Microprocesseur sécurisé.

1 Introduction

Avec le développement de médias de forte capacité et peu chers tels que les CD−ROM et autres DVD−ROM la diffusion de logiciels et de contenus artistiques tel que les films ou la musique est entrée dans une ère de diffusion de masse : le très faible coût de reproduction du contenu numérique, qui consiste à recopier une suite de 0 et de 1, échappe aux lois économiques classiques de la production dans la mesure où le coût est concentré dans la confection de l'original et non plus dans la production des copies vendues.

L'apparition de réseaux mondiaux à fort débit de type Internet apporte un autre type de distribution de logiciels ou de contenus artistiques où chacun peut acheter une copie et la télécharger directement dans son ordinateur.

Malheureusement, le développement de versions inscriptibles de ces médias et plus généralement le faible prix du méga−octet des disques durs permettent aussi le développement d'utilisations allant bien au delà de l'usage dans un but de copie privée de sauvegarde... Cet usage illégal est aussi aidé par la possibilité qu'offre Internet de disséminer mondialement tout type d'information numérique.

Une autre problématique est celle de la sécurité informatique et la nécessité d'avoir des systèmes résistants aux attaques informatiques bien sûr mais aussi aux attaques physiques d'un système sensible, telle que la manipulation des bus systèmes d'un ordinateur, des liens de communication, etc. Bien que moins connus du grand public, de tels systèmes où les intrusions sont à éviter sont utilisés tous les jours : les banques, les distributeurs de billets automatiques, le commerce électronique ou encore les systèmes confidentiels défense, et de manière plus générale les

télécommunications.

1.1 Spécialisation du processeur

Le premier problème peut être résolu en spécialisant chaque ordinateur au point qu'un programme ne peut plus tourner que sur un seul ordinateur unique au monde. La technique utilisable est l'usage de la cryptographie à clé publique et clé secrète [8] au sein même du microprocesseur. C'est la technique utilisée dans les cartes à puces. Mais dans le cas d'un ordinateur générique le principe est plus difficile à mettre en ?uvre car il y a des éléments périphériques qui sont espionnables (contrairement à une carte à puce qui est un ordinateur complet) : les bus sont espionnables et leurs états sont modifiables, la mémoire est espionnable et modifiable, les entrées−sorties aussi, etc. Bien que compliquées, de telles attaques sont possibles et une fois le logiciel mis à nu son contenu peut être répliqué par des pirates.

La suite de cet article présentera des techniques essayant de parer ce genre d'attaque en chiffrant et signant électroniquement tous les flux d'information entrant et sortant du microprocesseur afin de résoudre le second problème.

1.2 Communications et routeurs sécurisés

Le fait de disposer d'un processus s'exécutant de manière chiffrée dépasse le seul domaine de la protection des programmes. Dans le domaine des télécommunications sécurisées l'usage de la cryptographie permet déjà d'assurer une communication secrète de bout en bout. Dans la mesure où les routeurs du réseau sont espionnables, il est néanmoins possible de savoir s'il y a ou non du trafic sur tel ou tel lien de communication et donc de savoir quelles sont les entités communiquant entre elles, à défaut de pouvoir en décrypter le contenu.

En utilisant un système de processus cryptographiques dans les routeurs on ne peut même plus accéder à l'algorithme de routage. Le fait que de l'information

transite entre tel et tel n?ud peut être dissimulé en cachant par exemple le flux dans un pseudo−bruit continu sur chaque lien. On ne pourra distinguer l'absence de la présence de transmission puisqu'on ne saura pas décrypter le code générant ce flux au niveau des routeurs.

1.3 Contenus artistiques numériques

Outre la protection de programmes ou de réseaux, il est nécessaire de pouvoir empêcher la copie illégale de contenus artistiques numériques. Le problème est que ces contenus doivent être concrétisés en sons et en images pour être perçus par leurs utilisateurs. Dès le moment où on a la transcription sous forme physique (c'est à dire analogique) du contenu numérique ce contenu peut être copié et renumérisé. Il y a certes une perte de qualité liée à cette double conversion numérique−analogique−numérique mais cette perte de qualité n'est pas forcément très sensible (D'autant plus que le grand public se familiarise avec les algorithmes de compressions avec perte : MPEG, JPEG ou pire encore avec le GSM...). Si on peut accéder directement au contenu numérique, typiquement avant les convertisseurs numérique−analogique on peut même arriver à copier les données d'origines. Néanmoins, la recompression avec les algorithmes avec perte fait qu'on ne retrouvera probablement pas le flot numérique comprimé d'origine. On peut se protéger d'une telle attaque en intégrant ces convertisseurs sur le processeur ou en utilisant un canal chiffré entre le processeur et les convertisseurs.

Une méthode plus définitive serait de rajouter une partie logicielle au contenu artistique, une petite application multimédia présentant tel groupe de musique ou telle mise en scène d'un film, ou encore de sortir simplement du cadre monodimensionnel du classique CD de musique qu'il suffit d'écouter une fois du début à la fin pour pouvoir le recopier.

Une autre approche serait de pousser l'approche haute−fidélité en rajoutant une partie logicielle gérant l'adaptation en temps réel du contenu sonore à l'acoustique de la salle à partir de retours avec des microphones, etc. Cette partie

logicielle serait chiffrée ainsi que le contenu artistique pour éviter une extraction triviale du contenu. Un pirate pourrait certes enlever cette partie logicielle, mais la copie résultante serait assez loin de l'original et facilement identifiable.

1.4 Plan

Dans la suite, nous présentons d'abord l'approche matérielle avec des exemples, avant de s'intéresser dans la section suivante au contexte logiciel d'utilisation.

2 Principes du système

2.1 Chiffrement

Il y a eu plusieurs propositions de crypto−microprocesseurs exécutant des instructions cryptées et chiffrant ses données afin de compliquer la rétroingéniérie des programmes [2, 3, 4] et il y a même plusieurs circuits commerciaux comme les Dallas Semiconductor (marque déposée) DS5000FP [5] et DS5002FP [6] qui sont des micro−contrôleurs de type 8051 en version sécurisée.

Globalement l'approche est schématisée sur la figure 1. Le principe est de chiffrer les accès à la mémoire sur le bus *d*, à savoir les données et les instructions accédées par le microprocesseur, avec une fonction de chiffrement des données C_D dépendant d'une clé secrète c_s unique au processeur. Ainsi, le seul moyen de pouvoir décrypter ce qui passe sur le bus de données chiffrées d_c ou en mémoire est de connaître la clé c_s ou d'avoir cassé la fonction C_D. En ayant une fonction cryptographique suffisamment sûre dans l'état actuel de la connaissance et une clé suffisamment longue pour épuiser toute attaque brute essayant toutes les clés possibles, le problème doit résister à un pirate [8]. Afin de compliquer la tâche d'un attaquant, la fonction C_D dépend aussi de l'adresse afin de rendre plus difficile des détections de zones mémoires uniformément remplies qui auraient permis des attaques à texte partiellement connu (typiquement les variables initialisées à 0 auraient été représentées de manière identique dans tout le programme, facilitant le

travail de l'attaquant.) et on stocke ou d'avoir cassé la fonction.

Dans les systèmes de Robert M. BEST, les périphériques d'entrée−sortie sont sur le bus mémoire et incluent le même système de chiffrement afin de pouvoir communiquer de manière secrète avec le processeur. Le système utilise une architecture de type HARVARD avec un système de chiffrement différent sur le bus de données et le bus d'instructions afin d'éviter que le programme puisse se lire lui−même et le sortir en clair sur un port d'entrée−sortie lors d'une attaque.

Dans le cas des systèmes de Dallas Semiconductor, les entrées−sorties sont sur un bus séparé non crypté qui simplifie l'utilisation du circuit pour le marché visé par ces micro−contrôleurs : terminaux de transactions financières ou de télévision payante, etc. Afin de mélanger un peu plus les cartes le bus d'adresse est lui−même mélangé via la fonction de chiffrement d'adresse C_A afin de rendre plus difficile la compréhension de l'allocation mémoire. Le processeur est doté d'un mécanisme d'autodestruction en cas d'ouverture du boîtier, d'un mécanisme de détection fine de variations dans la tension d'alimentation, d'un système pouvant empêcher la reprogrammation. En sus, dès que le processeur n'accède pas à la mémoire, un système d'adressage génère de fausses lectures à la mémoire afin de troubler toute surveillance.

La programmation d'un tel système est classique si ce n'est la phase de chargement initiale qui utilise un mode particulier du processeur permettant de chiffrer les instructions au fur et à mesure de leur stockage en mémoire programme.

Le DS5002FP est aussi doté d'un revêtement du silicium empêchant l'usage de micro−sonde (la version DS5002FPM), d'une patte d'autodestruction (en fait l'effacement de la clé) qu'on peut relier à un mécanisme externe de détection d'intrusion, d'un vrai générateur aléatoire de 64 bits pour tirer au hasard la clé qui est de 64 bits contre 40 bits pour le DS5000FP. L'intérêt est que même le programmeur du circuit n'a pas accès à la clé *cs* et ne peut pas déchiffrer son propre programme.

Le système semble robuste, mais il a pourtant été cassé dans le cas du DS5002FP [7] ! L'attaque utilise d'abord le fait que la fonction C_D est bijective : toute valeur envoyée depuis la mémoire à l'entrée de C_D^{-1} est traduite en une instruction potentielle. Le fait que le jeu d'instruction soit publié aide ensuite : on va essayer d'envoyer toutes les instructions possibles séparées par des "reset" pour repartir dans un état cohérent à chaque fois. Comme le jeu d'instruction est sur 8 bits il n'y a que 256 essais à faire. L'idée étant de générer l'instruction commandant le port d'entrée−sortie pour faire sortir de l'information, il faut aussi faire la recherche sur l'adresse immédiate du port, ce qui multiplie le nombre d'essais à faire. De proche en proche on va construire un programme qui aura pour effet de sortir le contenu de la mémoire sur le port d'entrée−sortie en clair.

Afin d'éviter cette approche, les machines de Robert M. BEST interprètent toute instruction non définie, c'est−à−dire les valeurs ne représentant aucune instruction du jeu d'instructions valides, comme une instruction d'autodestruction, ce qui doit empêcher ce type d'attaque contrairement à ce qui est prétendu dans [7] qui semble ignorer ce détail. Mais dans le cas d'un ordinateur générique où on est amené à exécuter malheureusement n'importe quel code, du simple programme buggué (on écrase un pointeur de fonction...) au virus ou cheval de Troie le plus hostile, on ne peut pas se permettre qu'une simple exécution d'instruction illégale détruise le processeur.

2.2 Signature électronique

Afin d'éviter ce type d'attaque, on peut soit chiffrer des blocs de données plus importants d'un coup, de type ligne de cache [7], soit rajouter une signature électronique [8]. Dans les deux cas le but est de devoir faire une recherche brute dans un espace beaucoup plus vaste et être concrètement impossible. Si on se contente de chiffrer des blocs de données ou d'instructions, toute injection pirate d'autres valeurs sur le bus externe seront bijectivement traduites par des valeurs internes qui perturberont l'exécution du programme. C'est donc une méthode avec signature

électronique que nous préférons. L'idée est d'associer à toute valeur *v* une valeur hachée par une fonction cryptographique *H* ayant comme propriété que H^{-1} est très difficile (en pratique impossible) à calculer avec les moyens actuels. Au lieu de stocker en mémoire, on va stocker la valeur cryptée de la concaténation de *v* et de son hachage, c'est−à−dire la donnée signée et chiffrée **.**

Lors du décryptage d'une information chiffrée d_s on extrait *(d",h")*. Si le principe ci−dessus a été utilisé pour le chiffrement, on doit retrouver le fait que *H(d")=h"*, ce qu'il suffit de calculer pour vérifier. Si la propriété n'est pas vérifiée, c'est qu'on essaye de déchiffrer des données qui n'ont pas été chiffrées de cette manière. Si la taille du résultat de *h* est de b_s bits, un pirate a potentiellement une chance sur 2^{bs} d'injecter une valeur qui passera la vérification. Si par exemple b_s = 128, cela fait une chance sur $3{,}4.10^{38}$ ce qui semble raisonnable pour éviter des attaques même intensives dans l'état actuel de la technologie.

Le problème avec ce système de signature est que si les informations sont chiffrées par blocs de b bits et que la signature a une taille de *bs* bits, une quantité de *n* bits non cryptés en interne nécessitera d'avoir au moins bits de mémoire externe sous forme cryptée. Concrètement si on crypte au niveau d'une ligne de cache de 32 octets (soit 256 bits) et qu'on a un résultat de hachage de 128 bits, l'occupation mémoire du programme augmente de 50 %, ce qui est raisonnable.

Le corollaire est qu'il va falloir trouver un moyen de ranger ces *bs* bits supplémentaires en mémoire. On peut choisir de stocker les blocs chiffrés de manière contiguë ou encore de stocker les données par blocs de *b* bits aux adresses virtuelles internes *a* et de rajouter les *bs* bits supplémentaires dans une autre zone virtuelle à une adresse $A_S + a$ par exemple.

Dans le premier cas un octet crypté à l'adresse virtuelle interne a se retrouvera en mémoire dans le bloc d'adresse virtuelle . C'est la traduction physique a_{pl} de cette adresse virtuelle linéarisée a_l qui sortira sur le bus mémoire lors des accès cryptés au lieu de l'adresse physique a_p classique. Cette conversion peut être intégrée à la MMU

du système.

Dans le second cas, l'octet se retrouvera dans deux zones et. Dans les deux cas, il faudra faire attention aux conflits d'adresse lorsqu'un programme manipulera à la fois des valeurs cryptées et des valeurs en clair dans la mémoire.

Dans la suite, on préférera la première approche car c'est la plus simple si on veut exporter une zone mémoire cryptée sur disque : il n'y a qu'une seule zone mémoire à sauvegarder. C'est celle schématisée sur la figure 2.

On raisonne en adresse mémoire virtuelle car on bénéficie ainsi du travail de gestion mémoire fourni par un système d'exploitation classique. Il gérera de la même manière un processus crypté ou non. Cela signifie aussi que si on a un processeur avec un cache en adresse physique, il faut stocker aussi l'adresse virtuelle dans le cache afin de permettre un chiffrement dépendant de l'adresse virtuelle.

2.3 Sécurisation et mise en ?uvre

Afin de mieux résister à une attaque tentant de modifier de manière interne le microprocesseur, on peut rajouter des procédures de tests en faisant suivre tout chiffreur par le déchiffreur [1] et de vérifier qu'on retrouve bien le même résultat. Une incohérence provoquera le déclenchement d'un système de destruction pure et simple du processeur.

On peut aussi étendre le système en faisant voter plusieurs systèmes de chiffrement/déchiffrement. Ainsi, si le processeur est soumis à des attaques de type *glitch*, des variations de fréquence d'horloge interdites ou de tension d'alimentation, des rayonnements ionisants, des variations de température, etc. il sera fort probable qu'il y aura des comportements différents entre les chiffreurs/déchiffreurs redondants. Dans la mesure où le système vise les processeurs modernes de plus de 10^7 ou 10^8 transistors, le coût de la redondance est négligeable.

Pour les mêmes raisons, la clé privée du processeur doit exister en plusieurs exemplaires à différents endroits du processeur, afin de compliquer des essais de

modification.

Le choix de l'algorithme à clé publique n'est pas critique d'un point de vue vitesse dans la mesure où il ne sert qu'à décoder un descripteur d'exécutable crypté pour en extraire les deux clés symétriques qui servent à chiffrer et déchiffrer respectivement le programme et ses données. Afin d'accélérer des changements de contextes entre différents processus cryptés avec des clés symétriques différentes on peut néanmoins ajouter dans le processeur un cache de descripteur de contextes matériels décryptés. Par contre l'algorithme doit évidemment être sûr car, s'il est cassé, il suffit à l'attaquant d'acheter tous les logiciels avec la clé publique de son processeur et il pourra décrypter les logiciels et donc en faire des copies qui ne seront plus cryptées.

Le choix de l'algorithme symétrique est plus problématique puisque c'est lui qui est utilisé pour chiffrer et déchiffrer les données entre les caches et la mémoire. Il faut donc qu'il soit rapide, tel que CS Cipher [9] par exemple (8 Go/s à 1 GHz), mais on peut de toute manière pipeliner et répliquer ces algorithmes pour faire du parallélisme de données jusqu'à obtenir le débit suffisant.

Une fois les bases de la sécurité établie et le schéma global défini sur la figure 2, il s'agit de ne plus laisser filtrer la moindre information d'un processus crypté vers un autre processus. Afin d'éviter qu'un bug du programme permette de lire et de décoder les instructions, les bus de données et d'instructions sont chiffrés avec 2 clés privées différentes, respectivement c_{sd} et c_{si}. Un processus ne peut pas accéder aux clés, un processus crypté est la seule entité pouvant accéder en clair à une de ses informations cryptées. En particulier même le noyau en mode superviseur ne peut accéder au contenu des registres ou du cache d'un processus crypté.

Comme le système doit pouvoir démarrer un processus en mode crypté, on rajoute une instruction RTIEC permettant de démarrer un processus à partir d'un descripteur de matériel crypté avec la clé publique du processeur. De même, lorsqu'un processus crypté est interrompu, il écrit le contexte matériel d'exécution

dans une zone spéciale sous forme chiffrée avec la clé symétrique des données afin qu'il puisse éventuellement être inspecté ou modifié par le programme lui−même avec des instructions d'accès à la mémoire.

Enfin, un processus crypté doit être capable d'écrire et de lire des données en clair. On rajoute donc au processeur des instructions STNC et LDNC permettant d'accéder à la mémoire sans passer par les systèmes de chiffrement.

Le mode d'intrusion JTAG et plus généralement tout mode de test ne doivent pas pouvoir être utilisés en mode crypté. Comme il faut bien mettre au point puis tester le processeur aussi en mode crypté, il faut un système capable de supprimer cette possibilité une fois le processeur testé juste après fabrication et que cette suppression ne soit pas contrôlée par un simple bit qu'un cutter laser intrusif pourrait par exemple rétablir.

3 Mise en ?uvre logicielle

Après avoir défini la réalisation matérielle il faut s'intéresser à l'aspect logiciel et ici plus particulièrement dans un contexte UNIX [10], système d'exploitation dont de nombreux sources sont disponibles, facilitant la mise en place d'un tel système, et qui a en plus l'avantage de fonctionner et d'être robuste.

Un système fonctionnel sera constitué du noyau UNIX permettant l'exécution coexistante de processus utilisateurs ou superviseur (« root »), chacun pouvant être crypté ou non.

La contrainte est que ni un processus superviseur, ni même le noyau ne peuvent surveiller ce qui se passe dans un processus crypté. Par contre un processus crypté avec un clé c_s peut partager de l'information directement de manière cryptée avec un autre processus crypté avec la même clé via des fichiers, des "sockets", des segments de mémoire partagée, etc. Dans le cas de segments de mémoire partagée cryptée, il faut qu'ils soient à la même adresse virtuelle dans chaque processus puisque le chiffrement dépend de l'adresse virtuelle. Avec la généralisation des

espaces d'adressages en 64 bits cette contrainte est facilement surmontable.

Cela signifie qu'un processus crypté n'est pas traçable, ni espionnable par un autre processus qui n'a pas la clé c_s car si le système d'exploitation est bien capable d'arrêter un processus crypté et de récupérer son contexte matériel, il ne sait ni l'interpréter ni le modifier sans la clé.

Comme la mise au point des programmes est difficile, un programmeur peut ponctuellement compiler un déboggueur avec la clé publique d'un client combinée à la clé symétrique (aléatoire) de son produit. Il aura ainsi un débogueur qui ne tournera que sur le poste client et qui ne sera capable que de tracer l'exécution de sa copie spécifique du logiciel.

3.1 Appels au système

Un processus chiffré doit pouvoir utiliser les services du noyau en utilisant classiquement les appels systèmes. Afin que le noyau comprenne les services demandés, le processus crypté doit déchiffrer les arguments passés avant de faire un "trap" dans le noyau.

Pour ce faire, il faut modifier la bibliothèques de d'interfaçage système UNIX pour qu'elle déchiffre les arguments dans une zone tampon intermédiaire allouée à la volée avant de faire le trap. Comme la partie de chiffrement dépend du programme considéré, elle ne peut pas faire partie d'une bibliothèque dynamique globale comme c'est la cas normalement. Si l'interface système au niveau trap ne change pas, cela permet de garantir la compatibilité ascendante et donc qu'un programme chiffré marchera encore sur des versions ultérieures du système (c'est par un mécanisme similaire qu'on arrive sur Sun à faire marcher des programmes SunOS 4.x en SunOS 5.x par exemple : en gérant une sémantique de trap SunOS 4.x au dessus de SunOS 5.x. même si la bibliothèque système est liée statiquement à la compilation).

Certains appels systèmes doivent être proposés en version chiffrée mais aussi en version non chiffrée. Par défaut, pour des raisons de sécurité, les appels systèmes

font des transferts chiffrés. Par exemple write() permettra d'écrire dans un fichier (au sens général d'UNIX, c'est−à−dire une entité représentable par un descripteur de fichier) et les données écrites le seront de manière chiffrée et donc illisibles par un attaquant. Afin de garder la sémantique UNIX des appels systèmes (si on veut écrire *n* octets physiquement en mémoire, qu'ils soient chiffrés ou non, c'est bien *n* qu'on passe en paramètre) les contraintes de format des données chiffrées devront être gérées : traitement des données par bloc et donc données de taille multiple d'une taille de bloc.

Mais il faut aussi pouvoir avoir des entrées−sorties non chiffrées pour communiquer avec l'utilisateur ou d'autres programmes. La bibliothèque système et éventuellement les autres bibliothèques proposeront des fonctions suffixées de "_nc" qui (dé)chiffreront à la volée les données transférées via des tampons allouées pour l'occasion. Les performances seront donc plus faibles que les appels systèmes communiquant directement en mode chiffré. La fonction write_nc() permettra par exemple d'écrire des données non chiffrées (en clair) dans un fichier.

On retrouvera cette distinction au niveau des bibliothèques standard. Typiquement les fonctions d'entrées−sorties tamponnées de la bibliothèque standard du C existeront en version capable d'accéder à des fichiers cryptés et en version pour des fichiers non cryptés.

Les fonctions d'allocation mémoire de type malloc(), calloc() appelées dans un processus crypté réserve en interne la zone pour permettre en plus le stockage de la signature et une taille de blocs entiers.

3.2 Création de processus cryptés

La création d'un nouveau processus se fait par clonage via un appel à une fonction de type fork() en UNIX. S'il s'agit d'un processus crypté, fork() créera un nouveau processus crypté. L'exécution d'un autre programme dans un processus UNIX est réalisée par un appel de type exec(). Si cet appel système rencontre un

fichier exécutable de type crypté il démarrera le programme en exécutant une instruction de type RTIEC sur un descripteur de contexte matériel présent dans le format exécutable après l'avoir mis en mémoire via un mmap(). De même que pour l'exécution d'un programme non crypté, le processus résultant hérite des droits, des descripteurs de fichiers, des variables d'environnement et le statut de traitement des signaux [10].

Les variables d'environnement doivent être cryptées au démarrage du programme si le processus qui lance l'exécution d'un programme crypté n'était pas un processus crypté. Cela doit être rajouté par le compilateur ou l'éditeur de lien lorsqu'est construit le programme crypté. Cela veut dire aussi qu'afin de transférer ces variables d'un processus crypté à un autre processus crypté lors d'un exec() il faut un structure de donnée plus relogeable en mémoire que l'actuel char **environ puisqu'une fois cryptée le système d'exploitation n'est plus capable de retrouver la structure des ces chaînes de caractères.

3.3 Signaux

Les signaux en UNIX sont des interruptions logicielles qui se traduisent par des appels de fonctions spécifiques d'un processus par le noyau lorsque le processus reçoit un signal [10]. Concrètement cela signifie que le noyau, qui fonctionne a priori en mode non crypté, doit appeler une fonction (a priori cryptée) du processus crypté avec en paramètre le numéro de signal, des informations supplémentaires sur le signal et le contexte d'exécution au moment où a eu lieu le signal.

Ces deux dernières informations contiennent une partie provenant du système d'exploitation qui est donc non chiffrée ainsi qu'une partie représentant le contexte matériel du processus qui est en format chiffré. Il faut donc que l'édition de lien rajoute une sur−couche chiffrant les éléments non chiffrés avant exécution de la fonction gérant le signal dans le programme crypté.

La réincarnation du processus crypté dans le contexte du signal est semblable

au démarrage d'un processus crypté via un appel système de type exec_c() et nécessite donc une instruction de type RTIEC prenant en paramètre un contexte matériel chiffré.

L'enregistrement d'une fonction auprès du système d'exploitation est fait par l'appel système sigaction_c() (au lieu du classique sigaction()) qui prendra en paramètre un descripteur de contexte matériel chiffré initialisé pour exécuter la fonction (au lieu de prendre classiquement un pointeur de fonction).

3.4 Compilation

La compilation d'un source pour produire un binaire exécutable en mode crypté est assez standard [10] mais le format des exécutables doit néanmoins être étendu pour comporter : un entête indiquant qu'on a affaire à un exécutable crypté ; un descripteur crypté avec la clé publique du processeur qui contient la clé symétrique de chiffrement de l'exécutable et le contexte matériel crypté permettant de lancer l'exécution du programme, à la place de la simple adresse de démarrage classique ; une zone "text" comportant le code exécutable crypté ; une zone "initialized data" contenant sous forme cryptée les objets initialisés du programme ; une zone "uninitialized data" (ou BSS) contenant les variables non initialisées qui seront remplies par des 0 (cryptés) lors du démarrage du programme (Ce remplissage par des 0 peut faciliter une attaque à texte connu?) ; une éventuelle table des symboles cryptée pour un déverminage par un débogueur crypté.

La phase de compilation en elle−même est inchangée : on génère du binaire non crypté. Par contre l'édition de liens est modifiée pour générer dans la phase finale des instructions et des données cryptées.

La table des symboles est étendue avec des indications comme quoi telle fonction doit avoir ses paramètres d'entrée ou de sortie cryptés afin de pouvoir être appelée par une fonction extérieure au programme non crypté (typiquement une fonction du système d'exploitation voulant exécuter une fonction associée à un

signal). À partir de cette description « à la IDL », l'éditeur de lien peut emballer la fonction avec une fonction qui chiffrera avant et déchiffrera après l'appel à la fonction du programme.

Si un programme s'intéresse finement à son allocation mémoire, typiquement pour externaliser ses données internes en mode crypté, il faudra qu'il tienne compte du fait que les données en mémoire virtuelle externe au cache prennent plus de place à cause de la signature des blocs.

Il faudra fournir au programmeur des fonctions permettant d'avoir simplement cette information.

4 Autres travaux du domaine et travaux futurs

Les recherches sur le matériel dans le domaine se sont tournés vers des petits systèmes [2, 3, 4, 5, 6] plutôt que de viser un processeur généraliste et un système d'exploitation standard devant travailler avec différents niveaux de sécurité. Il n'y a pas eu à notre connaissance d'étude dans le domaine des systèmes d'exploitation pour processeur incluant un exécution cryptée.

Il existe d'autres approches telle que la location de procédures par exemple mais cela nécessite d'interroger un serveur distant régulièrement et cette technique ne résiste pas au traçage du programme.

Dans la continuation de cet article il faudra s'attacher à définir plus finement l'architecture du processeur en vue d'une réalisation, de développer un modèle de niveaux de sécurité et de compléter le système avec une vérification globale de cohérence pour éviter les attaques par rejouage de blocs déjà chiffrés.

5 Conclusion

Les techniques présentées doivent permettre de rendre quasi−impossible le piratage logiciel par des moyens standard. Il est clair qu'une puissance étatique peut mettre des ressources dépassant de loin les moyens du pirate oeuvrant dans son garage [1] mais on imagine mal quel serait l'intérêt pour un état de dépenser des

fortunes à décrypter un logiciel qui peut être acheté à l'épicerie du coin de la rue ou encore réécrit à base de composants préexistants ou même complètement.

De même, on peut considérer que le domaine du logiciel est trop morcelé, sauf peut−être pour des systèmes d'exploitations et des suites bureautiques bien connues, pour qu'une organisation secrète privée d'une telle ampleur puisse se mettre en place pour décoder tel ou tel logiciel.

Le contenu numérique artistique même chiffré pose un problème dans la mesure où il est toujours copiable au moment où il est rematérialisé sous forme d'images ou de son. On peut donc en faire des copies, mêmes si elle ne sont pas faites en numérique et sont donc des copies de moindre qualité. Le développement de contenus plus interactifs devrait freiner cette facilité de copie.

Bibliographie

1. Ross J. ANDERSON et Markus KUHN. « Low Cost Attacks on Tamper Resistant Devices ». Dans Bruce CHRISTIANSON, éditeur, Security protocols: 5th international workshop, Paris, France, April 7−9, 1997: proceedings, volume 1361 de Lecture Notes in Computer Science, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1998. Springer−Verlag.

2. Robert M. BEST. « Preventing Software Piracy with Crypto−Microprocessors ». Dans Proc. IEEE Spring COMPCON'80, pages 466−469, février 1980.

3. Robert M. BEST. « Crypto Microprocessor for Executing Enciphered Programs ». Rapport Technique US4278837, United States Patent, juillet 1981. Consulté le 21 février 2000 sur http://patent.womplex.ibm.com/details?&pn=US04278837__.

4. Robert M. BEST. « Crypto Microprocessor that Executes Enciphered Programs ». Rapport Technique US4465901, United States Patent, août 1984. Consulté le 21 février 2000 sur http://patent.womplex.ibm.com/details?&pn=US04465901__.

5. Dallas Semiconductor. « DS5000FP Soft Microprocessor Chip », novembre 1999. Récupérable par WWW à http://www.dalsemi.com/DocControl/PDFs/5000fp.pdf.

6. Dallas Semiconductor. « DS5002FP Secure Microprocessor Chip », mai 1999. Récupérable par WWW à http://www.dalsemi.com/DocControl/PDFs/5002fp.pdf.

7. Markus G. KUHN. « Cipher Instruction Search Attack on the Bus−Encryption Security Microcontroller DS5002FP ». IEEE Transactions on Computers,

47(10):1153−1157, octobre 1998.

8. Alfred J. MENEZES, Paul C. VAN OORSCHOT, et Scott A. VANSTONE. Handbook of applied cryptography. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431−9868, USA, 1997.

9. Jacques STERN et Serge VAUDENAY. « CS−Cipher ». Dans Serge VAUDENAY, éditeur, Fast Software Encryption: 5th International Workshop, volume 1372 de Lecture Notes in Computer Science, pages 189−205, Paris, France, 23−25 mars 1998. Springer−Verlag.

10. Uresh VAHALIA. Unix Internals : the New Frontiers. Prentice−Hall, 1996.

REVENDICATIONS

1. Microprocesseur sécurisé, comprenant en interne des moyens de sécurisation de données échangées avec des moyens de mémorisation,

caractérisé en ce que lesdits moyens de sécurisation comprennent d'une part des moyens de cryptage et d'autre part des moyens de calcul d'une signature électronique,

de façon qu'au moins certaines desdites données échangées soient soumises à un cryptage puis à un ajout d'une signature.

2. Microprocesseur sécurisé selon la revendication 1, caractérisé en ce que lesdits moyens de sécurisation associent à une valeur v une valeur hachée par une fonction cryptographique H, ou signature, et en ce que la donnée échangée est la valeur cryptée, selon une fonction de cryptage prédéterminée, de la concaténation de ladite valeur v et de ladite signature.

3. Microprocesseur sécurisé selon la revendication 2, caractérisé en ce que les bits correspondant au cryptage de ladite valeur v et les bits correspondant au cryptage de ladite signature sont stockés dans des zones distinctes desdits moyens de mémorisation.

4. Microprocesseur sécurisé selon l'une quelconque des revendications 1à 3, caractérisé en ce qu'il comprend des moyens de vérification du chiffrement, assurant une comparaison entre une donnée à chiffrer et une donnée déchiffrée délivrée par des moyens de test assurant le déchiffrement de la donnée chiffrée.

5. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 4, caractérisé en ce qu'il comprend au moins un des moyens de sécurisation appartenant au groupe comprenant :

- des moyens de détection d'une attaque de type "glitch" ;
- des moyens de détection de variations de fréquence d'horloge et/ou de tension d'alimentation ;

- des moyens de détection de rayonnements ionisants ;
- des moyens de détection de variations de température.

6. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 5, caractérisé en ce qu'il comprend des moyens d'auto−destruction.

7. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 6, caractérisé en ce qu'il comprend au moins deux emplacement de stockage d'au moins une clé privée.

8. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 7, caractérisé en ce qu'il gère au moins deux clés privées, associées respectivement aux informations circulant sur le bus de données et sur le bus d'instructions.

9. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 8, caractérisé en ce qu'il dispose d'instructions (par exemple STNC et LDNC) permettant d'accéder auxdits moyens de mémorisation sans passer par lesdits moyens de sécurisation.

10. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 9, caractérisé en ce qu'il comprend des moyens de suppression de tout mode de test.

11. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 10, caractérisé en ce qu'il est mis en ?uvre dans un contexte UNIX.

12. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 11, caractérisé en ce qu'il met en ?uvre des commandes spécifiques (par exemple write_nc) permettant des échanges de données non cryptées, les échanges de données par défaut étant effectués sous une forme cryptée.

13. Microprocesseur sécurisé selon l'une quelconque des revendications 1 à 12, caractérisé en ce qu'il comprend des moyens de compilation d'un code source, produisant un code exécutable présentant :

- un entête précisant qu'il s'agit d'un exécutable crypté ;
- un descripteur crypté avec une clé publique dudit microprocesseur et le

contexte matériel crypté permettant de lancer l'exécution dudit exécutable ;

- une zone (text) comprenant le code exécutable crypté ;
- une zone comprenant sous forme cryptée les objets initialisés du programme ;
- une zone contenant les variables non initialisées ;
- éventuellement une table des symboles cryptée.

ABRÉGÉ

Microprocesseur sécurisé.

L’invention concerne un microprocesseur sécurisé, comprenant en interne des moyens de sécurisation de données échangées avec des moyens de mémorisation, lesdits moyens de sécurisation comprenant d’une part des moyens de cryptage et d’autre part des moyens de calcul d’une signature électronique, de façon qu’au moins certaines desdites données échangées soient soumises à un cryptage puis à un ajout d’une signature.

Fig. 1