
CryptoPage-2 : un processeur sécurisé contre le rejeu

Ronan KERYELL et Cédric LAURADOUX

—

Laboratoire Informatique & Télécommunications

Département Informatique

École Nationale Supérieure des Télécommunications de Bretagne

16 octobre 2003

0-1

Besoins élémentaires :

- Grilles de calcul sécurisées en milieu naturel
(\equiv hostile)
 - ▶ Qu'est-ce qui prouve que l'ordinateur distant est sûr ?
 - ▶ Administrateur ou pirate distant peut espionner les calculs
 - ▶ Administrateur ou pirate distant peut modifier les calculs
- Applications d'authentification ou de chiffrement style carte à puce



- Routeurs sécurisés
- Programmes et systèmes d'exploitation sécurisés ou secrets
- Installation automatique sécurisée d'ordinateurs en réseau (DHCP, IPsec,...)
- Exécution de code réservée à un processeur
- Protection de contenu multimédia contre le piratage
- Code mobile chiffré et sécurisé (crypto-mobilet)

Impossible à faire avec des ordinateurs classiques ☹



Résister à toute attaque physique ou logicielle

- Attaques sur le processeur (supposé intègre et non discuté ici)
- Attaques sur les bus
- Modification des valeurs en mémoire
 - ▶ Programmes
 - ▶ Données



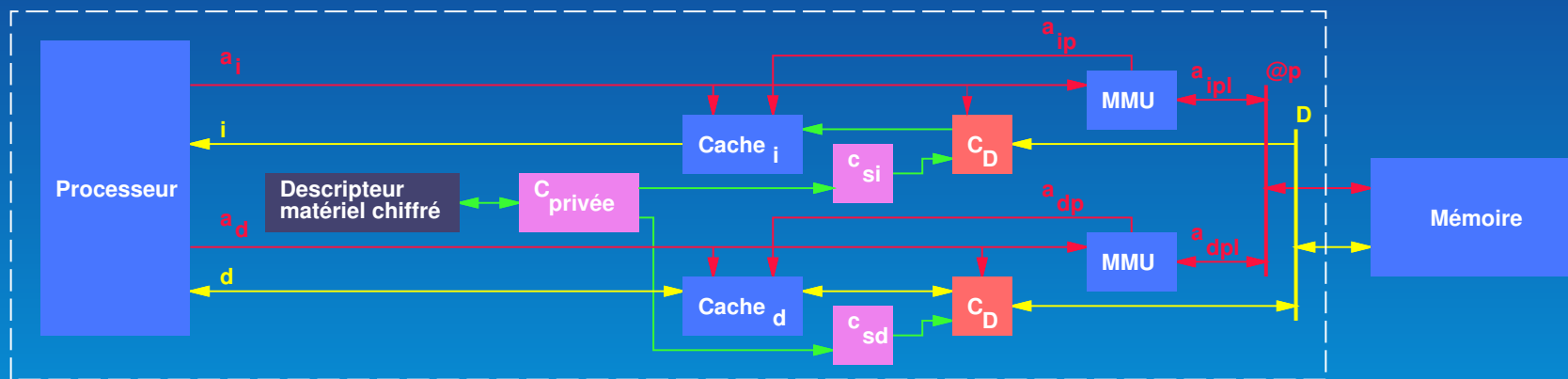
- Spécialisation du processeur : devient unique au monde
 - ▶ Utilisation de cryptographie à clé publique/clé secrète
 - ▶ Producteur du logiciel chiffre avec la clé publique du processeur
 - ▶ Processeur exécute le programme en déchiffrant avec sa clé secrète unique
 - ▶ Impossible de déchiffrer sans la clé secrète
 - ▶ Pour des raisons de performance utilisation d'un algorithme mixte avec chiffrement symétrique




avec clé de session

- Signature des instructions et données : pas possible d'injecter de fausses valeurs
- Chiffrement dépendant de l'adresse pour résister aux attaques par substitution
- Pour des raisons de performance garder usage du cache





- Pas possible de modifier la mémoire ou de changer des valeurs de place
-  Possibilité de **rejouer** une vieille ligne de mémoire : correctement chiffrée et **signée** par le processeur... ☹
- Exemple d'exploitation style `printf` :

```
for(i = 0; i < TAILLE; i++)  
    affiche(*p++);
```

 - ▶ Si variable `i` stockée en mémoire
 - ▶ Pirate envoie des interruptions au processeur pour faire vider son cache



- ▶ Possible de renvoyer au processeur ancienne ligne de mémoire avec ancienne valeur de *i* (simplement en bloquant le fil d'écriture)
- ▶ *i* n'avance plus dans le programme
- ▶ Débordement de la boucle et sortie de données ou instructions confidentielles ☹

```
for( i = 0; i < TAILLE; i++)  
    affiche(*p++);
```



- Attaque par rejeu possible car vérification locale et non globale de la mémoire (pour des raisons de performance... ☹)
 - ~ Vérificateur global de la mémoire et efficace \exists ?
- Vérificateur hors-ligne : vérifier régulièrement que la mémoire est correcte
Rapide mais piratage possible entre les passages...
- Vérificateur en ligne : vérifier à chaque accès mémoire toute la mémoire
Sûr mais lent

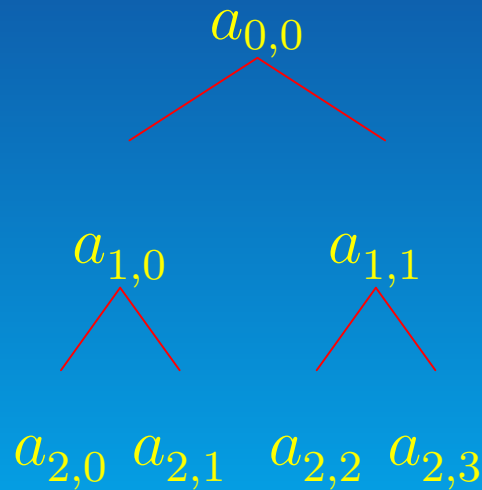


- Utiliser les mamelles accélératrices de l'informatique : **hiérarchie** et **cache**



- Fonction de hachage arborescente

Hiérarchie



- $a_{i,j} = H(a_{i+1,2j}, a_{i+1,2j+1})$

- Bas de l'arbre = mémoire à vérifier
- À chaque lecture, calculer $a_{0,0}$ et comparer à une valeur stockée de manière sûre dans le processeur
- Idée : exploiter hiérarchie pour faire du caching



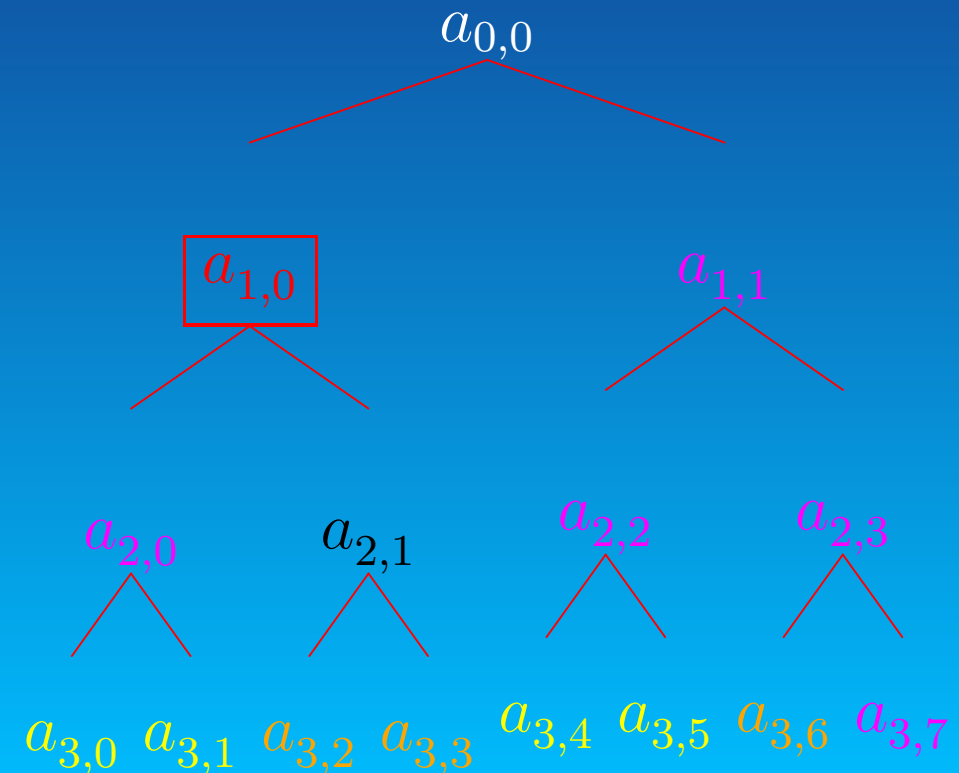
Cache

En cache

Lecture depuis la mémoire

Conflit détecté

Calcule la fonction de hachage



- Injection de fausses données ($a'_{3,3}$) impossible à cause du hachage cryptographique
→ résistance à la 2^{ème} préimage : difficile de trouver un $a'_{3,2}$ tel que

$$H(a'_{3,2}, a'_{3,3}) = a_{2,1} = H(a_{3,2}, a_{3,3})$$

- Rejeu impossible car test de mémoire global : idem injection de fausses données



- Flux d'adresses : information sur les algorithmes, sur les données (si flot de contrôle dépendant des données)
- Chiffrer les adresses !
- Compatible avec les systèmes d'exploitations classiques avec adresse

$$A = p||l||d$$

- Compromis à trouver entre obscurantisme, localité et compatibilité



~> Chiffrer séparément

▶ p : garde la notion de page

▶ l : obscurantisme intra page

▶ d : naturellement fait par le chiffrement des données

- Avoir en plus 2 zones mémoires : 1 chiffrée et 1 en clair pour un accès complet aux systèmes d'exploitation classiques



- Difficile sans réalisation ou modélisation fine...
- Commencée avec SimpleScalar
- Système entre cache(s) interne(s) et mémoire ou cache(s) externe(s)
 - ▶ Utilise pleinement le(s) cache(s) interne(s)
 - ▶ Ralentissement que sur les accès externes
- Différents points de ralentissement
- Différents algorithmes de chiffrement possible



- Chiffrement asymétrique à clé publique : très lent
- Utilisé que lors des changements de processus
- Cache interne de descripteurs de processus style TLB des MMU mais EPDB (*Enciphered Process Descriptor Buffer*)
- File d'attente de sortie des descripteurs et chiffrement en parallèle avec exécution d'un autre processus
- Utiliser un système d'exploitation prenant en compte CryptoPage : reprendre les concepts classiques de



l'ordonnanceur à 2 niveaux pour tenir compte de la mémoire d'échange secondaire

- Prise en compte de l'EPDB par l'ordonnanceur du SE
 - ▶ Préchargement d'un EPD
 - ▶ Exécution avec un EPD dans l'EPDB
 - ▶ Sortie d'un EPD en fonction de l'ordonnancement et en parallèle à une exécution,...
- Approche RISC de la sécurité : ajouter des instructions simples pour manipuler les EPD dans l'EPDB,...



- Utilisation algorithme symétrique avec clef de session dans EPD
- Algorithmes assez rapides
- AES pipelinable en 16 cycles
- Accélération des accès mémoire à la IBM360/91
 - ▶ Garder une file des accès mémoire en cours de chiffrement
 - ▶ Profondeur en fonction du chemin d'écriture en mémoire
 - ▶ Piocher dedans si on a besoin d'y réaccéder



- Calcul des condensés assez lents
 - ▶ SHA-1 génère 160 bits en 80 cycles
 - ▶ À réaliser pour faire plus rapide que le cycle du processeur
- Exécution spéculative optimiste : cas courant on n'est pas piraté...
- Astuce d'AEGIS : utiliser une fonction de hachage simple jointe au bloc qui sera chiffré de toute manière ! Plus rapide mais résistance à étudier



- Plus prospectif :
 - ▶ Réfléchir à la parallélisation du système
 - ↪ Problème de mises à jour concurrentes de l'arbre
 - ▶ Faire du SMT entre différents cryptoprocessus
 - ↪ Typage des lignes de cache par cryptoprocessus



- Beaucoup de projets ne résistant pas à une attaque par rejeu
- Grand public : TCPA/Palladium/... contourne le problème en supposant que tout loge dans une mémoire sécurisée interne au processeur
- Projet de vérificateur hors-ligne au MIT
- Processeur AEGIS au MIT : approche semblable à CryptoPage-2
 - ▶ Ont fait des modélisations ! -25 % en performance. Raisonnable



- ▶ Vérification en écriture lors de la sortie du cache
- ▶ Implantation de l'arbre de MERKLE non détaillée
- ▶ Organisation du cache dans CryptoPage-2 ne nécessitant pas de cryptographie incrémentale
- ▶ Sortie de cache optimisée dans CryptoPage-2 car cache de l'arbre
- ▶ Pas de chiffrement des adresses
- ▶ Pas de prise en compte des problèmes d'OS



- Candidature à un projet incitatif GET 2004
- Réaliser une carte à puce libre mais sécurisée
 - ▶ Libre : éviter la sécurité par obscurantisme
 - ▶ ENST Paris
 - Conception des opérateurs cryptographiques de base
 - Résistance aux analyses de consommation électrique statistique
 - Logique asynchrone
 - ▶ ENST Bretagne
 - Sécurisation de la mémoire



- Isolation sécurisée des divers processus (carte multi-applications)
- Intégration dans un système d'exploitation libre (JayaCard, *BSD, Linux,... ?)



- Domaine d'avenir pour des processeurs enfin sécurisés
- Domaine transversal vaste pour garantir sécurité sans sacrifier les performances
- Modélisation fine et réalisation concrète à faire
- Adapter un vrai système d'exploitation
- Réfléchir aux bonnes et mauvaises utilisations d'un tel système, implications sociales
- Avance difficile sans bourse de thèse... ☹



Table des transparents

Introduction

- 1 Besoins de sécurité
- 3 | Résister !
- 4 Concepts de base

CryptoPage-1

- 6 CryptoPage-1
- 7 ¿ Attaques par rejeu ?
- 9 Vérificateur de mémoire

Vérificateur

- 11 Arbres de MERKLE

- 12 Arbre de Merkle en cache
- 13 ¿ Pourquoi ça marche ?
- 14 Chiffrement des adresses
- 16 Idées de performance

Performances

- 17 D(é)Chiffrement des descripteurs de processus
- 19 Chiffrement des lignes de cache
- 20 Vérification de la mémoire
- 22 Autres projets en rapport

Conclusion

- 24 OpenSmartCard
- 26 Conclusion
- 27 Table of contents

