

---

# **CryptoPage-2 :** **un processeur sécurisé contre le rejeu**

---

**Guillaume DUC et Ronan KERYELL**

`Guillaume.Duc@enst-bretagne.fr & rk@enstb.org`

—

**Laboratoire Informatique & Télécommunications**

**Département Informatique**

**École Nationale Supérieure des Télécommunications de Bretagne**

**21 octobre 2004**

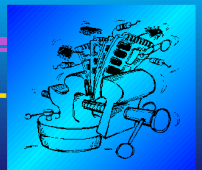
0-1

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 utilisant cryptographie



- 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 (vente de logiciels)
- Protection de contenu multimédia contre le piratage
- Code mobile chiffré et sécurisé (crypto-mobilet). Cas particulier : avions... [Ware, 1970]
- 



Impossible à faire avec des ordinateurs classiques ☹

Dilemme :

- Sécurité  $\Rightarrow$  carte à puce et faible performance
  - Informatique haute performance  $\Rightarrow$  peu de sécurité
- ~> Rajouter mécanismes de sécurité aux processeurs classiques




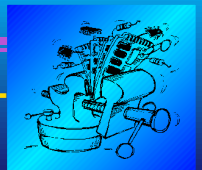
## Détecter voire résister à toute attaque physique ou logicielle




- 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

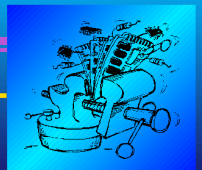
Dans la suite : périmètre de sécurité = processeur



- Toute modification extérieure de l'état du système implique un arrêt voire une destruction
- Transformer tout changement de la sémantique du programme par un attaquant en déni de service
- Pas la peine de faire mieux : difficile de résister à une coupure de l'alimentation électrique...
-  ne gère pas les aspects temporels (protocoles,...)
- Il faut pouvoir exécuter un programme chiffré pas à pas pour des raisons de debug

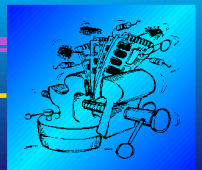


-  Chiffrement de la mémoire
-  Authentification de la mémoire
-  Système et programmation





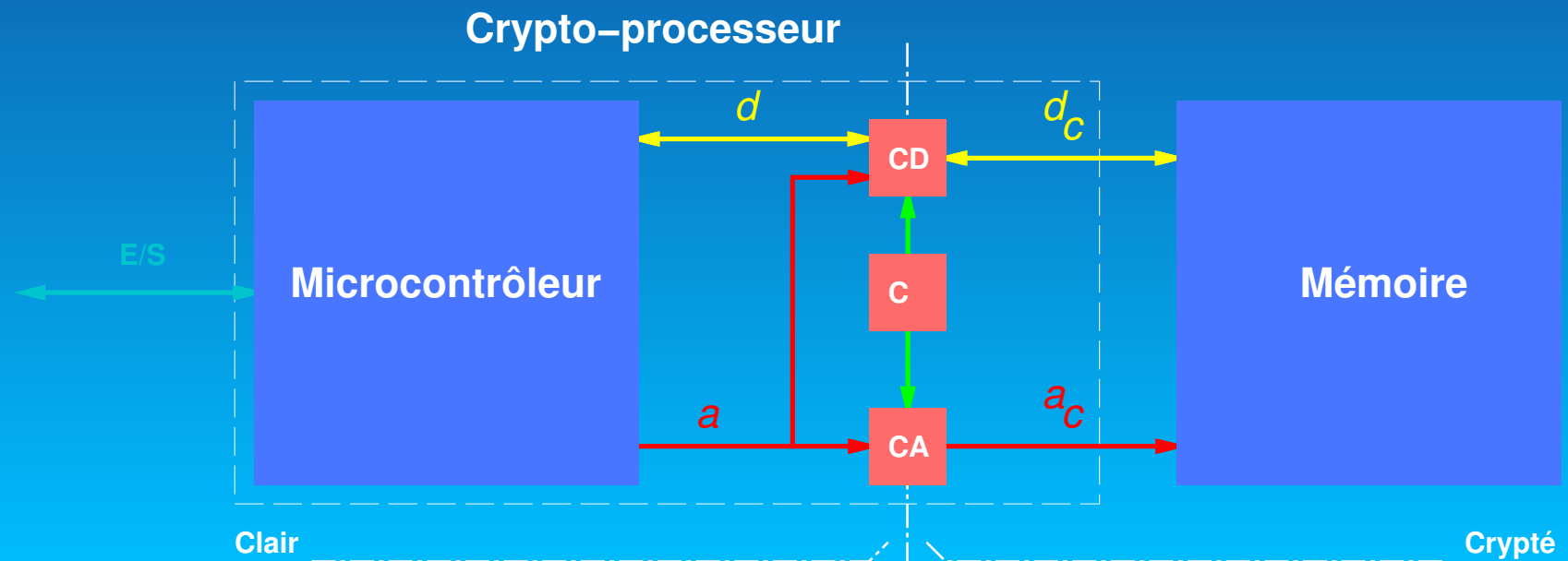
- Spécialisation du processeur : devient unique au monde
  - ▶ Utilisation de cryptographie à clé publique/clé secrète
  - ▶ Chiffrement des instructions et des données
  - ▶ Producteur du logiciel chiffre avec la clé publique du processeur
  - ▶ Processeur exécute le programme en déchiffrant avec sa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





- 8051 sécurisé pour terminaux cartes bancaires [Dat, 1999a, Dat, 1999b]
- Chiffrement des bus du processeur



- ▶ Pour éviter les attaques à texte connu (zones de 0...)

$$d_c = C_D(c_s, a, d)$$

- ▶  Le circuit a été « cassé » [Kuhn, 1998]  
 $C_D$  est bijective : toute instruction lue en mémoire va être exécutée. Jeu d'instruction sur 8 bits : 256 essais d'injection à faire après le RESET   
construction de proche en proche d'un programme sortant sur un port la mémoire déchiffrée !
- Dans les circuits proposés par Robert M. BEST [Best, 1980, Best, 1981, Best, 1984] toute instruction

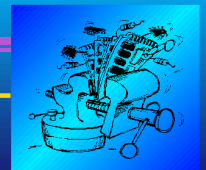


invalide provoque la destruction du processeur.

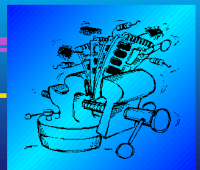


bugs, virus,...

- Beaucoup d'idées de processeurs sur ce principe  
[Gilmont et al., 1998]

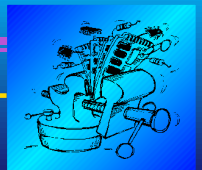


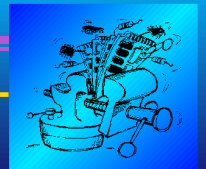
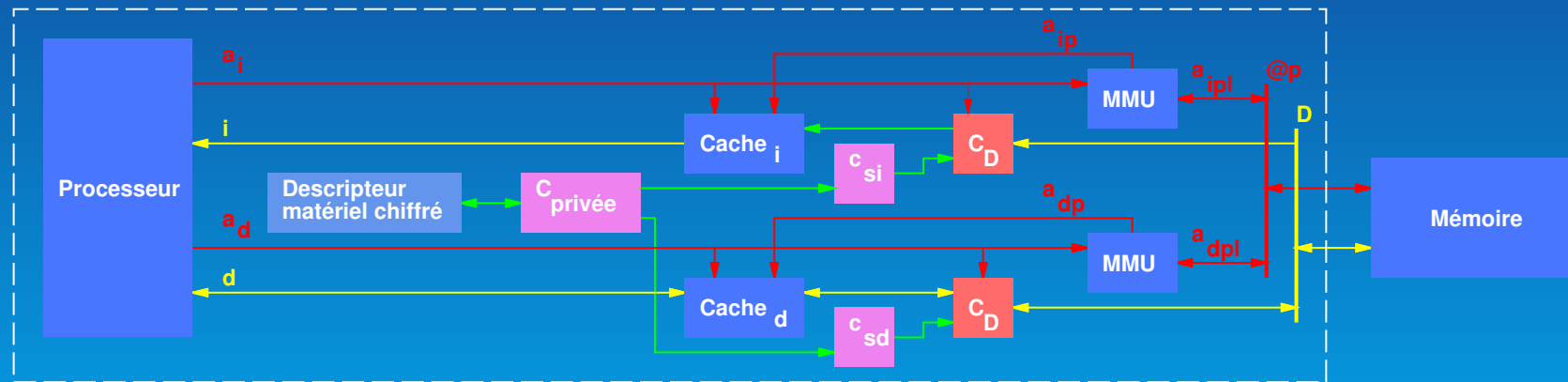
- Rajouter un code d'authentification à des « Pages » ou lignes de cache de données
- Au lieu de chiffrer  $\boxed{d}$  on chiffre  $\boxed{d} \boxed{H(d)}$
- Au déchiffrement on a  $\boxed{d''} \boxed{h''}$  et il suffit de vérifier que  $H(d'') = h''$
- Si  $H$  fournit des valeurs sur  $b_s$  bits la probabilité d'être trompé est de  $2^{-b_s}$  :  
pour  $b_s = 128$ , cela fait une chance sur  $3,4.10^{38}$ , raisonnable dans l'état actuel de la technologie





- 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

[Keryell, 2000]







- Rajout de  $b_s$  bits par page de  $b$  bits...
-  Que faire de ces  $b_s$  bits supplémentaires de manière transparente ?
- Corollaire : modifier l'adressage mémoire dans le traducteur  
*adresse virtuelle* 

*adresse physique*

- 2 solutions étudiées
- Solution dilatée plus simple pour les programmes :  
connexité est préservée (sauvegarde sur disque,...). Une adresse  $a$  se retrouve

CryptoPage

—CryptoPage-1—



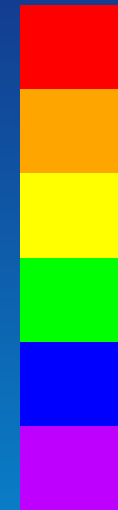
en :

$$\left[ \left\lfloor \frac{a}{b} \right\rfloor \frac{b + b_s}{b}, \left\lfloor \frac{a}{b} + 1 \right\rfloor \frac{b + b_s}{b} - 1 \right]$$

Avant



Dilaté



Éclaté




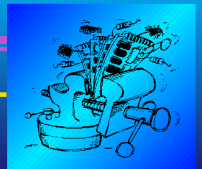
$A_s \rightarrow$



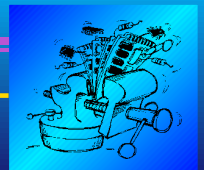
- Pas de chiffrement au niveau adresse (au moins au niveau des pages)
- Bénéficie du support système de la mémoire virtuelle




-  Faire attention aux programmes gérant simultanément des programmes/données chiffrés et en clair : éviter les conflits d'adresse  $\rightsquigarrow$  géré par `malloc()`
- Si chaînage à la CBC : encore des bits supplémentaires



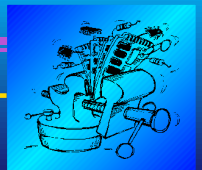
- Chiffrement de la mémoire
- ✎ Authentification de la mémoire
- Système et programmation



- 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



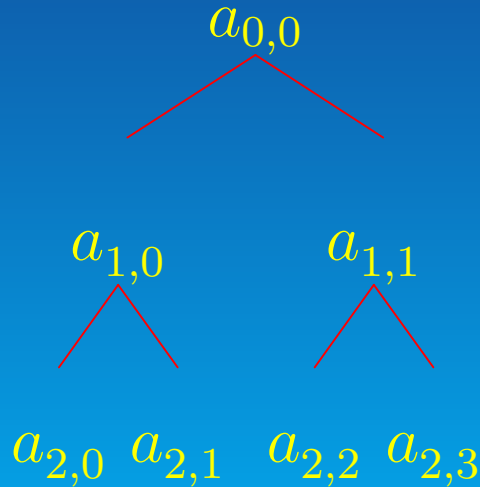
- Utiliser les mamelles accélératrices de l'informatique : **hiérarchie**, **cache** et **parallélisme**





- Fonction de hachage cryptographique 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



Algorithme de lecture vérifiée $\mathcal{R}_V(i, j)$	Algorithme d'écriture vérifiée $\mathcal{W}_V(a_{i,j}, i, j)$
$a_{i,j} = \mathcal{R}(i, j)$ <b>tant que</b> $(i > 0)$ $f = j \oplus 1; p = \lfloor \frac{j}{2} \rfloor$ $a_{i,f} = \mathcal{R}(i, f); a_{i-1,p} = \mathcal{R}(i-1, p)$ <b>si</b> $(a_{i-1,p} \neq H(a_{i,\min(j,f)}, a_{i,\max(j,f)}))$ <b>erreur</b> <i># On remonte l'arbre :</i> $i = i - 1; j = p$ <b>renvoie</b> $a_{i,j}$	<b>si</b> $(i > 0)$ $f = j \oplus 1; p = \lfloor \frac{j}{2} \rfloor$ $a_{i,f} = \mathcal{R}_V(i, f)$ $a_{i-1,p} = H(a_{i,\min(j,f)}, a_{i,\max(j,f)})$ <i># On remonte l'arbre :</i> $\mathcal{W}_V(a_{i-1,p}, i-1, p)$ $\mathcal{W}(a_{i,j}, i, j)$



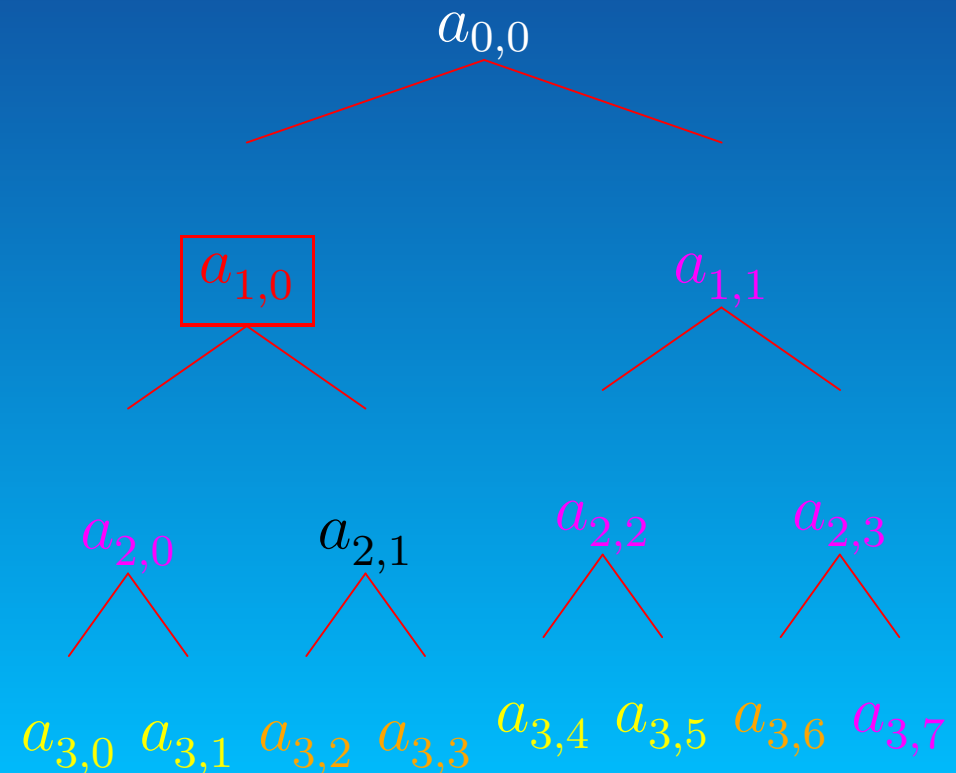
## Cache

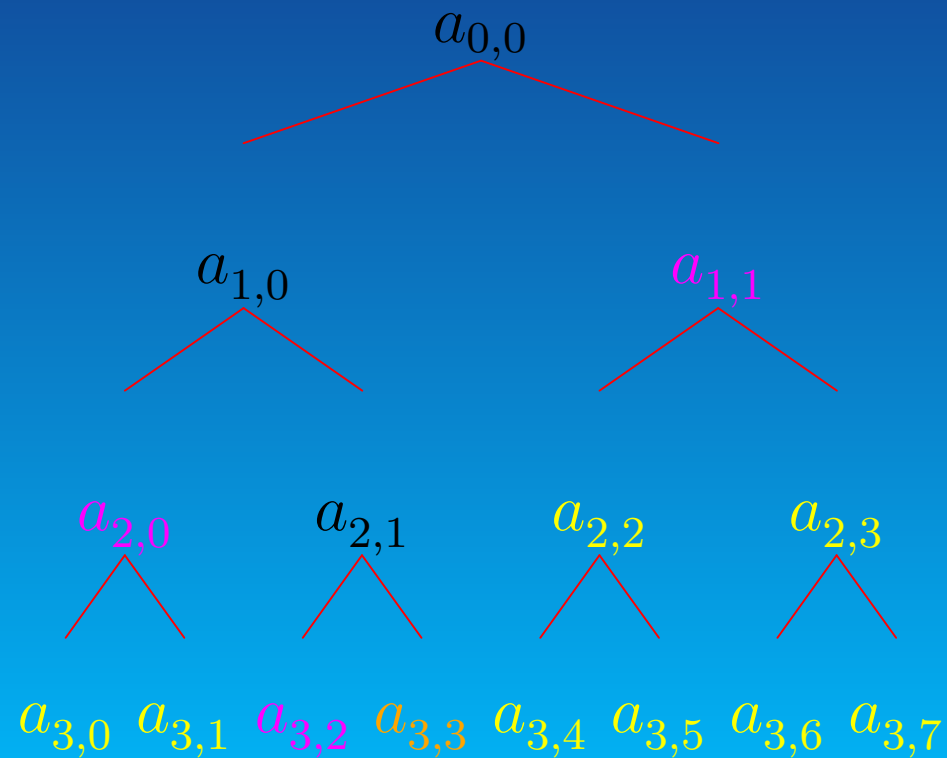
En cache

Lecture depuis la mémoire

Conflit détecté

Calcule la fonction de hachage





Algorithme de lecture vérifiée $\mathcal{R}_{\mathcal{VC}}(i, j)$	Algorithme d'écriture vérifiée $\mathcal{W}_{\mathcal{VC}}(a_{i,j}, i, j)$
<b>si</b> $(i = 0 \vee \mathcal{C}(a_{i,j}) = \text{hit})$ <b>renvoie</b> $\mathcal{R}_{\mathcal{C}}(i, j)$ $f = j \oplus 1; p = \lfloor \frac{j}{2} \rfloor$ $a_{i-1,p} = \mathcal{R}_{\mathcal{VC}}(i-1, p)$ $a_{i,j} = \mathcal{R}(i, j); a_{i,f} = \mathcal{R}(i, f)$ <b>si</b> $(a_{i-1,p} \neq H(a_{i,\min(j,f)}, a_{i,\max(j,f)}))$ <b>erreur</b> $\mathcal{W}_{\mathcal{C}}(a_{i,j}, i, j); \mathcal{W}_{\mathcal{C}}(a_{i,f}, i, f)$ <b>renvoie</b> $a_{i,j}$	<b>si</b> $(i > 0)$ $f = j \oplus 1; p = \lfloor \frac{j}{2} \rfloor$ $a_{i,f} = \mathcal{R}_{\mathcal{VC}}(i, f)$ $a_{i-1,p} = H(a_{i,\min(j,f)}, a_{i,\max(j,f)})$ <i># On remonte l'arbre</i> $\mathcal{W}_{\mathcal{VC}}(a_{i-1,p}, i-1, p)$ $\mathcal{W}_{\mathcal{C}}(a_{i,j}, i, j)$




- Injection de fausses données ( $a'_{3,3}$ ) impossible à cause du hachage cryptographique  
→ 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
- Si adresse intervient dans calcul hachage : résistance à la permutation



-  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 obscurcissement, 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





- Pour mieux résister
  - ▶ Faire suivre tout chiffreur par un déchiffreur et réciproquement pour vérifier l'intégrité physique
  - ▶ Faire voter plusieurs chiffreurs/déchiffreurs pour détecter *glitches*, variations de fréquence d'horloge, température, rayonnements ionisants,...
  - ▶ Cible : processeurs de  $10^7$  ou  $10^8$  transistors  $\rightsquigarrow$  coût faible de la redondance

Destruction du processeur si incohérence

- Tests
  - ▶ Suppression du JTAG en mode chiffré




- ▶ Possibilité de faire un test en mode chiffré en usine puis grillage de *plusieurs* fusibles
- Étudier le graphe de dépendance afin de prouver qu'une donnée chiffrée ne peut pas être dirigée vers une donnée non chiffrée



- Chiffrement de la mémoire
- Authentification de la mémoire
- ✎ Système et programmation



- Classiquement
  - ▶ Le SE peut tout faire : Contrôle le matériel
  - ▶ On doit faire confiance au SE...
  - ▶ Un SE : dizaines de millions LOC...
- CryptoPage
  - ▶ Ne fait pas confiance au SE
  - ▶ Rajoute des primitives matérielles de manipulation de base pour le SE
  - ▶ Garde dans le SE le gros du travail (décisions,...)
  - ▶  Le SE peut faire du déni de service ou ne pas



faire ce qu'on attend : à gérer au niveau utilisateur...



- Processeur peut être en mode chiffré ou non
- Processeur peut être en mode superviseur ou non
- En mode chiffré, on ne peut influencer l'exécution depuis l'extérieur si ce n'est en interrompant l'exécution
- Il n'y a aucun moyen d'accéder aux registres lors du mode chiffré (en dehors du processus en cours d'exécution bien sûr), que ce soit par des moyens matériels, des registres internes, des remises à 0, des interruptions logicielles ou matérielles,...
- Contexte d'exécution chiffrée contient une clé de

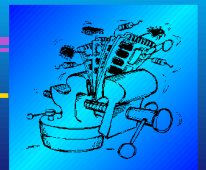


chiffrement symétrique du programme, une clé de chiffrement symétrique des données, des contenus de registres, la valeur de la racine de l'arbre de vérification ;

- Seul moyen de démarrer une exécution chiffrée est de partir d'un descripteur de contexte d'exécution chiffrée qui est alors déchiffré via la clé secrète du processeur
- Si un programme chiffré est interrompu, son état (registres,...) est stocké dans un nouveau descripteur de contexte chiffré avec la clé secrète du processeur. Ainsi le programme peut-être continué plus tard sans



fuite d'information.





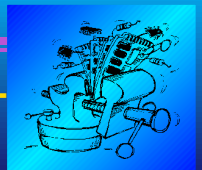
- Nouvelles instructions (version RISC)
  - ▶ RTIEC  $r$  : *ReTurn from Interrupt on an Enciphered Context*
  - ▶ LDNC  $r_d, r_a, r_{ac}$  : *LoaD Non-Ciphered*
  - ▶ STNC  $r_d, r_a, r_{ac}$  : *STore Non-Ciphered*
- Exemple dans Unix (sources disponibles, robuste et souvent libre)
  - ▶ Un processus peut être  $[\text{root}, \text{user}] \times [\text{clair}, \text{chiffré}]$
  - ▶ Même root (piratable,...) ne peut pas espionner un processus chiffré



- ▶ Pour partager de l'information chiffrée entre processus : partage d'une clé commune
- ▶ Déverminage d'un processus chiffré : le dévermineur doit être chiffré avec la même clé

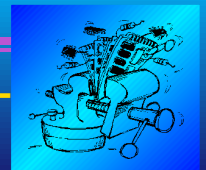


- Un processus chiffré doit pouvoir utiliser les services du noyau non chiffré
- Déchiffrage *par le processus chiffré* des arguments avant le trap
  - ▶ Pas d'édition de lien dynamique possible directement
  - ▶ Compatibilité système ascendante à gérer au niveau trap
- Appels systèmes exportent/importent des données chiffrées
  - ▶ Pour garder la sémantique Unix, les tailles des

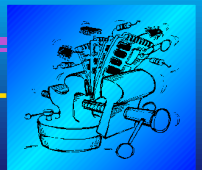


données sont celles des données chiffrées

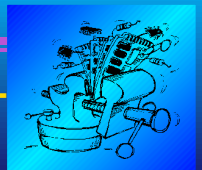
- Bibliothèques standards : existent en version chiffrées et en clair
  - ▶ `malloc()` et `calloc()` alloue la place supplémentaire pour la signature et aligne tout sur des blocs entiers



- Rajout du système CryptoPage dans un PC virtuel à base de *x86* (BOCHS)
  - ▶ Privilège ordinateur complet à environnement de test
  - ▶ 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



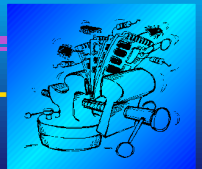
- ▶ Si programme chiffré interrompu, état (registres,...) stocké dans un nouveau descripteur de contexte chiffré
- Extension de Linux 2.6 pour gérer des processus chiffrés
  - ▶ Hypothèse : système d'exploitation éventuellement compromis sans fuite des processus chiffrés
  - ▶ Processus chiffrés opaques au système d'exploitation
- Chaîne logicielle de production de programme chiffrés





- Un processus peut être interrompu et relancé par le SE (multi-tâche)
- Si un processus est rejoué cela ne peut-être que globalement avec données d'origine
  - ▶ Permet la mise au point
  - ▶ Permet les `fork()` Unix
- Mais le rejeu global peut aussi être néfaste...  
Mécanisme de type télé-rupteur
  - ▶ Si cela est néfaste (effets de bords), utiliser un protocole sécurisé avec le monde extérieur (d'autres processeurs sécurisés)



- ▶ Ajouter un mécanisme de « *cookies* » permanents dans le processeur. Chaque processus chiffré peut stocker une valeur dans ce cookie pour éviter le rejeu
- ▶ Construction de mécanismes d'anti-rejeu de plus haut niveau à partir de ces cookies élémentaires
- ▶ Développement de processus sécurisés serveurs anti-rejeu pour économiser de la mémoire de cookies



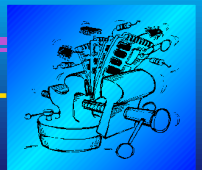



- Déclencher exécution de code utilisateur sur événement système
-  Le SE peut exécuter du code arbitraire dans processus chiffré ?
- *A priori* que des fonctions utilisateurs enregistrées auprès du SE via `sigaction()` ou `signal()`
- Besoins
  - ▶  Empêcher que le SE puisse démarrer une exécution arbitraire à n'importe quelle adresse
  - ▶ Exécuter dans un contexte de vérification de

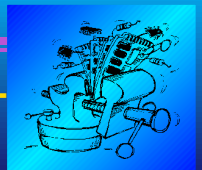


mémoire correct et mise à jour du contexte chiffré  
du processus lourd

-  Le programmeur doit tenir compte d'un SPAM de signaux par le SE...

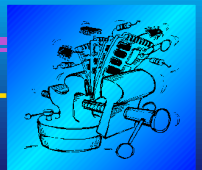


- Rajout au contexte chiffré une adresse d'exécution du portail
- Rajouter instruction `signal` pour déclencher exécution adresse portail d'un contexte chiffré
- Par rapport à SE classique, multiplexage des signaux fait en mode utilisateur et non système pour simplifier implémentation ie `sigaction()` majoritairement une bibliothèque utilisateur
- Signaux eux-mêmes interruptibles...  sauvegarde du contexte courant
  - ▶ Garantir une restauration des contextes dans

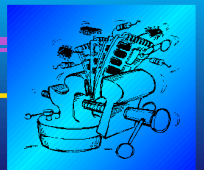


l'ordre... Sinon sémantique séquentielle non garantie et attaque possible ☹

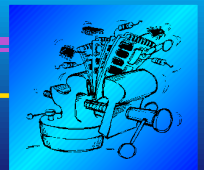
- ▶ Utilise un mécanisme de chaînage des descripteurs interrompus au niveau des valeurs de hachage de MERKLE pour le garantir
- Mécanismes plus génériques pour faire du multi-thread ou des signaux mal imbriqués encore à clarifier...



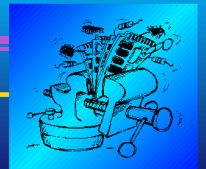
- Difficile sans réalisation ou modélisation fine...
- Commencée avec SimpleScalar (mesures précises mais par ordinateur complet) mais passée sur BOCHS (ordinateur complet, mais pas de mesure de performance...)
- Système entre cache(s) interne(s) et mémoire ou cache(s) externe(s)
  - ▶ Utilise pleinement le(s) cache(s) interne(s)
  - ▶ Ralentissement que sur les accès externes
- Différents points de ralentissement



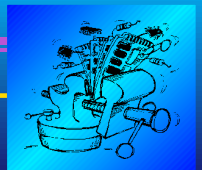
- Différents algorithmes de chiffrement possible



- Chiffrement asymétrique à clé publique : très lent
- Utilisé que lors des changements de processus.  
Pourrait utiliser du chiffrement à clé secrète... Mais si hibernation ?
- 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

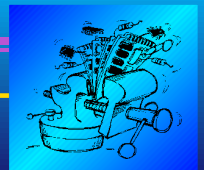


- Utiliser un système d'exploitation prenant en compte CryptoPage : reprendre les concepts classiques de l'ordonnanceur à 2 niveaux pour tenir compte de la mémoire d'échange secondaire
- Prise en compte de l'EPDB par l'ordonnanceur du SE
  - ▶ Préchargement d'un EPD
  - ▶ Exécution avec un EPD dans l'EPDB
  - ▶ 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,...



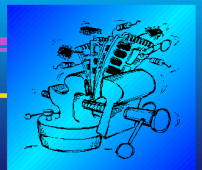
- Utilisation algorithme symétrique avec clef de session dans EPD
- Algorithmes assez rapides
- AES pipelinable en 16 cycles
- Accélération des accès mémoire à la IBM360/91
  - ▶ Garder une file des accès mémoire en cours de chiffrement
  - ▶ Profondeur en fonction du chemin d'écriture en mémoire
  -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



- Calcul des condensés assez lents
  - ▶ SHA-1 génère 160 bits en 80 cycles
  - ▶ Utiliser d'autres fonctions de hachage
- Exécution spéculative optimiste : cas courant on n'est pas piraté...
- Astuce d'AEGLIS : utiliser une fonction de hachage simple jointe au bloc qui sera chiffré de toute manière ! Plus rapide mais résistance à étudier



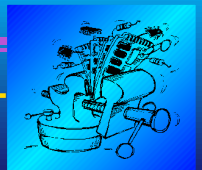
- Plus prospectif :
  - ▶ Réfléchir à la parallélisation du système
    - ↪ Problème de mises à jour concurrentes de l'arbre
  - ▶ Faire du SMT entre différents cryptoprocessus
    - ↪ Typage des lignes de cache par cryptoprocessus



- Cartes à puces virtuelles
- Protection des serveurs WWW,...
- *Dongles* virtuels
- Routeurs à logiciel chiffré
- Machines virtuelles
- Agents secrets (code mobile, Applets,...)
- Grilles de calcul chiffrées
- Antivols électroniques pour ordinateur

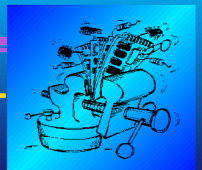


- Faire une version allégée pour version économique (bus chiffré avec composant mémoire adapté,...)
- Étendre les concepts dans des SoC
- Protection des IP par chiffrement pour éviter vols chez fondeurs ou utilisateurs
- FPGA avec chiffrement asymétrique du programme
  - ▶ Programme reste secret
  - ▶ 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,...



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





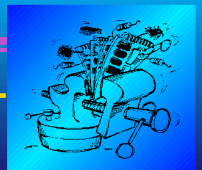
- Isolation sécurisée des divers processus (carte multi-applications)
- Intégration dans un système d'exploitation libre (JayaCard, \*BSD, Linux,... ?)



- Programme Incitatif GET OpenSmartCard et Sec-SoC pour développer des cartes à puces sécurisées à la CryptoPage et des briques pour construire des system-on-chips sécurisés
- Programme Incitatif GET SemaGRID de système de fichier sémantique sécurisé sur grille
  - ▶ Application sécurisée à la CryptPage
  - ▶ Comment construire une couche de SE sûre au dessus d'un SE non sûr (usurpation des appels systèmes,...) ?



- Beaucoup de projets ne résistant pas à une attaque par rejeu
- Grand public : TCPA/Palladium/XBox/... contourne le problème en supposant que tout loge dans une mémoire sécurisée interne au processeur, ne résiste pas aux attaques de bus
- Projet de vérificateur hors-ligne au MIT [Clarke et al., 2003]
- Processeur AEGIS au MIT [Suh et al., 2003] : approche semblable à CryptoPage-2
  - ▶ Ont fait des modélisations ! -25 % en



performance. Raisonnable

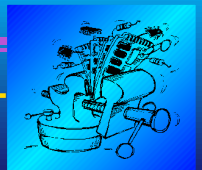
- ▶ Vérification en écriture lors de la sortie du cache
- ▶ Implantation de l'arbre de MERKLE non détaillée
- ▶ Organisation du cache dans CryptoPage-2 ne nécessitant pas de cryptographie incrémentale
- ▶ Sortie de cache optimisée dans CryptoPage-2 car cache de l'arbre
- ▶ Pas de chiffrement des adresses
- ▶ Pas de prise en compte des problèmes d'OS
- ▶ Ont regardé le partage de l'authentification dans un SMP [Clarke et al., 2004]



- eXecute Only Memory (XOM) [Lie et al., 2000]
  - ▶ Pas d'authentification globale
  - ▶ Pensaient que les SE allaient de soi... ☺
  - ▶ Reproposent quelque chose de compatible avec un SE : portage d'IRIX [Lie et al., 2003]
- Gestion multiprocesseur de la vérification mémoire [Shi et al., 2004] à Georgia Tech
  - ▶ Déportent l'arbre de MERKLE dans un contrôleur mémoire
  - ▶ Chaque processeur accède à ce contrôleur via un bus sécurisé

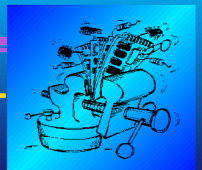



- Domaine d'avenir pour des processeurs enfin sécurisés
- Domaine transversal vaste pour garantir sécurité sans sacrifier les performances
- Modélisation fine et réalisation concrète à faire
- Complexité compatible avec processeurs généralistes modernes
- Possible d'adapter un vrai système d'exploitation
- Complexité fonctionnelle considérable des processeurs et des SE... Certains problèmes ne

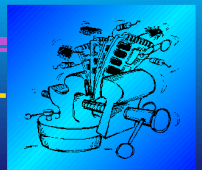


peuvent être détectés qu'en réalisant ☹

- Réfléchir aux bonnes et mauvaises utilisations d'un tel système, implications sociales
- Pas de solution satisfaisante pour les contenus artistiques numériques (qui est prêt à se faire greffer des implants neuronaux ?)
- Démonstration...



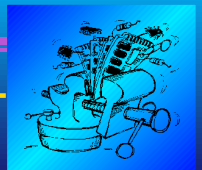
- Avoir une clause au contrat de vente permettant de transférer un programme vers un autre ordinateur (panne, vol,...)
-  On ne peut même plus savoir ce qui tourne sur son ordinateur
  - ▶ Bugs irréparables
  - ▶ Comportements cachés
- Mais la boîte de Pandore existe déjà
  - ▶ Actuellement c'est difficile (OS propriétaires,...), même avec les sources (empilements logiciels)





gloutons,...)

- ▶ Il y a déjà des clauses interdisant la rétro-ingénierie sur des contrats de logiciels et des comportements cachés
- Rien que l'identifieur unique du Pentium III a provoqué des réactions...
- On n'a pas le droit de regarder ses DVD avec des logiciels libres



# References

[Dat, 1999a] (1999a). « *DS5000FP Soft Microprocessor Chip* ». Dallas Semiconductor.

<http://www.dalsemi.com/DocControl/PDFs/5000fp.pdf>.

[Dat, 1999b] (1999b). « *DS5002FP Secure Microprocessor Chip* ». Dallas Semiconductor.

<http://www.dalsemi.com/DocControl/PDFs/5002fp.pdf>.

[Best, 1980] BEST, R. M. (1980). « Preventing Software Piracy with Crypto-Microprocessors ». In *Proc. IEEE Spring COMPCON'80*, pages 466–469.

[Best, 1981] BEST, R. M. (1981). « Crypto Microprocessor for Executing Enciphered Programs ». Technical Report US4278837, United States Patent. Consulté le 21 février 2000 sur

[http://patent.womplex.ibm.com/details?&pn=US04278837\\_\\_](http://patent.womplex.ibm.com/details?&pn=US04278837__).

[Best, 1984] BEST, R. M. (1984). « Crypto Microprocessor that Executes Enciphered Programs ». Technical Report US4465901, United States Patent. Consulté le 21 février 2000 sur

<http://patent.womplex.ibm.com/detail>

[Clarke et al., 2003] CLARKE, D., SUH, G. E., GASSEND, B., van DIJK, M., and DEVADAS, S. (2003). « Offline Integrity Checking of Untrusted Storage ». In *Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*.

[Clarke et al., 2004] CLARKE, D., SUH, G. E., GASSEND, B., van DIJK, M., and DEVADAS, S. (2004). « Checking the Integrity of Memory in a Snooping-Based Symmetric Multiprocessor (SMP) System ». Technical Report, MIT Laboratory for Computer Science.

[Gilmont et al., 1998] GILMONT, T., LEGAT, J.-D., and QUISQUATER, J.-J. (1998). « An Architecture of Security Management Unit for Safe Hosting of Multiple Agents ». In *Proceedings of COST-254*, pages 79–82.

<http://ldos.fe.uni-lj.si/cost254/pap>

[Keryell, 2000] KERYELL, R. (2000). « CryptoPage-1 : vers la fin du piratage informatique ? ». In

CryptoPage

—Conclusion—



*Symposium d'Architecture (SympA'6)*, pages 35–44, Besançon, France.

[http://www.cri.enscm.fr/~kerspell/publications/ENSTBr\\_INFO\\_200](http://www.cri.enscm.fr/~kerspell/publications/ENSTBr_INFO_200)

[Kuhn, 1998] KUHN, M. G. (1998). « Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP ». *IEEE Transactions on Computers*, 47(10):1153–1157.

[Lie et al., 2003] LIE, D., THEKKATH, C. A., and HOROWITZ, M. (2003). « Implementing an Untrusted Operating System on Trusted Hardware ». In *SOSP'03*.

[Lie et al., 2000] LIE, D., THEKKATH, C. A., MITCHELL, M., LINCOLN, P., BONEH, D., MITCHELL, J. C., and HOROWITZ, M. (2000). « Architectural Support for Copy and Tamper Resistant Software ». In *Architectural Support for Programming Languages and Operating Systems*, pages 168–177.

[Shi et al., 2004] SHI, W., LEE, H.-H. S., GHOSH, M., and LU, C. (2004). « Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems ». In *Proceedings of PACT 2004 — Parallel Architecture and Compilation Techniques*.

[Suh et al., 2003] SUH, G. E., CLARKE, D., GASSEND, B., van DIJK, M., and DEVADAS, S. (2003). « The AEGIS Processor Architecture for Tamper-Evident and Tamper-Resistant Processing ». Technical Report, MIT Laboratory for Computer Science.

[Ware, 1970] WARE, W. H. (1970). « Security Controls for Computer Systems ». Technical Report, The Rand corporation.

<http://www.rand.org/publications/>



# Table des transparents

## Introduction

- 1 Besoins de sécurité
- 4 ; Résister !
- 5 Dénier de service
- 6 Agenda

## CryptoPage-1

- 7 Concepts de base
- 9 Exemple du crypto-processeur DS5002FP
- 12 Vérification de la mémoire
- 13 Améliorations
- 14 CryptoPage-1

- 15 Adressage mémoire
- 17 Agenda
- 18 ¿ Attaques par rejeu ?
- 20 Vérificateur de mémoire

## Vérificateur

- 22 Arbres de MERKLE
- 24 Arbre de Merkle en cache
- 25 Écriture
- 27 ¿ Pourquoi ça marche ?
- 28 Chiffrement des adresses
- 30 Mise en œuvre
- 32 Agenda
- 33 Système d'exploitation

## Logiciel

CryptoPage

—Conclusion—



35 Hypothèses matérielles

38 Mise en œuvre logicielle

40 Appels systèmes

42 Réalisation

44 La vie est un long fleuve tranquille...

46 Signaux à la Unix

48 Portail d'interruption

50 Idées de performance

## Performances

52 D(é)Chiffrement des descripteurs de processus

55 Chiffrement des lignes de cache

56 Vérification de la mémoire

58 Quelques applications

## Applications

59 Autres besoins

61 OpenSmartCard

63 Participation à d'autres projets

64 Autres projets en rapport dans la littérature

## Conclusion

67 Conclusion

69 Éthique

71 Bibliographie

72 Liste des transparents

