
CryptoPage-2 : un processeur sécurisé contre le rejeu

Guillaume DUC et Ronan KERYELL

—

Laboratoire Informatique & Télécommunications

Département Informatique

École Nationale Supérieure des Télécommunications de Bretagne

8 septembre 2004

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 ☹



Détecter voire résister à toute attaque physique ou logicielle

- Attaques sur le processeur (supposé intègre physiquement et non discuté ici. Cf. projets GET,...)
 - ▶ Intrusions
 - ▶ Canaux cachés
- Attaques sur les bus externes
- Modification des valeurs en mémoire
 - ▶ Programmes
 - ▶ Données

Toute modification extérieure de l'état du système



implique un arrêt voire une destruction



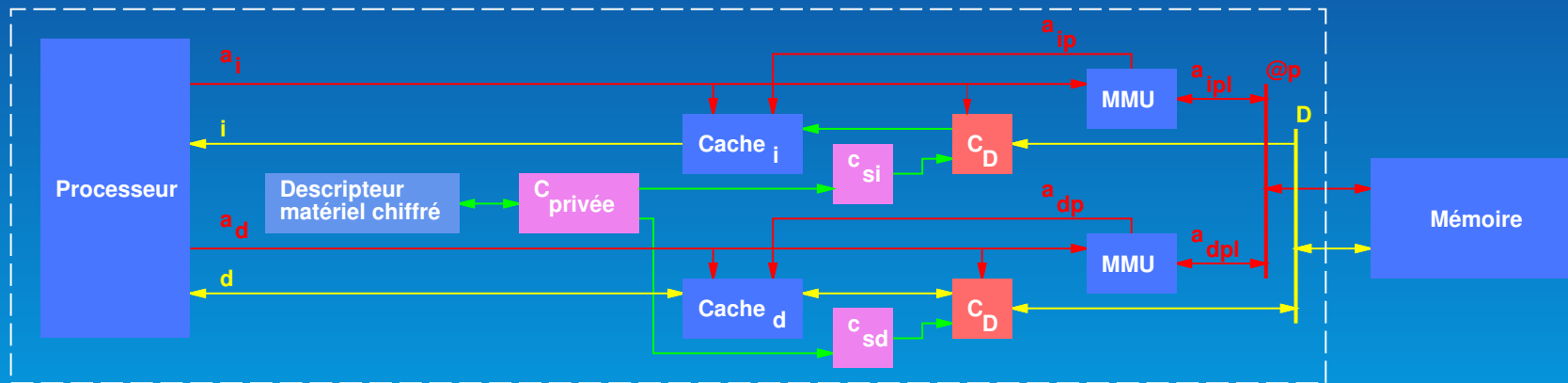
- Spécialisation du processeur : devient unique au monde
 - ▶ Utilisation de cryptographie à clé publique/clé secrète
 - ▶ Chiffrement des instructions et des données
 - ▶ Producteur du logiciel chiffre avec la clé publique du processeur
 - ▶ Processeur exécute le programme en déchiffrant avec sa clé 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

- Vérification des instructions et données avec une CRC : pas possible d'injecter de fausses valeurs
- Chiffrement dépendant de l'adresse pour résister aux attaques par substitution
- Rajout d'un vecteur d'initialisation style CBC pour éviter des attaques à texte connu
- 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 ou alors cacher les valeurs entre vérifications
- Vérificateur en ligne : vérifier à chaque accès mémoire toute la mémoire
Sûr mais lent

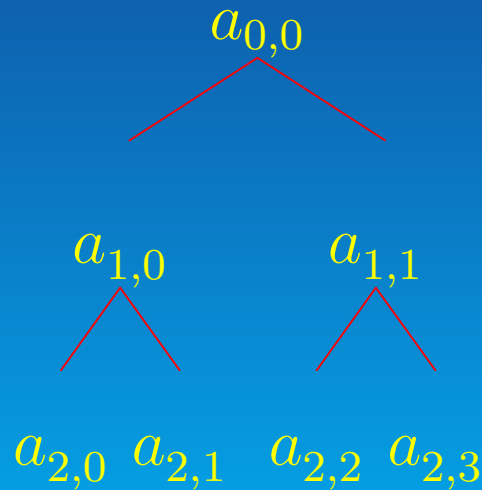


- Utiliser les mamelles accélératrices de l'informatique : **hiérarchie**, **cache** et **parallélisme**



- 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



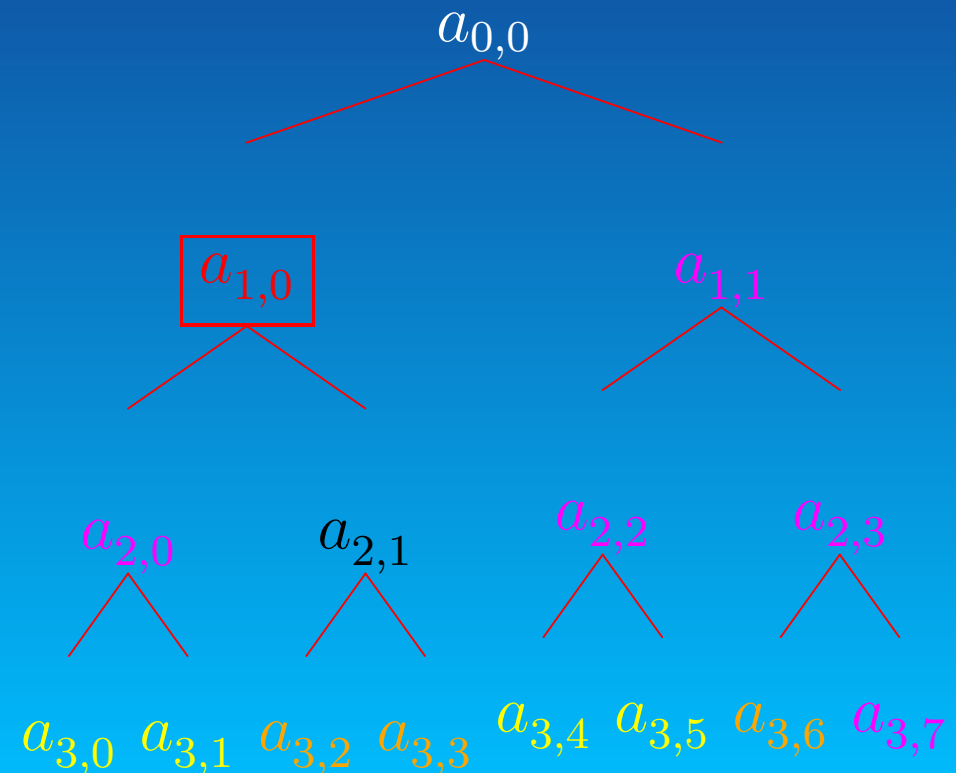
Cache

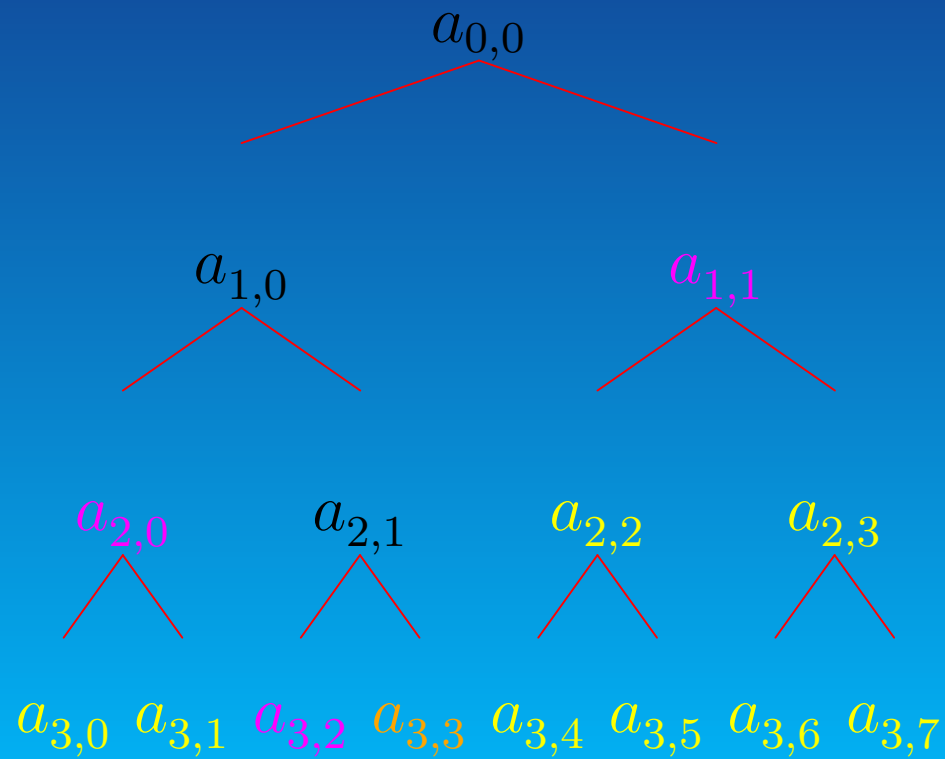
En cache

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



- Rajout du système CryptoPage dans un PC virtuel à base de *x86* (BOCHS)
 - ▶ Processeur en mode chiffré ou non
 - ▶ Processeur en mode superviseur ou non
 - ▶ en mode chiffré, influence \rightsquigarrow arrêt exécution
 - ▶ Pas moyen d'accéder aux registres lors du mode chiffré
 - ▶ Seul moyen de démarrer exécution chiffrée :
descripteur de contexte d'exécution chiffrée
 - ▶ Si programme chiffré interrompu, état
(registres,...) stocké dans un nouveau descripteur



de contexte chiffré

- Extension de Linux 2.6 pour gérer des processus chiffrés
 - ▶ Hypothèse : système d'exploitation éventuellement compromis sans fuite des processus chiffrés
 - ▶ Processus chiffrés opaques au système d'exploitation
- Chaîne logicielle de production de programme chiffrés



- 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
- Complexité compatible avec processeurs généralistes modernes
- Possible d'adapter un vrai système d'exploitation
- Réfléchir aux bonnes et mauvaises utilisations d'un tel système, implications sociales



- Faire une version allégée pour version économique (bus chiffré avec composant mémoire adapté,...)
- Étendre les concepts dans des SoC
- Protection des IP par chiffrement pour éviter vols chez fondeurs ou utilisateurs
- FPGA avec chiffrement asymétrique du programme
 - ▶ Programme reste secret
 - ▶ Possible de mettre à jour *in situ* par le client sans risque de piratage
- Faire des antivols pour composants électroniques



(barrettes mémoires, disques durs, écrans,...)

- ▶ Authentification des composants entre eux
- ▶ Utilisation de canaux cachés pour compatibilité avec matériel standard
- ▶ Tout l'ensemble commandé par une condition logicielle
- ▶ Location de matériel validée par réseau, électricité,...



Table des transparents

Introduction

- 1 Besoins de sécurité
- 3 ; Résister !
- 5 Concepts de base

CryptoPage-1

- 7 CryptoPage-1
- 8 ¿ Attaques par rejeu ?
- 10 Vérificateur de mémoire

Vérificateur

- 12 Arbres de MERKLE
- 13 Arbre de Merkle en cache
- 14 Écriture
- 15 ¿ Pourquoi ça marche ?
- 16 Chiffrement des adresses
- 18 Système d'exploitation

Logiciel

- 20 Conclusion

Conclusion

- 21 Autres besoins
- 23 Table of contents

