0 to 15. Let that number be j. Look up in the table the number in the i'th row and j'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101. Selection functions $S_1, S_2, \ldots, S_8$ of the algorithm appear in the Appendix.

The permutation function P yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following table:

P

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

The output $P(L)$ for the function P defined by this table is obtained from the input L by taking the 16th bit of L as the first bit of $P(L)$, the 7th bit as the second bit of $P(L)$, and so on until the 25th bit of L is taken as the 32nd bit of $P(L)$. The permutation function P of the algorithm is repeated in the Appendix.

Now let $S_1, \ldots, S_8$ be eight distinct selection functions, let P be the permutation function and let E be the function defined above.

To define $f(R, K)$ we first define $B_1, \ldots, B_8$ to be blocks of 6 bits each for which

$$B_1 B_2 \ldots B_8 = K \oplus E(R) \tag{6}$$

The block $f(R, K)$ is then defined to be

$$P(S_1(B_1)S_2(B_2) \ldots S_8(B_8)) \tag{7}$$

Thus $K \oplus E(R)$ is first divided into the 8 blocks as indicated in (6). Then each B_i is taken as an input to S_i and the 8 blocks $S_1(B_1), S_2(B_2), \ldots, S_8(B_8)$ of 4 bits

each are consolidated into a single block of 32 bits which forms the input to P. The output (7) is then the output of the function f for the inputs R and K.

APPENDIX

PRIMITIVE FUNCTIONS FOR THE DATA ENCRYPTION ALGORITHM

The choice of the primitive functions KS, S_1, ..., S_8 and P is critical to the strength of an encipherment resulting from the algorithm. Specified below is the recommended set of functions, describing S_1, ..., S_8 and P in the same way they are described in the algorithm. For the interpretation of the tables describing these functions, see the discussion in the body of the algorithm.

The primitive functions S_1, ..., S_8, are:

S_1

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

S_2

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

S_3

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

S_4

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

S_5

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

S_6

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

S_7

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

S_8

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

The primitive function P is:

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

Recall that K_n, for $1 \leq n \leq 16$, is the block of 48 bits in (2) of the algorithm. Hence, to describe KS, it is sufficient to describe the calculation of K_n from KEY for $n = 1, 2, \ldots, 16$. That calculation is illustrated in figure 3. To complete the definition of KS it is therefore sufficient to describe the two permuted choices, as well as the schedule of left shifts. One bit in each 8-bit byte of the KEY may be utilized for error detection in key generation, distribution and storage. Bits 8, 16, ..., 64 are for use in assuring that each byte is of odd parity.

Permuted choice 1 is determined by the following table:

$PC-1$

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

The table has been divided into two parts, with the first part determining how the bits of C_0 are chosen, and the second part determining how the bits of D_0 are chosen. The bits of KEY are numbered 1 through 64. The bits of C_0 are respectively bits 57, 49, 41, ..., 44 and 36 of KEY, with the bits of D_0 being bits 63, 55, 47, ..., 12 and 4 of KEY.

With C_0 and D_0 defined, we now define how the blocks C_n and D_n are obtained

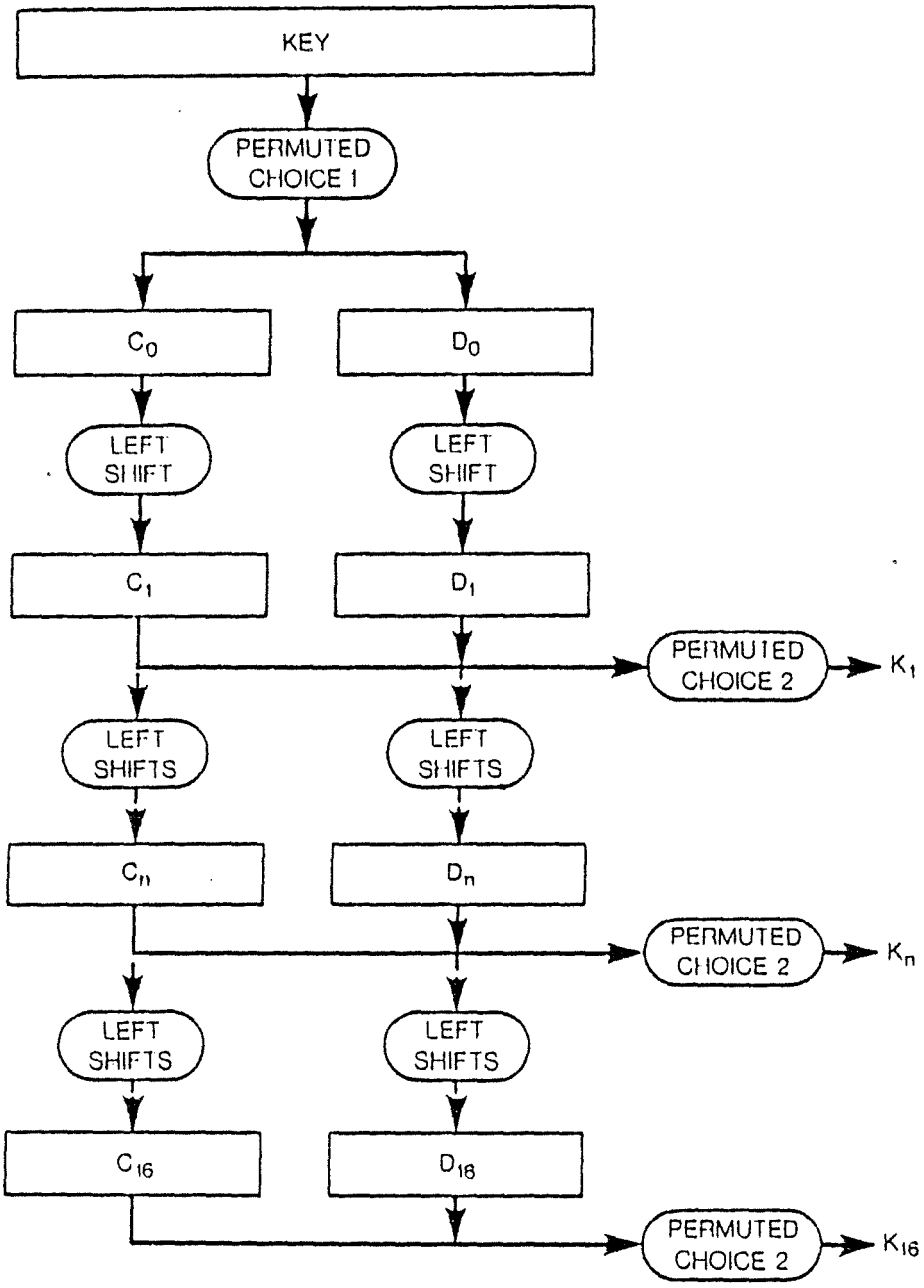


Figure 3 Key schedule calculation

from the blocks C_{n-1} and D_{n-1}, respectively, for $n = 1, 2, \ldots, 16$. That is accomplished by adhering to the following schedule of left shifts of the individual blocks:

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2

6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

For example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3, ..., 28, 1.

Permuted choice 2 is determined by the following table:

PC–2

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

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

ANNEXE 3

COMMENT CONSTRUIRE UNE GRANDE QUANTITE DE FICHIERS SIGNIFICATIFS ?

Les attaques exhaustives et "meet-in-the-middle" nécessitent la construction d'une grande quantité de fichiers significatifs. Ceci ne constitue pas un frein à la fraude. En effet, nous présenterons ci-dessous une technique de génération systématique de variations d'un même fichier.

Cette technique, de même que l'exemple qui l'illustre, sont dûs à DAVIES et PRICE <DAVIES, 84, 278-279>.

Nous expliciterons le principe de construction à partir du cas d'un fichier de texte. Toutefois, la technique est facilement généralisable à tout type de fichiers.

Pour générer une grande quantité de variations d'une même lettre, il suffit de trouver des mots ou des phrases qui peuvent être interchangés sans pour autant perdre le sens original de la lettre. Ainsi, "cher Monsieur" peut sans problème être remplacé par "Monsieur", ce qui nous donne deux lettres différentes.

Si x est le nombre d'alternatives trouvées, et si chacune de ces alternatives est indépendante des autres, il existe potentiellement 2^x lettres différentes qui peuvent servir à l'ennemi.

Chaque lettre peut être construite systématiquement par un processeur qui en calcule le CAF avant de passer à la lettre suivante. Le CAF est stocké en compagnie d'un nombre de x bits qui identifie la variante.

Ci-dessous, nous reproduisons une lettre qui permet de construire jusqu'à 2^{37} (137 milliards!) lettres différentes, chacune d'elles ayant la même signification que toutes les autres.

Dear Anthony,

This letter is / I am writing to introduce you to / to you Mr. --- Alfred P. --

Barton, the new / newly appointed chief / senior jewellery buyer for

our / the Northern European / Europe area / Division. He will take / has taken over

the --- responsibility for all / the whole of our interests in

watches and jewellery / jewellery and watches in the area / region. Please afford / give

him every / all the help he may need / needs to seek out / find the most

modern / up to date lines for the top / high end of the market. He is

empowered / authorized to receive on our behalf samples / specimens of the

latest / newest watch and jewellery / jewellery and watch products, up / subject to a

limit / maximum of ten thousand pounds sterling. He will carry / hold

a signed copy of this letter / document as proof of identity. An order

with his signature, which is appended / attached authorizes / allows you to

charge the cost to this company at the above / head office address. We

fully / ----- expect that our level / volume of orders will increase in the

following / next year and trust / hope that the new appointment will be / prove

advantageous / an advantage to both our companies.

NOTATIONS ET ABREVIATIONS

Nous trouvons ci-dessous l'ensemble des notations et abréviations introduites dans ce mémoire. Elles sont généralement classées selon la partie dans laquelle elles sont définies.

PARTIE 1

K est une clé

$E_K()$ est une fonction de chiffrement utilisant la clé K

$D_K()$ est une fonction de déchiffrement utilisant la clé K

H() est une fonction d'authentification

F est un fichier authentique

F' est un fichier frauduleux

CAF = H(F) est le code d'authentification du fichier F

CAF' = H(F') est le code d'authentification frauduleux du fichier F'

m est la taille (en bits) d'un CAF

b est la taille (en bits) d'un fichier

S(F) est la signature numérique de F

V est la fonction de vérification associée à S; elle possède entre autres la propriété de régénérer F : V(S(F)) = F

CDE : Code Détecteur d'Erreur

CDM : Code Détecteur de Manipulation

DES : Data Encryption Standard

RSA : Rivest-Shamir-Adleman

Ko : (Kilo octets) 1024 octets

Mo : (Mega octets) 1024 Ko

Go : (Giga octets) 1024 Mo

PARTIE 2

Fi est un bloc du fichier authentique F. Nous parlons aussi de bloc authentique

Fi' est un bloc du fichier frauduleux F'. Nous parlons aussi de bloc frauduleux

t est la taille (en bits) d'un bloc

n est le nombre de bloc d'un fichier

Hi, dans le processus de calcul de H(F), est le résultat intermédiaire qui suit le traitement du bloc authentique Fi

VI est la valeur initiale qui joue le rôle de H0, c'est-à-dire de 'premier résultat intermédiaire'

Hi', dans le processus de calcul de H(F') est le résultat intermédiaire qui suit le traitement du bloc frauduleux Fi'

CBC est le mode d'opération Cipher Block Chaining défini pour le DES

⊕ est le OU-exclusif bit-à-bit

※ est l'opérateur de concaténation

Ko/s : Kilo octets par seconde

Mo/s : Mega octets par seconde

Go/s : Giga octets par seconde

PARTIE 3

ms : (milliseconde) 10^{-3} secondes

*M*s : (microseconde) 10^{-6} secondes

ns : (nanoseconde) 10^{-9} secondes

PARTIE 4

MM82 est une fonction d'authentification due à MEYER et MATYAS qui est présentée au point 3.1. de la quatrième partie

MS88 est une fonction d'authentification due à MEYER et SCHILLING qui est présentée au point 3.2. de la quatrième partie

VF est la valeur finale qui est utilisée dans le QU88

QU88 est une fonction d'authentification due à QUISQUATER qui est présentée au point 3.3. de la quatrième partie

MAA est le Message Authenticator Algorithm qui est présenté au point 3.4. de la quatrième partie

PARTIE 5

CAD : Centre d'Achat et de Développement

TRM : Tamper-Resistant Module

GLOSSAIRE

AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)
Organisme non gouvernemental des Etats-Unis. Il met au point et publie des standards non obligatoires mais largement respectés.

ALGORITHME
La description précise des différentes étapes nécessaires à l'obtention des solutions d'un problème.

ANALYSE STATISTIQUE
Analyse d'une fonction d'authentification visant à découvrir d'éventuelles faiblesses qui pourraient être exploitées pour accélérer une attaque contre cette même fonction.

ATTAQUE DE L'INTERIEUR
Attaque d'un système informatique émanant d'un de ces utilisateurs légitimes, ou d'un membre du personnel de gestion de son système d'exploitation. L'attaque est différente de celle dite 'de l'extérieur' au sens où l'ennemi soit connaît la fonction d'authentification, soit peut obtenir autant d'accès à la fonction qu'il le désire.

AUDIT
Un examen des décisions prises ou des travaux réalisés. Cette opération est effectuée à titre exceptionnel et par des personnes indépendantes de celles ayant pris les décisions ou réalisé les travaux.

AUTHENTIQUE
L'état d'un fichier ou d'un CAF lorsqu'il est considéré par les responsables du système d'authentification comme dépourvu de toute erreur ou fraude.

AUTHENTIFICATION D'UN FICHIER
L'ensemble des mesures prises pour vérifier qu'un fichier n'a subi aucune modification par rapport au moment où il a été déclaré authentique

AUTORITE RESPONSABLE DU SYSTEME D'AUTHENTIFICATION
Une ou plusieurs personnes à qui l'on accorde notamment le pouvoir de décider si un fichier est authentique ou pas. Cette autorité est aussi dite 'de confiance' au sens où elle est supposée ne pas compromettre la sécurité du système d'authentification même si elle en a la possibilité.

BIT <2>
L'un des chiffres 0 et 1 lorsqu'il est employé en numérotation binaire.

BIT DE PARITE
Il s'agit, dans un octet, du bit de poids le plus fort lorsqu'il est utilisé comme un CDE. Il est alors calculé de telle sorte que la somme des huit bits de l'octet soit toujours paire.

BLOC <1>
Un tableau ou une séquence de bits ayant une longueur fixe. Il peut également être considéré comme la valeur d'un

numéral sous forme binaire.

CENTRE D'ACHAT ET DE DEVELOPPEMENT
L'entité du système informatique chargée de l'achat et du développement des fichiers, ainsi que de leur protection.

CHAINAGE DES BLOCS D'UN FICHIER
Lors du calcul du CAF d'un fichier, il s'agit de l'opération consistant à lier au traitement du bloc courant le résultat intermédiaire qui précède ce bloc. Le chaînage a pour conséquence de provoquer le principe de propagation des erreurs.

CHIFFREMENT <3>
L'action visant à transformer un texte clair en un texte chiffré de telle manière que les données originelles ne puissent être déterminées sans une certaine clé.

CHIFFREMENT ASYMETRIQUE
Technique de chiffrement où la clé permettant de recouvrer le texte clair ne sait être obtenue de l'analyse de celle utilisée pour produire le texte chiffré.

CHIFFREMENT SYMETRIQUE
Technique de chiffrement où la clé permettant de recouvrer le texte clair est facilement dérivable de celle utilisée pour produire le texte chiffré. Sauf mention contraire, le terme 'chiffrement' fait référence au chiffrement symétrique.

CIPHER BLOCK CHAINING (CBC)
L'un des modes de chaînage qui permet d'utiliser le DES à des fins d'authentification.

CLE
Un bloc contrôlant un processus de chiffrement et/ou de déchiffrement, ou encore un processus d'authentification.

CLE FAIBLE <1>
Une clé qui donne à une fonction de chiffrement ou d'authentification des propriétés spéciales qui peuvent, dans certaines circonstances, affaiblir leur sécurité.

CLE PUBLIQUE
Dans le cadre du chiffrement asymétrique, une clé utilisée pour le chiffrement mais qui est inutilisable pour le déchiffrement. Il est dès lors possible de la rendre publique sans compromettre la sécurité.

CLE SECRETE (ou PRIVEE)
Dans le cadre du chiffrement asymétrique, une clé qui permet de rendre claires des données chiffrées grâce à la clé publique correspondante.

CODE D'AUTHENTIFICATION DE FICHIER (CAF)
Le résultat d'une fonction d'authentification de fichier.

CODE DETECTEUR D'ERREUR (CDE)

Valeur accolée à un bloc de données et qui apporte la redondance nécessaire à la détection des modifications accidentelles qui peuvent altérer ces données.

CODE DETECTEUR DE MANIPULATION (CDM)

Une valeur redondante (style CDE) incluse à la fin d'un fichier au moment de l'authentification. Son rôle est d'augmenter la capacité d'une fonction d'authentification à détecter toute modification frauduleuse du fichier.

CONDENSAT

Le résultat d'une fonction de condensation. Dans le cadre de l'authentification de fichier, le condensat prend le nom de CAF.

CONDITIONS PRATIQUES

Conditions d'évaluation de la sécurité d'un système lorsqu'il est fait l'hypothèse que l'ennemi possède des moyens techniques et financiers en quantité réaliste.

CONFIDENTIALITE

Propriété d'une information exprimant que celle-ci doit rester uniquement connue de ses détenteurs légitimes.

CRYPTANALYSE

L'étude et le développement de méthodes permettant de briser un système de sécurité.

CRYPTOLOGIE <1>

La science qui inclut tous les aspects du chiffrement et de la cryptanalyse.

DATA ENCRYPTION STANDARD (DES)

L'algorithme de chiffrement symétrique qui offre à l'heure actuelle le meilleur produit sécurité * vitesse.

DECHIFFREMENT <3>

L'action de transformer un texte chiffré en un texte clair correspondant au texte original avant chiffrement. Seule une certaine clé permet cette action.

ENNEMI

Nous avons pris la convention d'appeler 'ennemi' tout adversaire du système d'authentification. Nous employons aussi en tant que synonymes les termes 'fraudeur' et 'attaquant'.

ENVIRONNEMENT SUR

Milieu qui est à l'abri de toute action frauduleuse non détectée.

FAIBLESSE D'UNE FONCTION D'AUTHENTIFICATION

Caractéristique d'une fonction d'authentification qui permet à un ennemi d'éviter de devoir la considérer comme une fonction générant des nombres pseudo-aléatoires.

FEDERAL INFORMATION PROCESSING STANDARD (FIPS) <1>
C'est un standard émis par le NBS et qui, bien que plus largement reconnu, est destiné en premier lieu aux agences et départements fédéraux américains.

FICHIER
Un ensemble d'informations logiques étant perçu comme une unité par un processeur.

FICHIER D'ACCOMPAGNEMENT
Fichier contenant toutes les informations relatives à la protection de l'intégrité d'un fichier, et notamment son CAF signé et l'identifiant du signataire.

FICHIER DE DONNEES
Fichier contenant des informations logiques utilisables par un programme pour la réalisation d'une ou plusieurs applications informatiques.

FICHIER SUSPECT
Fichier qui, après avoir transité par un environnement sûr, nécessite une vérification d'intégrité.

FONCTION A SENS UNIQUE <1>
Une fonction y = f(x) qui est relativement facile à calculer mais qui est très difficile à inverser. En d'autres termes, étant donné un x, il est facile de trouver y; mais connaissant un y, il est très difficile (à l'exception de quelques cas) de trouver au moins un x tel que y = f(x). L'exception provient du fait que par chance, la paire x, y peut déjà être connue.

FONCTION "COLLISION FREE"<6>
Une fonction de condensation pour laquelle il est impossible, dans des conditions pratiques, de trouver une collision, c'est-à-dire deux fichiers possédant le même condensat.

FONCTION D'AUTHENTIFICATION DE FICHIER
Dans le cadre de l'authentification des fichiers, une fonction de condensation à sens unique

FONCTION DE CONDENSATION <1>
Une fonction totalement définie qui, appliquée à un fichier de taille fixe b, génère un nombre apparamment aléatoire de taille m, avec m significativement plus petite que b.

FONCTION DE SIGNATURE
Une fonction permettant de générer une signature numérique à partir d'un fichier.

FONCTION DE VERIFICATION
Une fonction permettant de vérifier qu'une signature numérique est correcte pour un fichier donné.

FRAUDULEUX
Qui émane d'un ennemi, qui intervient dans la conception d'une fraude.

INTEGRITE
Etat d'une chose qui est resté inchangée.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO) <1>
Une organisation internationale dans laquelle des représentants d'organismes nationaux de standardisation (tel ANSI) se rencontrent pour agréer des standards à usage planétaire.

LISTE DES FICHIERS A PROTEGER
Fichier contenant la liste des fichiers qui doivent subir une authentification avant utilisation.

LISTE DES FONCTIONS DE VERIFICATION VALABLES
Fichier contenant la liste des fonctions de verification qui sont reconnues légales par l'autorité responsable du système d'authentification.

LOGICIEL
Un ensemble de programmes et de fichiers de données organisé de manière à accomplir une fonction déterminée de traitement de l'information.

MANIPULATION CHOISIE
Altération d'un fichier entièrement choisie pour sa signification frauduleuse. Elle est donc indépendante de la fonction d'authentification utilisée pour protéger ce même fichier.

MANIPULATION IMPOSEE
Altération d'un fichier qui, au moins pour une partie, est choisie parce qu'elle permet de déjouer la fonction d'authentification qui protège ce même fichier. Elle est de ce fait souvent sans signification.

MEET-IN-THE-MIDDLE
Une attaque contre une fonction d'authentification qui permet de créer un fichier frauduleux entièrement choisi par l'ennemi. Elle ne requiert que la racine carrée des moyens nécessaires à une attaque exhaustive sur la même fonction.

MEMOIRE PRIMAIRE
Mémoire où doivent être stockées les instructions et les données pour pouvoir être directement accessibles par un processeur.

MEMOIRE SECONDAIRE
Mémoire plus capacitaire, mais d'accès plus lent que la mémoire primaire.

MESSAGE AUTHENTICATOR ALGORITHM (MAA)
Un algorithme d'authentification destiné à être utilisé dans un contexte où la clé est inconnue de l'ennemi. Il est conçu pour être efficacement implémenté sur la plupart des processeurs classiques.

MESSAGE FINANCIER
Un fichier créé à des fins de communication et contenant des

informations ayant des implications financières.

MODULO, MODULE

Une opération est dite modulo - n si son résultat est augmenté ou diminué d'un nombre entier de fois la valeur entière strictement positive n de manière à ce qu'il soit contenu entre 0 et n-1. Dans cette opération, n est appelé le module.

NATIONAL BUREAU OF STANDARDS (NBS)

Ce service du département du commerce des Etats-Unis est chargé d'améliorer l'utilisation du traitement automatique des données. Dans ce but, le NBS édicte notamment des standards appelés FIPS.

NUMEROTATION BINAIRE <2>

Numérotation employant les chiffres 0 et 1 et dont la base de numérotation est deux. A titre d'exemple, dans ce système de numérotation, le numéral 110 représente le nombre 6, c'est-à-dire

$$1 * 2^2 + 1 * 2^1 + 0 * 2^0.$$

OCTET

C'est un bloc de 8 bits qui représente une unité standard de longueur d'information.

OU-EXCLUSIF (⊕)

C'est une opération binaire qui est définie par :

$0 \oplus 0 = 0,$

$0 \oplus 1 = 1,$

$1 \oplus 0 = 1,$ et

$1 \oplus 1 = 0.$

OU-EXCLUSIF BIT-A-BIT (bitwised exclusive-OR)

C'est une opération qui travaille sur des opérandes de plusieurs bits. Son résultat s'obtient par un OU-exclusif entre les bits de même poids. A titre d'exemple :

$$\begin{array}{rr} & 10001 \\ \oplus & \underline{00111} \\ = & 10110 \end{array}$$

Cette opération possède deux propriétés intéressantes, à savoir :

- (pour tout X), $X \oplus X = 0,$
- (pour tout X), $X \oplus 0 = 0.$

PARADOXE DE L'ANNIVERSAIRE

Combien de personnes doit-il y avoir dans un groupe pour qu'il soit plus que probable qu'au moins deux d'entre elles aient la même date d'anniversaire ? La réponse, vingt trois, peut paraître paradoxale. Ce problème est à la base d'un

certain type d'attaque contre une fonction d'authentification : les attaques "meet-in-the-middle".

PERSONNE DE CONFIANCE
(voir 'autorité responsable du système d'authentification')

POIDS D'UN BIT
L'exposant qui est attribué à un bit lors de l'évaluation du numéral auquel il appartient.

PROCESSEUR <2>
Dans un ordinateur, unité fonctionnelle capable d'interpréter et d'exécuter des instructions.

PROCESSUS
L'exécution d'une suite d'actions dont la description est contenue dans un programme.

PROGRAMME <2>
Plan de travail spécifiant les actions à effectuer pour obtenir les solutions d'un problème et mis sous une forme permettant son exécution par un ordinateur.

PROPAGATION DES ERREURS
Lors du calcul du CAF d'un fichier, phénomène caractérisé par le fait que si une erreur se glisse dans un bloc de ce fichier, elle ne se répercute pas seulement sur le résultat intermédiaire suivant, mais également sur tous ceux qui lui succèdent, et ce jusqu'au dernier.

PROTECTED PROCESSOR
(voir 'secure processor')

PROTOCOLE
Receuil de règles successives à observer en vue de mettre correctement en oeuvre une technique.

PSEUDO-ALEATOIRE
Propriété d'un résultat généré par un algorithme mais qui néanmoins apparaît comme étant d'origine aléatoire.

RECHERCHE EXHAUSTIVE <1>
C'est une tentative de résolution d'un problème en essayant toutes les possibilités, hypothèse étant faite que chacune peut être rapidement testée et que la solution peut être reconnue.

REDONDANCE
Information superflue qui peut éventuellement être utilisée pour vérifier qu'un fichier a été généré par une source autorisée.

REGULARITE D'UNE FONCTION D'AUTHENTIFICATION
(voir 'Faiblesse d'une fonction d'authentification').

RSA <1>
Rivest, Shamir et Adleman ont donné leurs initiales à un

algorithme de chiffrement à clé publique connu sous le nom de chiffrement RSA. Il y correspond un protocole de signature digitale RSA.

SECURE PROCESSOR
Un TRM contenant un système informatique complet.

SIGNATURE NUMERIQUE (ou DIGITALE) D'UN FICHIER <1>
Un nombre dépendant de tous les bits d'un fichier ainsi que d'une clé secrète. Son authenticité peut être vérifiée en utilisant une clé publique.

SYSTEME D'EXPLOITATION
Logiciel destiné à servir d'interface à tous les travaux soumis à un ordinateur, et à fournir des programmes d'utilité générale tant au programmeur qu'à l'utilisateur.

SYSTEME INFORMATIQUE
Un ensemble de matériels, de programmes, de données et, le cas échéant de personnes. Il est organisé de façon à accomplir un ensemble de fonctions déterminées de traitement de l'information.

SYSTEME INFORMATIQUE REPARTI
Plusieurs systèmes informatiques interconnectés par un système de communication et coopérant à la réalisation d'une tâche commune.

TAMPER-RESISTANT MODULE (TRM) <5>
Un composant matériel contenant des possibilités de stockage et de calcul, et qui protège ceux-ci des manipulations frauduleuses ou accidentelles provenant de l'extérieur.

TERMINAL INTELLIGENT
Dans le jargon informatique, terminal possédant ses propres moyens de stockage et de calcul.

TEXTE CLAIR <3>
Le fichier avant son chiffrement et après son déchiffrement.

TEXTE CHIFFRE <3>
Le fichier après son chiffrement et avant son déchiffrement.

VALEUR (ou VARIABLE) D'INITIALISATION (VI)
C'est une valeur qui, dans certains cas, sert à initialiser le processus d'authentification.

VERSION LOGICIELLE D'UN PROGRAMME
Un programme qui est matérialisé par des données et des instructions contenues dans un fichier.

VERSION MATERIELLE D'UN PROGRAMME
Un programme qui est matérialisé par un composant physique.

VIRUS <4>
Un programme qui peut 'infecter' d'autres programmes en y incluant une version, éventuellement évoluée, de lui même. Tout programme infecté peut devenir à son tour un virus.

WORKING-BACKWARDS

Une attaque contre une fonction d'authentification qui permet de créer un fichier frauduleux entièrement choisi par l'ennemi, sauf en ce qui concerne le premier bloc qui est imposé et sans signification.

SOURCES: <1> <DAVIES, 84, 367-380>

<2> NORME FRANCAISE Z 61-000 (1980), **Traitement de l'information, vocabulaire international de l'informatique**, AFNOR, Paris

<3> <MADRYGA, 84)>

<4> <COHEN, 87>

<5> <WOOD, 86>

<6> <DAMGARD, AP>

BIBLIOGRAPHIE ET ENTRETIENS

BIBLIOGRAPHIE

A.P. (1988), " Encore des virus dans les ordinateurs aux Etats-Unis ", **Le soir**, 6 juillet, 7

ANSI Standard Committee on Financial Services (1982), " Financial institution message authentification ", **ANSI standard X9.9**

AKL, S.G. (1984), " On the security of compressed encodings ", **Advances in cryptology, proceedings of crypto 83**, Plenum Press, New York, 209-230

ANONYME (1986), " Archivage électronique ", **Séminaire CAP SOGETI INFORMATION**, Paris

BEKER, H. - PIPER, F. (1982), **Cipher systems - The protection of communications**, Northwood Books, Londres

BRICKELL, E.F. - DELAURENTIS, J.M. (1986)," An attack on a signature scheme proposed by OKAMOTO and SHIRAISHI ", **Advances in cryptology, proceedings of crypto 85**, LNCS, Springer-Verlag, 218, 28-32

CAMPANA, M. - GIRAULT, M. (1988), " Comment utiliser les fonctions de condensation dans la protection des données? ", **Actes de SECURICOM 88**, Paris, 91-110

CARLSON, R.A. - LUNT, T.F. (1986), " The trusted domain machine, a secure communication device for security guard applications ", **Proceedings of the 1986 symposium on security and privacy**, Oakland, 182-186

COHEN, F. (1987), " Computer viruses : theory and experiments ", **Computer & Security**, 6, 22-35

COPPERSMITH, D. (1986), " Another birthday attack ", **Advances in cryptology, Proceedings of crypto 85**, LNCS, Springer-Verlag, 218, 14-17

COPPERSMITH, D. (1987), " Cryptography ", **IBM Journal of Research Development**, 31, 2, 244-247

DAMGARD, I.B. (à paraître), " Collision free hash functions and public key signature schemes ", **Proceedings of Eurocrypt 87**

DAVIES, D.W. (1983), " Applying the RSA signature to electronic mail ", **Computer**, 16, 2, 55-62

DAVIES, D.W. - PRICE, W.L. (1984), **Security for computer networks**, John Wiley & Sons, Chichester

DAVIES, D.W. (1985), " A message algoritm suitable for a mainframe computer ", **Advances in cryptology, proceedings of crypto 84**, LNCS, Springer-Verlag, 392-400

DELESCAILLES, J.P. - QUISQUATER, J.J. (1988), Communication privée

DESMEDT, Y. (1988), " Major security problems with the 'unforgeable' (FEIGE-) FIAT-SHAMIR proofs of identity and how to overcome them ", **Actes de SECURICOM' 88**, 147-159

DIFFIE, W. - HELLMAN, M. (1976), " New directions in cryptography ", **IEEE transactions on information theory**, 22, 6, 644-654

DoD Computer Security Center (1983), " Department of Defense trusted computer system evaluation criteria ", **CSC- STD-001-83**

FERNANDEZ, E.B. - SUMMERS, R.C. - WOOD, C. (1981), **Database Security and Integrity**, Addison-wesley, Reading

FERREIRA, R. (non publié), " Network vulnerability and security of cryptographic algorithms ", C.T.I., Fontenay aux Roses

GILBERT, E.N. - MAC WILLIAMS, F.J. - SLOANE, N.J.A. (1974), " Codes which detect deception ", **The Bell system technical journal**, 53, 3, 405-424

GIRAULT, M. (à paraître), " Hash-functions using modulo-n operations ", **proceedings of Eurocrypt'87**

GLISS, H. (1988), " Analysis of the international hacking attack against tne NASA SPANet (Space Physics Analysis Network) ", **Actes de SECURICOM'88**, 17-23

HARROIS-MONIN, F. (1987), " Informatique : Les casses du siècle ", **L'express**, 16 octobre, 60-62

HELLMAN, M.E. (1979), " DES will totally insecure within ten years ", **IEEE Spectrum**, 16, 7, 32-39

HIGHLAND, H.J. (1985), " Microcomputer security : data protection techniques ", **Computer & Security**, 4, 123-134

HIGHLAND, H.J. (1987), " Computer viruses and sudden death! ", **Computer & security**, 6, 8-16

ISO (1985), " Report on the reviews of DIS 8731 part 1,2 and 3 ", **ISO/TC97/SC20/WG1, N30**

ISO (1986a), " Banking - Requirements for message authentifications ", **ISO/TC68/SC2/WG2, IS 8730**

ISO (1986b), " Banking - Approved algorithm for message authentification, Part 1 : DEA1 with truncation ", **ISO/TC68/SC2/WG2, DIS 8731/1**

ISO (1986c), " Banking - Approved algorithm for message authentication, Part 2 : Message Authenticator Algorithm ", **ISO/TC68/SC2/WG2, DIS 8731/2**

JUENEMAN, R.R. (1983), " Analysis of certain aspects of output feedback mode ", **Advances in cryptology, Proceedings of crypto 82**, Plenum press, New York, 99-127

JUENEMAN, R.R. - MATYAS, S.M. - MEYER, C.H. (1984), " Message authentication with manipulation detection codes ", **Proceedings of the 1983 symposium on security and privacy**, IEEE Computer Society Press, 33-54

JUENEMAN, R.R. - MATYAS, S.M. - MEYER, C.H. (1985), " Message authentication ", **IEEE communication magazine**, 23, 9, 29-40

JUENEMAN, R.R. (1987), " A high speed manipulation detection code ", **Advances in cryptology, Proceedings of crypto 86**, LNCS, Springer-Verlag, 263, 327-346

MAC WILLIAMS, F.J. - SLOANE, N.J.A. (1977), **The theory of error-correcting codes**, Elsevier, New York

MADRYGA, W.E. (1984), " A high performance encryption algorithm ", **Proceedings of IFIP/SEC'84**, Toronto, 367-380

MATYAS S.M. (1983), " Digital signatures with the data encryption standard ", **IEEE COMPCON FALL 83**, 436-441

MEINADIER, J.P. (1984), **Structure et fonctionnement des ordinateurs**, Larousse, Paris

MEYER, C.H. - MATYAS, S.M. (1982), **Cryptography - A new dimension in computer data security**, John Wiley & Sons, New York

MEYER, C.H. - SCHILLING, M. (1988), " Secure program load with manipulation detection code ", **Actes de SECURICOM 88**, Paris, 111-130

MONOD-BROCA, A. (1986), " Fraud and failures, a proposal to maintain data integrity in case of incidents", preprint of **IFIP/SEC'86**, Monte Carlo, 276-282

NBS (1977), " Data Encryption Standard ", **FIPS publication 46**

NBS (1980), " DES modes of operations ", **FIPS publication 81**

NBS (1981), " Guidelines for implementing and using the NBS Data Encryption Standard ", **FIPS publication 74**

OKAMOTO, T. - SHIRAISHI, A. (1985), " A fast signature scheme based on quadratic inequalities ", **Proceedings of the 1985 symposium on security and privacy**, Oakland, 123-132

ONG, H. - SCHNORR, C.P. - SHAMIR, A. (1984), " An efficient signature scheme based on quadratic equations ", **Proceedings of the 16TH symposium on the theory of computing**, Washington, 208-216

PAILLES, J.C. - GIRAULT,M. (1986), " The security processor CRIPT ", **preprint of IFIP/SEC 86**, Monte Carlo, 127-139

PARKER, D.B. (1986), " Consequential loss from computer crime ", **preprint of IFIP/SEC'86**, Monte Carlo, 425-429

POZZO, M.M. - GRAY,T.E. (1986), " Computer virus containment in untrusted computing environments ", **Preprint of IFIP/SEC'86**, Monte carlo, 118-126

PRICE, N.L. (1985), " Modes of operation and hash fonctions ", **ISO/TC97/SC20/WG2, N31**

RABIN, M. (1978), " Digital signatures ", **Foundations of secure computation**, Academic Press, New York

RIVEST, R.L. - SHAMIR, A. - ADLEMAN, L. (1978), " A method for obtaining digital signatures and public-key cryptosystems ", **Communication of the ACM**, 21, 2, 120-126

SIMMONS,G.J. (1988), " A natural taxonomy for digital information authentification schemes ", à paraître dans **Advances in cryptology, Proceedings of crypto 87**

SLOANE, N.J.A. (1982), " Error-correcting codes and cryptography, part II ", **Cryptologia**, 6, 3, 258-278

SUGARMAN, R. (1979), " On foiling computer crime ", **IEEE spectrum**, 16, 7, 31-32

WEGMAN, M.N. - CARTER, J.L. (1981), " New hash functions and their use in authentication and set equality ", **Journal of computer system science**, 22, 3, 265-279

WHITE, S.R. - COMERFORD L. (1987), " ABYSS: a trusted architecture for software protection ", **Proceedings of the 1987 symposium on security and privacy**, Oakland, 38-51

WINTERNITZ, R.S. (1984a), " Producing a one-way hash function from DES ", **Advances in cryptology, Proceedings of crypto 83**, plenum press, New York, 203-207

WINTERNITZ, R.S. (1984b), " A secure one-way hash function build from DES ", **IEEE compcon 84**, 88-90

WOOD, C.C. - ZEIDLER, H.M. (1986), " Security modules : potent information security system components ", **Computer & Security**, 5, 114-121

YUVAL, G. (1979), " How to swindle Rabin ", **Cryptologia**, 3,3, 187-189

ENTRETIENS

QUISQUATER JEAN-JACQUES, Philips Research Laboratory
Avenue Van Becelaere, 2 box 8
1170 Bruxelles

vu à Bruxelles les 14/9/1987 et 9/5/1988

FERREIRA RONALD, Centre de Technologies Informatiques
Avenue du Général Leclerc, 4 à 16
92260 Fontenay aux Roses (France)

vu à Fontenay aux Roses du 01/09/1987 au 31/01/1988 et le 02/08/1988