CryptoPage-1 : vers la fin du piratage informatique ?

Version courte

Ronan Keryell
LIT-ENSTBr* BP832
F-29285 BREST CEDEX

15 mars 2001

Résumé

Afin d'aider à résoudre les problèmes de piratage informatique et les besoins de confidentialité dans les applications et les communications modernes, cet article présente une technique matérielle permettant de rajouter du chiffrement fort à un processeur. Ce mécanisme associé aux caches et à la gestion mémoire permet de chiffrer les instructions du programme ainsi que les données en mémoire pour assurer une bonne résistance aux attaques sur le bus du microprocesseur. Cela permet de concevoir par exemple des programmes capables de ne tourner *que* sur une seule machine au monde, assurant ainsi la confidentialité de l'application et des communications.

1 Introduction

Avec le développement de médias de forte capacité et peu chers tel 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.

Cet article peut être obtenu à
`http://www.cri.ensmp.fr/~keryell/articles_et_programmes.html#ENSTBr_INFO_2001-001` et une version plus complète à
`http://www.cri.ensmp.fr/~keryell/articles_et_programmes.html#ENSTBr_INFO_2000-001`.

*Laboratoire d'Informatique des Télécommunications de l'École Nationale Supérieure des Télécommunications de Bretagne. `Ronan.Keryell@enst-bretagne.fr`.
Rapport ENSTBr/INFO/RR/2001-001.

2 Principes du système

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 œuvre 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 leur 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.

2.1 Chiffrement

Globalement l'approche est schématisée sur la figure 1. Le principe est de chiffrer les accès à la mémoire sur le bus D en insérant des opérateurs de chiffrement $C_D(c_s,a,d)$ qui dépendent des adresses utilisées afin de compliquer le travail d'un attaquant. Afin d'éviter des attaques où on arriverait à faire décrypter par le programme son propre code et le sortir sur une entrée-sortie, on utilise 2 clés différentes pour les instructions et les données.

Classiquement, pour des raisons de performance, on utilise un algorithme symétrique pour ce chiffrement et les clés sont obtenues par déchiffrement avec la clé privée d'un descripteur chiffré du logiciel.

Pour produire un logiciel pour cette machine, il faut donc chiffrer données et code avec 2 clés aléatoires et accompagner le logiciel avec le descripteur contenant les 2 clés aléatoires, chiffré avec la clé publique du processeur.

Avec une telle approche on est quasiment sûr que le code ne peut s'exécuter que sur le processeur cible. Néanmoins on peut injecter des instructions ou données aléatoires sur le bus externe afin d'extraire de l'information sur le déroulement du programme [7].

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 $C_D(c_s,a,d)$, on va stocker la valeur cryptée de la concaténation de v et de son hachage, c'est à dire la donnée signée et chiffrée $d_s = C_D\Big(c_s,a,\big(d,H(d)\big)\Big)$.

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^{b_s} 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 b_s bits, une quantité de n bits non cryptés en interne nécessitera d'avoir au moins $\lceil \frac{n}{b} \rceil (b + b_s)$ 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 b_s 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 b_s 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 $[\lfloor \frac{a}{b} \rfloor \frac{b+b_s}{b}, \lfloor \frac{a}{b} + 1 \rfloor \frac{b+b_s}{b} - 1]$. 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 $[\lfloor \frac{a}{b} \rfloor b, \lfloor \frac{a}{b} \rfloor b + b - 1]$ et $[A_S + \lfloor \frac{a}{b} \rfloor b_s, A_S + \lfloor \frac{a}{b} \rfloor b_s + b_s - 1]$. 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.

2.3 Sécurisation et mise en œuvre

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.

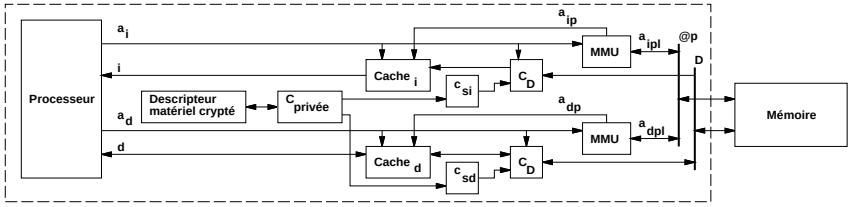

FIG. 1 – *Principe d'un processeur* CRYPTOPAGE*-1*.

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 même raisons, la clé privée du processeur doit exister en plusieurs exemplaires à différents endroits du processeur, afin de compliquer des essais de modification.

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 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.

4 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 œuvrant 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.

Un problème sociologique à ce genre d'approche pour les applications de type « grand public » est néanmoins l'acceptation par les utilisateurs d'avoir un programme qui tourne sur leur ordinateur sans pouvoir savoir ce qu'il fabrique réellement car il est clair qu'un programme peut avoir un comportement caché échappant totalement à son utilisateur. Mais tout utilisateur d'un système d'exploitation propriétaire bien connu dont les sources ne sont pas visibles par n'importe qui et dont les licences interdisent toute rétroingéniérie utilise déjà une telle boîte de PANDORE sans en avoir pleinement conscience[1]. La présence d'un identificateur matériel unique dans les processeurs Pentium III d'Intel a toutefois suscité déjà des inquiétudes...

Références

[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.

1. Il s'agit bien sûr d'un doux euphémisme... De manière générale, même si on possède les sources de tous les logiciels, cela fait un tel empilement de couches que l'utilisateur λ n'a aucun moyen de vérifier l'état de son système.