0-0

CryptoPage
un processeur à mode d'exécution sécurisée et hautes performances

Cyril BRÛLEBOIS, Guillaume DUC & Ronan KERYELL

—

High Performance Computing Architectures & Security

Département Informatique

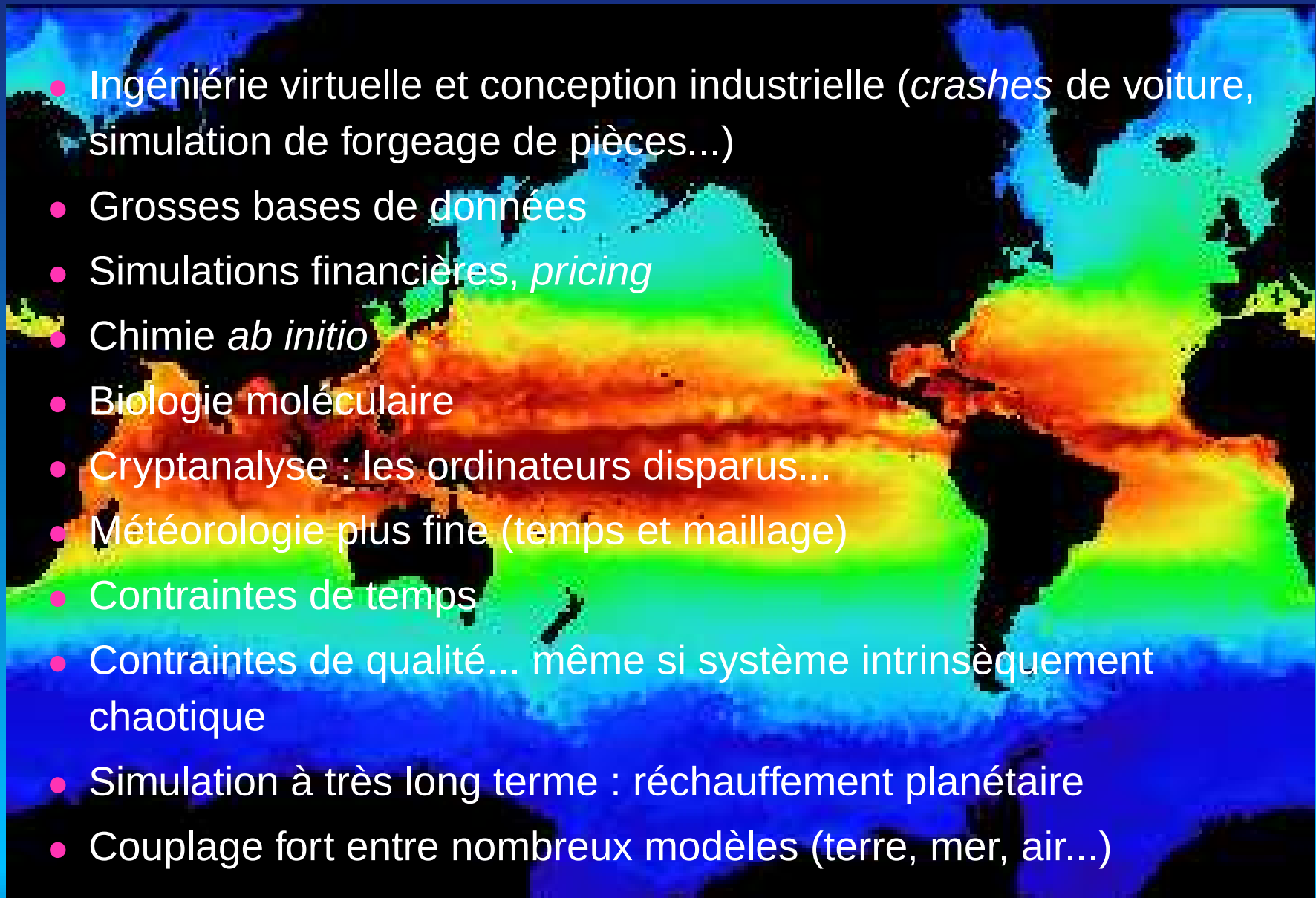
École Nationale Supérieure des Télécommunications de Bretagne

SSTIC 2007

30 mai 2007

Résumé

Durant ces dernières années, plusieurs architectures informatiques sécurisées ont été proposées. Elles chiffrent et vérifient le contenu de la mémoire afin de fournir un environnement d'exécution résistant aux attaques. Quelques architectures, comme notamment HIDE, ont aussi été proposées pour résoudre le problème de la fuite d'informations via le bus d'adresse du processeur. Cependant, malgré l'importance de ces mécanismes, aucune solution pratique combinant le chiffrement, la vérification de l'intégrité mémoire ainsi qu'une protection contre la fuite d'informations n'a encore été proposée, à un coût raisonnable en terme de performances. Dans cet article, nous proposons CryptoPage, une architecture qui implémente ces trois mécanismes avec un impact faible sur les performances (de l'ordre de 3 %). Nous présentons enfin un exemple d'utilisation pour construire des grilles de calcul sécurisées.



- Ingéniérie virtuelle et conception industrielle (*crashes* de voiture, simulation de forgeage de pièces...)
- Grosses bases de données
- Simulations financières, *pricing*
- Chimie *ab initio*
- Biologie moléculaire
- Cryptanalyse : les ordinateurs disparus...
- Météorologie plus fine (temps et maillage)
- Contraintes de temps
- Contraintes de qualité... même si système intrinsèquement chaotique
- Simulation à très long terme : réchauffement planétaire
- Couplage fort entre nombreux modèles (terre, mer, air...)

Militaire avec moratoire sur essais ⇝ simulations fines

- Projets ordinateurs ASCI aux USA
- Machine TERA10 du CEA/DAM en France, Saclay
  - 4000 camions de terre enlevée
  - 4352 processeurs Intel Montecito Itanium 2 (soit 8704 cœurs, SMT), chacun de 1 720 000 000 transistors...
  - 7ème au TOP500 avec 53 TFLOPS LINPACK
  - 30 To mémoire
  - 54 serveurs d'E/S
  - 1 Po sur 7800 disques

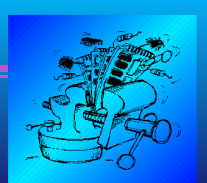
Mais le vrai marché est ailleurs...

- Électronique grand public
- Téléphones portables
- Systèmes communicants sur batterie ou sur pile : économies d'énergie
- *Smart dust*
- Réseaux *ad hoc*
- Systèmes embarqués
- Les SoC-SiP aujourd'hui ≡ supercalculateurs d'antan

Bénéficient des optimisations développées dans le domaine des supercalculateurs

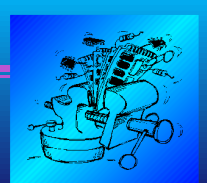
Besoins élémentaires :

- Société de l'information ⇝ besoin de confiance en technologies numériques
  - Point de vue utilisateur : pas de fuites de données personnelles
  - Point de vue architecte système : empêcher des attaques
- 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



- Jeux P2P sans triche
- Protection de contenu multimédia contre le piratage & DRM
- Données médicales sensibles qui voyagent partout
- Vote électronique
- Code mobile chiffré et sécurisé (crypto-mobilette)
- Grilles de calcul (*Grid computing*) sécurisées en milieu naturel (≡ hostile)
  - On ne fait pas confiance à ordinateur distant qui fait tourner son programme...
  - 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

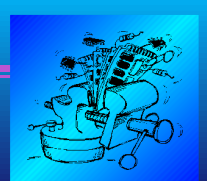
  ⇝ Calcul distribué : asymétrie de la confiance



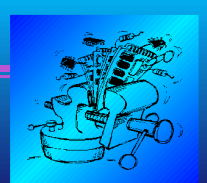
- SoC complexes avec assemblage de nombreux composants et logiciels avec sécurité non prouvable

Impossible à faire avec des ordinateurs/processeurs classiques ☹

- Toutes applications sécurisées reposent sur *une hypothèse* : **matériel sous-jacent sécurisé**, *donc* système d'exploitation aussi ☺
- ... jamais vrai, sauf dans cartes à puce et autres JavaCards ! ☹
- ✷— Besoin de plateformes sécurisées à haute performance

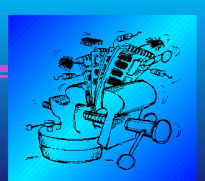


- Invulnerabilité : secrets hermétiquement scellés dans circuits électroniques
- Transparence : matériel spécifique pour ne pas miner performances
- Extensible : algorithmes cryptographiques et protocoles pour élargir zone de confiance



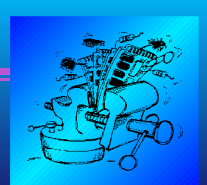
- Corneille au 21ème siècle
- ... ce n'est pas un choix cornélien

¡ Tout le monde choisit la performance ! ☺

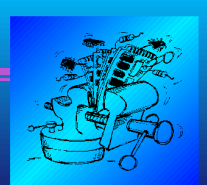


Trusted Computing Platform (TCP/IP ☺)

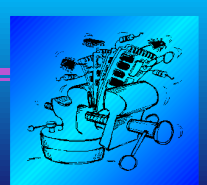
- Télécom Paris : électronique
  - Chaîne de CAO pour faible émission électromagnétique
  - Résistance aux analyses de consommation électrique statistique
  - Logique asynchrone
  - Conception des opérateurs cryptographiques de base
  - FPGA sécurisés
- ENST Sophia : MP-SoC
  - Bus chiffrés
  - SoC sécurisés
  - Systèmes d'exploitation prouvés (L4 avec CEA LIST)



- ENST Bretagne : processeur générique CryptoPage
  - Chiffrement données et programmes
  - Détection des attaques
  - Systèmes d'exploitation
- Proposer panoplie de protections avec différents rapports qualité-prix



- Exécution sécurisée
  - USAF (WARE, 1970)
  - Processeurs de BEST (1979)
  - Dallas DS5002FP (1995) and DS2252T module
  - Hardware Security for Software Privacy Support (GILMONT *et al.*, 1998)
  - XOM (Encryption : 3DES or AES, Integrity : HMAC, no replay protection) (LIE *et al.*, 2000)
  - Aegis (chiffrement : AES CBC, integrité : cached MERKLE trees or multiset hash functions) (SUH *et al.*, 2003)
  - CryptoPage (Encryption : AESCBC, Integrity : cached MERKLE trees, ciphered addresses) (KERYELL *et al.*, 2000)
- Éviter fuites d'information

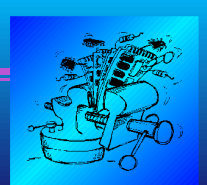


- Oblivious RAM (GOLDREICH, 1987)
- Dallas DS5002FP (1995), broken by KUHN (1998)
- HIDE (ZHUANG *et al.*, 2004)
- Improved version of HIDE presented at PACT'06 by GAO *et al.*

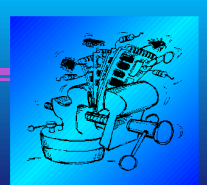
- Trusted Computing Group et Trusted Platform Module
  - Stockage sécurisé (scellé)
  - Attestation à distance
  - Chaîne de confiance des différents maillons logiciels
  - Mais vulnérable aux attaques matérielles en dehors du TPM

Objectifs de CryptoPage

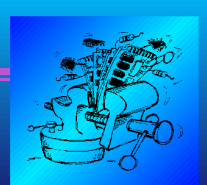
- Concevoir une architecture garantissant deux propriétés aux processus sécurisés :
  - *Propriété de confidentialité* : un attaquant doit pouvoir obtenir le moins d'information possible sur le code ou les données d'un processus
  - *Propriété d'intégrité* : bonne exécution d'un processus non altérable par attaque sauf dénis de service
- Dénis de service non considéré car inévitables (contrôle de l'alimentation...)
- Exécution des processus sécurisés en parallèle avec d'autres processus, sécurisés ou non
- Tout est sous contrôle total d'un attaquant (OS, administrateur mémoire, etc.), excepté le processeur
- Garder des performances raisonnables



- Tout ce qui est en dehors du processeur est sous contrôle total de l'attaquant
  - Bus
  - Mémoire
  - Disques
  - Entrées-sorties
  - Système d'exploitation
  - ...
- Néanmoins processeur ne peut pas être modifié ou espionné
  - On ne considère pas attaques par analyse de consommation SPA, DPA, temporelle, canaux cachés...
  - À gérer ailleurs en dehors du projet. Cf projet GET TCP



- Protection contre les fuites d'information
- Chiffrement mémoire
- Protection de l'intégrité mémoire contre attaques
  - Injection d'une valeur à une adresse mémoire
  - Permutation spatiale
  - Permutation temporelle (attaque par rejeu)

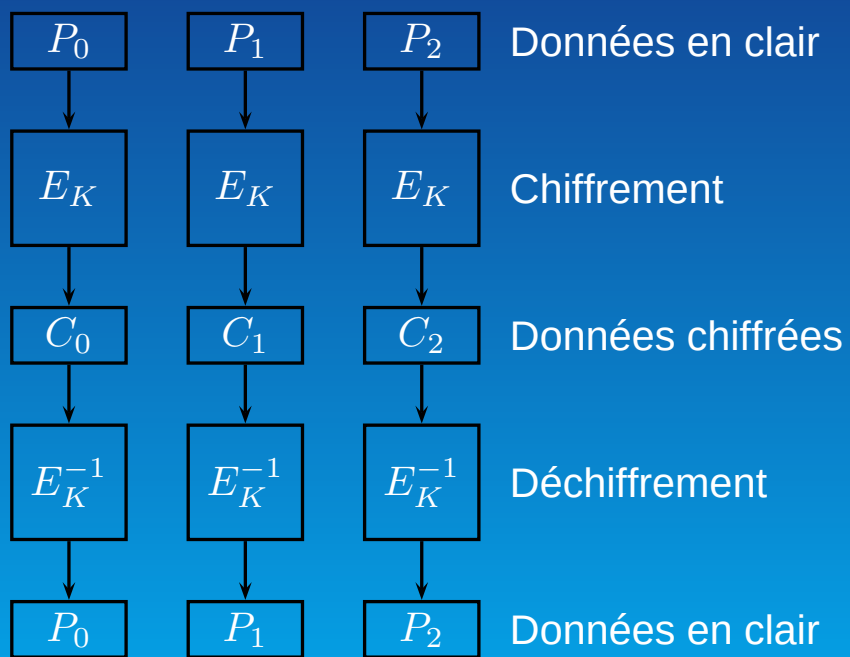


- $H : \{0,1\}^* \mapsto \{0,1\}^n$ avec n fixe et de petite taille (128 bits pour MD5, 160 bits pour SHA-1)
- Résistance à la pré-image : étant donné h, il est difficile de trouver x tel que $h = H(x)$
- Résistance à la seconde pré-image : étant donné $h = H(x)$, il est difficile de trouver $y \neq x$ tel que $h = H(y)$
- Résistance aux collisions : il est difficile de trouver $x \neq y$ tels que $H(x) = H(y)$

- $H : \{0,1\}^* \times \{0,1\}^k \mapsto \{0,1\}^n$ avec n et k fixes
- Semblable à une fonction de hachage mais nécessite une clé pour pouvoir être calculée

- Algorithmes de chiffrement par blocs travaillent sur des blocs de données de taille fixe (128 bits pour AES)
- Si taille du message différente de celle d'un bloc, utilisation d'un mode d'opération
- Exemples :
  - ECB : *Electronic CodeBook*
  - CBC : *Cipher-Block Chaining*

Exemple de l'ECB

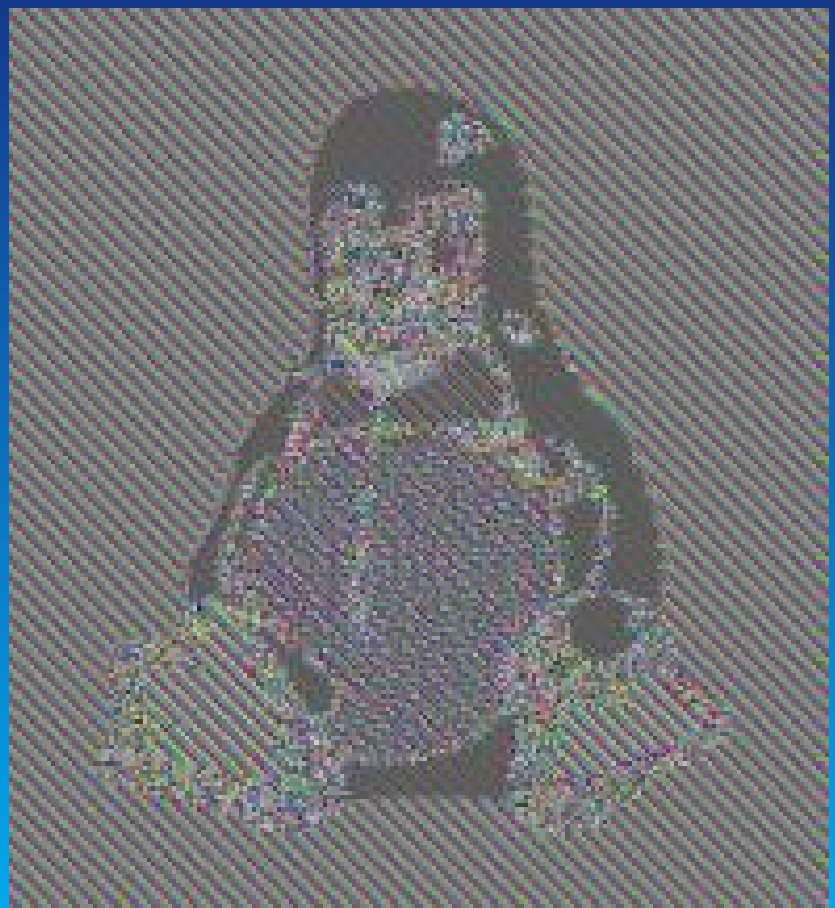


Mode ECB

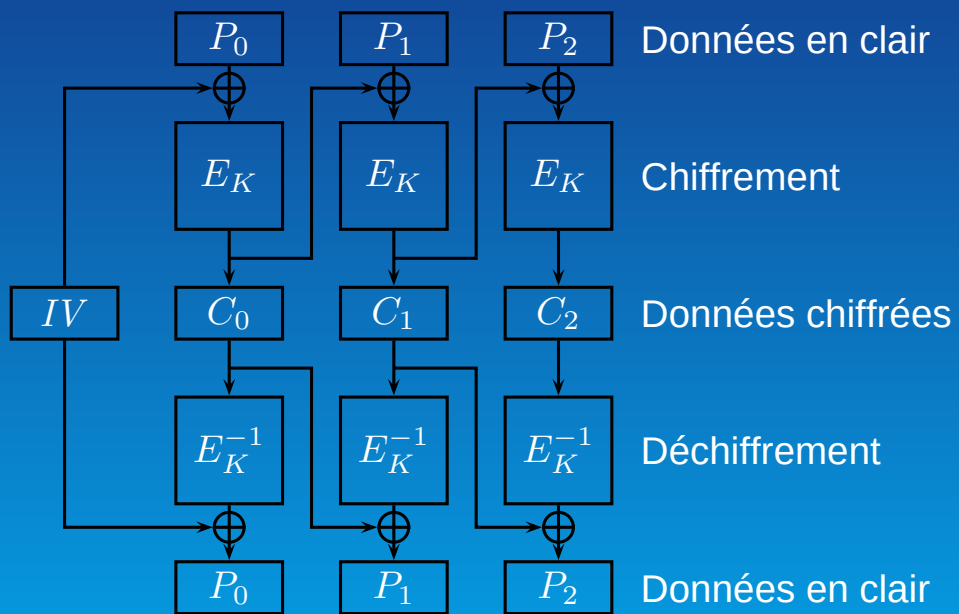
Simple & parallélisable

Problème avec l'ECB en image. . .

Non chiffré



Chiffré en mode ECB

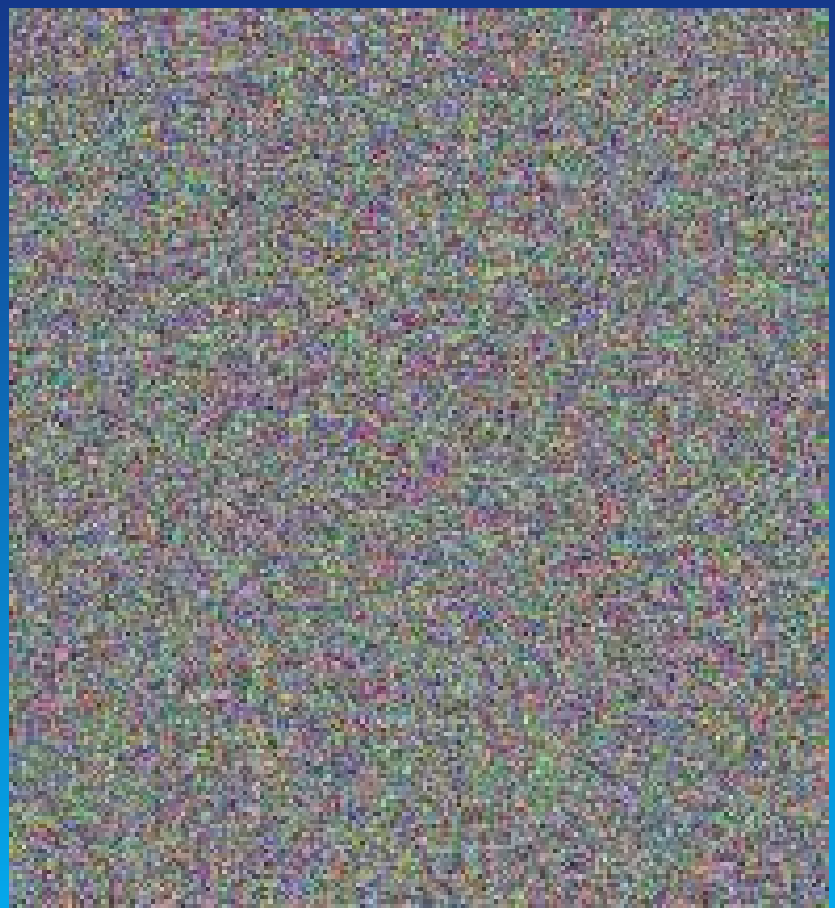


Mode CBC

À peine plus compliqué mais non parallélisable au chiffrement ☹, mais parallélisable au déchiffrement ☺

Exemple du CBC en image

Non chiffré



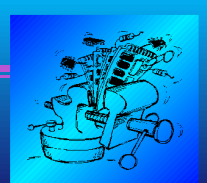
Chiffré en mode CBC

-  Injecter une valeur à une adresse mémoire ?
- Protection :
  - Authentification des instructions et données : pas possible d'injecter de fausses valeurs
  - Stocker en mémoire $D||H(D,K)$ où D est la donnée, H un MAC et K une clé secrète (non connue par un attaquant)

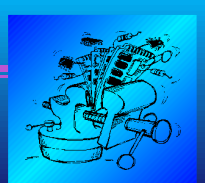
- Prendre une valeur et le MAC associé et les copier à une autre adresse en mémoire ?
-  Non détecté grâce au mécanisme précédent car le MAC est correct (même valeur)
- Protection : utiliser un MAC dépendant de l'adresse mémoire

Spécialisation du processeur : devient unique au monde

- Chiffrement données et instructions des processus sécurisés
- 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 par bloc avec clé de session
- Chiffrement dépendant de l'adresse pour résister aux attaques par substitution mais aussi pour éviter des reconnaissances de zones
- Deux clés par processus sécurisé pour éviter déchiffrement programme par erreur ou injection virale de code



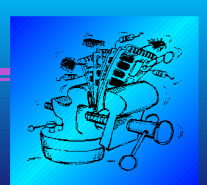
- Pour des raisons de performance garder usage du cache
- Chiffrement effectué entre le cache et la mémoire sur des lignes de cache
- Si interruption : chiffrement puis effacement du contexte matériel clair du processus sécurisé en cours



- Peut-être le problème le plus difficile à résoudre
- Rejouer la valeur prise par une variable dans le passé ou plus simplement supprimer les écritures

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

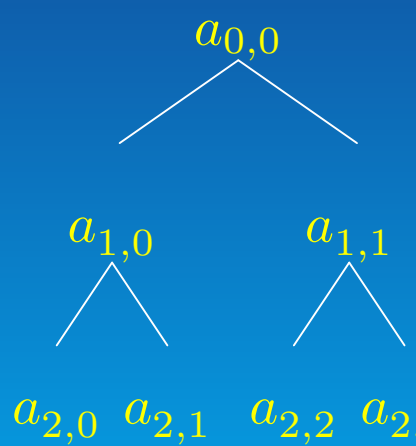
  - Si variable `i` stockée en mémoire
  - Pirate envoie des interruptions au processeur pour faire vider son cache
  - Possible de renvoyer au processeur ancienne ligne de mémoire avec ancienne valeur de `i` (simplement en bloquant le fil d'écriture)
  - `i` n'avance plus dans le programme
  - Débordement de la boucle et sortie de données ou instructions confidentielles ☹



-  Attaque : rejouer une vieille valeur mémoire
- Ajouter une horloge lors du calcul du MAC ? Mais nécessité de stocker celle-ci pour toutes les données de la mémoire (ou nécessité de mettre à jour tous les MAC à chaque écriture en mémoire. . .)
- Créer un hachage de toute la mémoire et stocker celui-ci dans une mémoire non rejouable (mémoire dans le processeur) :
  -  Utilisation d'arbres de hachage (arbres de MERKLE) pour faciliter le calcul et la mise à jour
  - Coût cependant élevé malgré des optimisations (utilisation du cache) : 20 % en moyenne (mesuré expérimentalement)

- 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 cachage

Cache

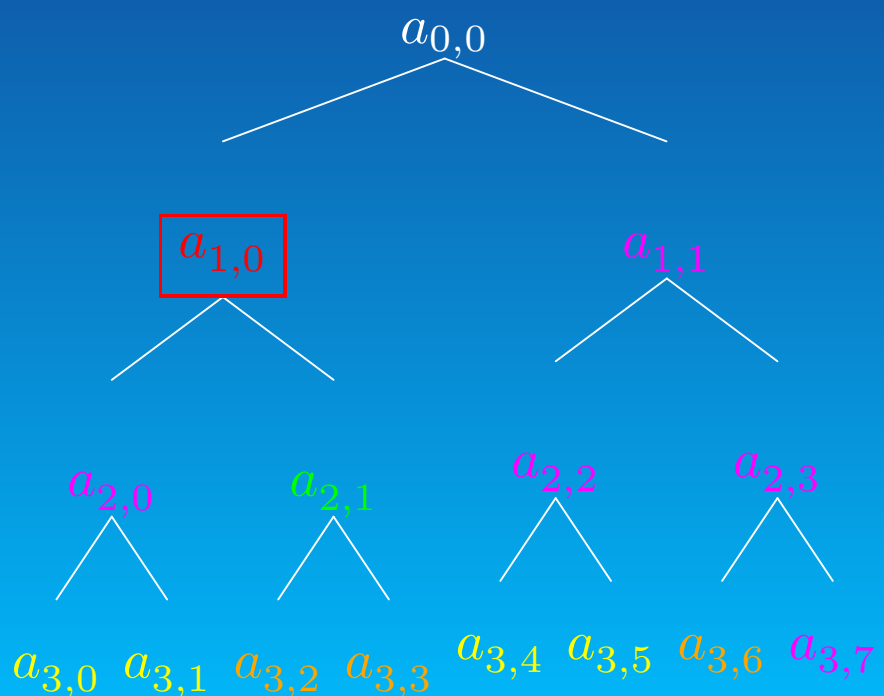
Exemple de lecture de $a_{3,3}$ truandée

En cache ( une donnée entre dans le cache *ssi* authentifiée)

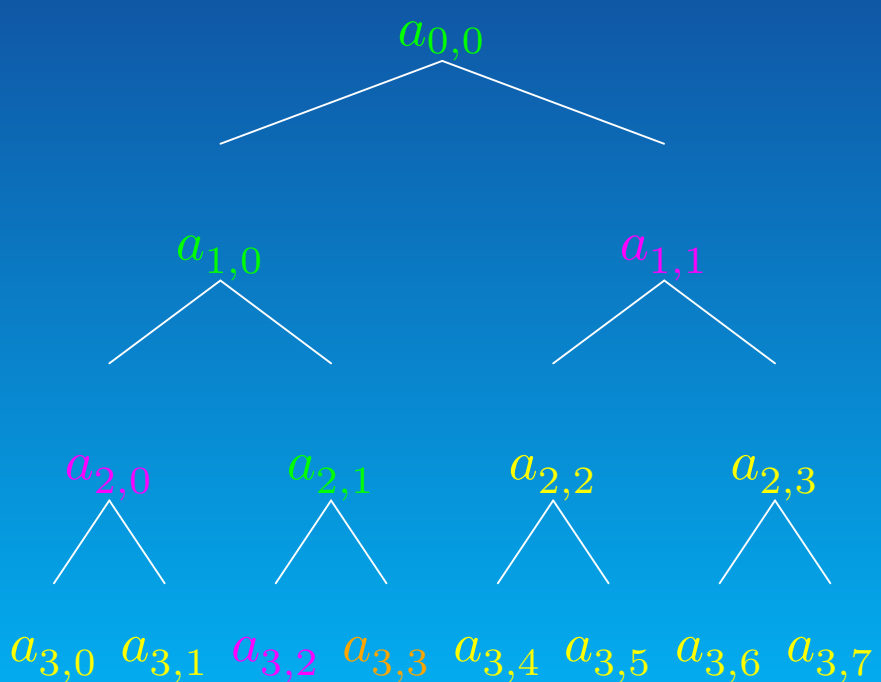
Lecture depuis la mémoire

Conflit détecté

Calcule la fonction de hachage

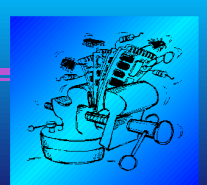


Exemple d'écriture de $a_{3,3}$



- Utilisation du chiffrement mémoire par de nombreuses architectures sécurisées (CRYPTOPAGE, XOM, AEGIS, etc.)
- Cependant, bus d'adresse non modifié (parfois chiffré)
- Possibilité de voir les motifs d'accès mémoire
- ZHUANG *et al.* : identification possible de certains algorithmes utilisés par l'application malgré le chiffrement des instructions et des données
- ⇝ Violation de la propriété de confidentialité. . .
- Infrastructure HIDE (ZHUANG *et al.*, 2004) pour minimiser ces fuites

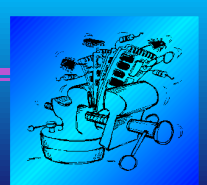
- Découpage de la mémoire en blocs (pour des questions de performance)
- Lecture d'une ligne depuis la mémoire entraîne son verrouillage dans le cache
- Cache plein : permutation des adresses à l'intérieur d'un bloc, re-chiffrement du contenu du bloc et déverrouillage des lignes appartenant au bloc
- Verrouillage : une ligne est lue depuis la mémoire au plus une fois entre chaque permutation
- Permutation : changement des adresses des lignes
- Re-chiffrement : impossibilité d'identifier une ligne avant et après la permutation
- ⇝ Impossibilité de savoir qu'une ligne est plus utilisée qu'une autre durant l'exécution du programme



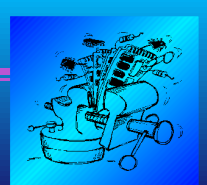
Extension de l'architecture CRYPTOPAGE pour proposer :

- Chiffrement mémoire
- Intégrité mémoire
- Protection contre les fuites sur le bus d'adresse

Le tout avec un coût très faible en terme de performances



- HIDE : Une ligne est lue et écrite en mémoire au plus une fois entre chaque permutation $\equiv$ on ne relit *jamais* une case mémoire 2 fois de suite avec même usage
- On utilise cette propriété pour faire de l'intégrité mémoire (résistance au rejeu) à faible coût en chiffrant différemment à chaque fois !
- À chaque permutation d'un bloc, on choisit un nombre aléatoire $R_{c,p}$ (confidentiel)
- Pour chaque donnée D_A en mémoire, on stocke en réalité $D_A \| H_K(D_A \| A \| R_{c,p})$ où H est un MAC et K une clé secrète
- À la relecture, on vérifie le MAC (on dispose de $R_{c,p}$)
- Si attaque par rejeu (ré-utilisation d'une donnée stockée lors d'une permutation p à une permutation p'), $R_{b,p}$ et $R_{b,p'}$ seront différents et donc MAC erroné

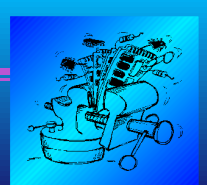


- Avantage : vérification d'un MAC très rapide (beaucoup plus que celle d'une branche d'un arbre de MERKLE)

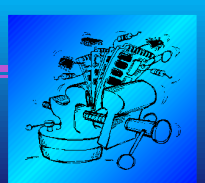
-  Il faut encore stocker les $R_{c,p}$ pour les différents blocs et les protéger eux-mémes contre le rejeu...
- Protection des $R_{c,p}$ à l'aide d'arbres de MERKLE
-  Astuce : profiter du mécanisme de TLB pour rajouter informations supplémentaires liées à la sécurité !
- Protéger seulement la structure des TLB avec arbre de MERKLE
- Avantages :
  - Arbre beaucoup plus petit en profondeur

    $\mathcal{O}(\log(\text{nombre de feuilles} = \text{nombre de pages mémoire utilisées}))$

  - Nécessité de vérifier beaucoup moins souvent (lors défaut de TLB, peu fréquents, sauf applications pathologiques)



- TLB gèrent bonne exécution des processus sécurisés mais sous contrôle du processeur *et* de l'OS
- Si gestion totale par processeur : plus compliqué et moins souple
- Peut-on déléguer gestion à OS *non sécurisé* ? D'un point de vue performance : OK car défauts de TLB moins nombreux que défauts de cache
- Idée :
  - Rajouter des instructions de manipulation sécurisée du TLB
  - Rajouter un mécanisme prouvant *a posteriori* dans un *Verification Buffer* de taille $n = \log_2(\#\text{TLB})$ hauteur de l'arbre de Merkle que tout s'est bien passé et autorisant processus à continuer



Code exécuté par un OS *non sécurisé*

LoadPageInfo(p) : { *Le processeur a besoin d'un* TLB *sécurisé d'une page mémoire* p }

{ *Cerne la branche de l'arbre à vérifier depuis le bas :* }

$d = n; q = p$

tant que $d > 0$:

$A_{d,q} =$ *getNodeAddress*(d, q) { *L'*OS *trouve l'adresse du nœud de l'arbre selon son bon plaisir* }

`LoadNode` $A_{d,q}, d, q$ { *Demande au processeur de charger un nœud dans le* VB }

si $b_{d,q}$ est déjà dans le cache :

break { *On a trouvé un nœud déjà certifié en cache : on a cerné* }

$d = d - 1; q = \lfloor \frac{q}{2} \rfloor$ { *Remonte dans l'arbre* }

$d = d + 1$ { *Redescend sur le premier nœud qui manque* }

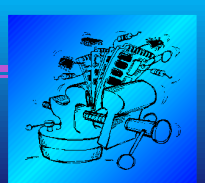
{ *Vérifie en descendant l'éventuelle branche manquante :* }

tant que $d \leq n$:

`HashCheck` d { *Demande processeur vérifier nœud dans* VB *et si correct bascule dans cache certifié* }

$d = d + 1$ { *Descend sur le nœud suivant qui manque* }

`ReturnFromInterrupt` { *Reprend l'exécution. Si attaque, le* TLB *manquera : double faute et dénis de service détecté* }



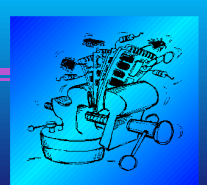
- Utilisation du mode compteur avec compteur choisi aléatoirement à chaque permutation
- À chaque permutation d'un bloc, on choisit un nombre aléatoire $R'_{c,p}$ confidentiel
- Une ligne est chiffrée de la façon suivante :

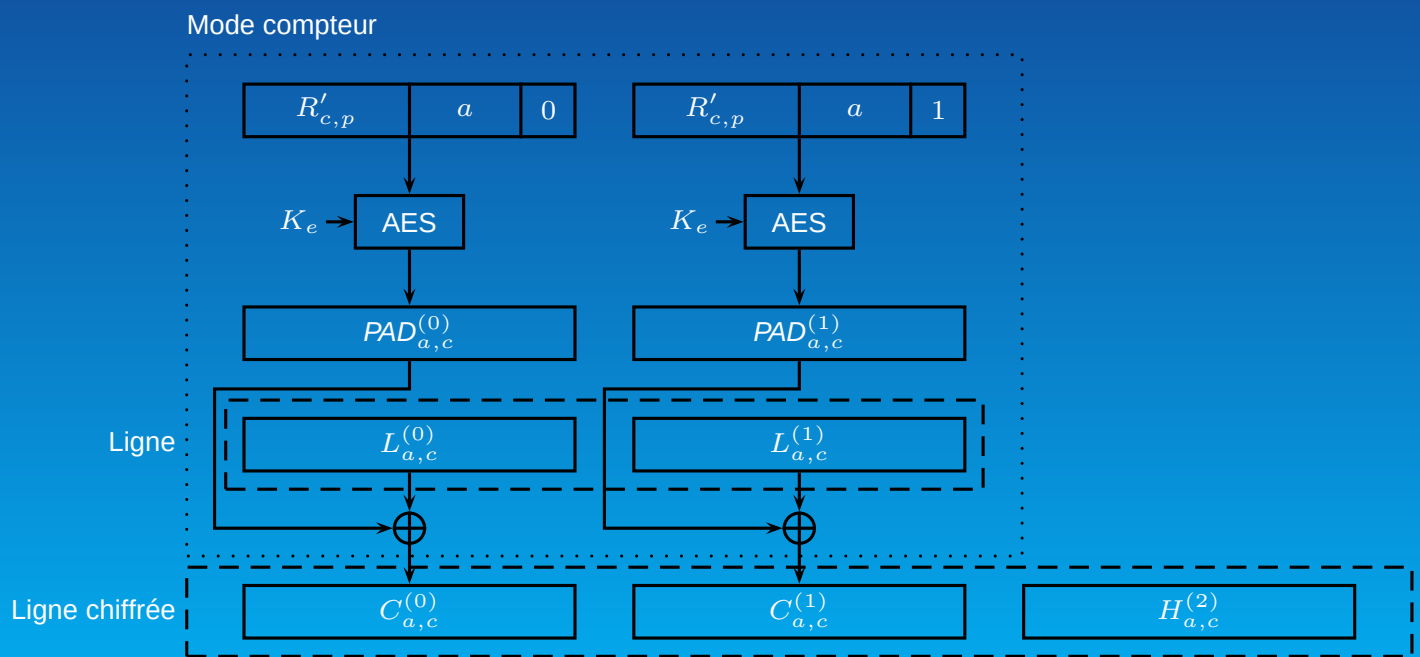
$$PAD_{a,c}^{(i)} = E_{K_e}(R'_{c,p} \| a \| i) \quad (1)$$

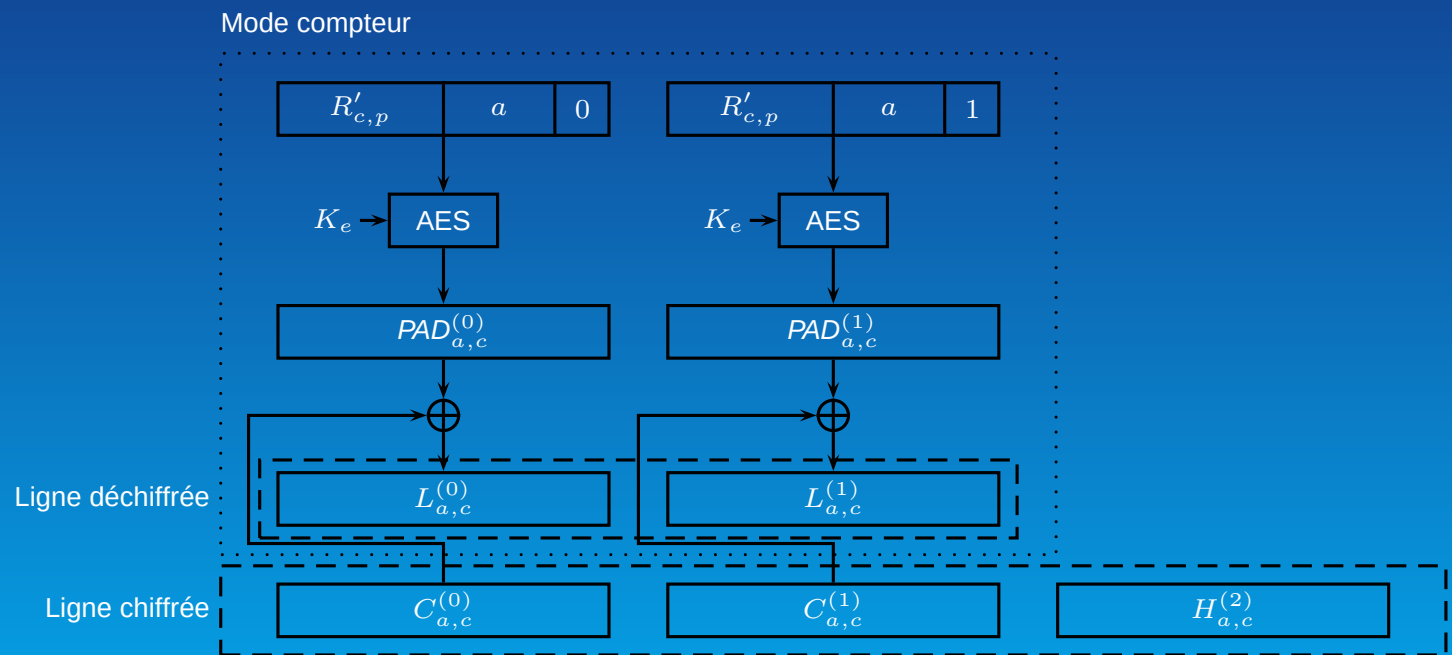
$$C_{a,c}^{(i)} = L_{a,c}^{(i)} \oplus PAD_{a,c}^{(i)} \quad (2)$$

$$C_{a,c} = C_{a,c}^{(0)} \| C_{a,c}^{(1)} \| \cdots \| C_{a,c}^{(l-1)} \quad (3)$$

- $R'_{c,p}$ est stocké de manière chiffrée et sécurisée dans le TLB étendu

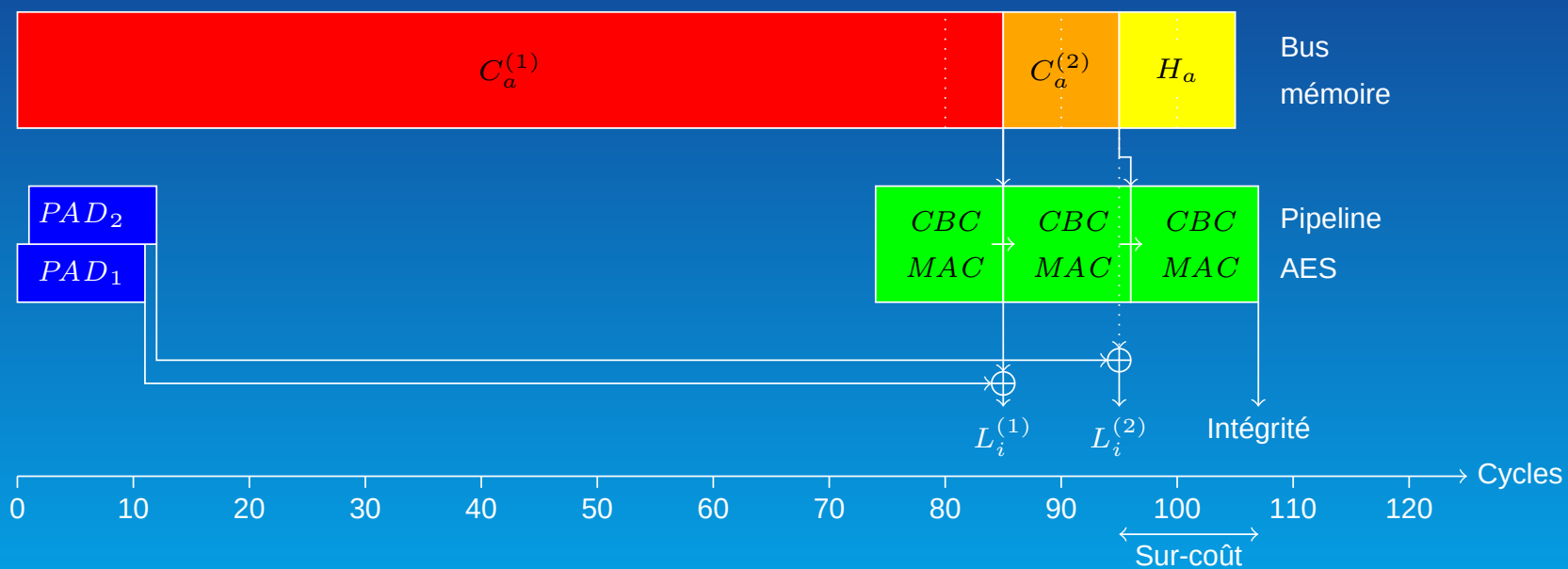



Mode compteur
$R'_{c,p}$ a 0
$R'_{c,p}$ a 1
K_e AES
K_e AES
$PAD^{(0)}_{a,c}$
$PAD^{(1)}_{a,c}$
Ligne
$L^{(0)}_{a,c}$
$L^{(1)}_{a,c}$
Ligne chiffrée
$C^{(0)}_{a,c}$
$C^{(1)}_{a,c}$
$H^{(2)}_{a,c}$

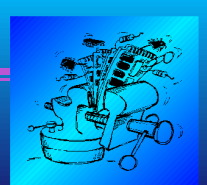


- Compteur ($R'_{c,p}||a||i$) toujours disponible pendant accès mémoire
- $\rightsquigarrow$ Information nécessaire pour déchiffrer la ligne calculable en parallèle avec accès mémoire

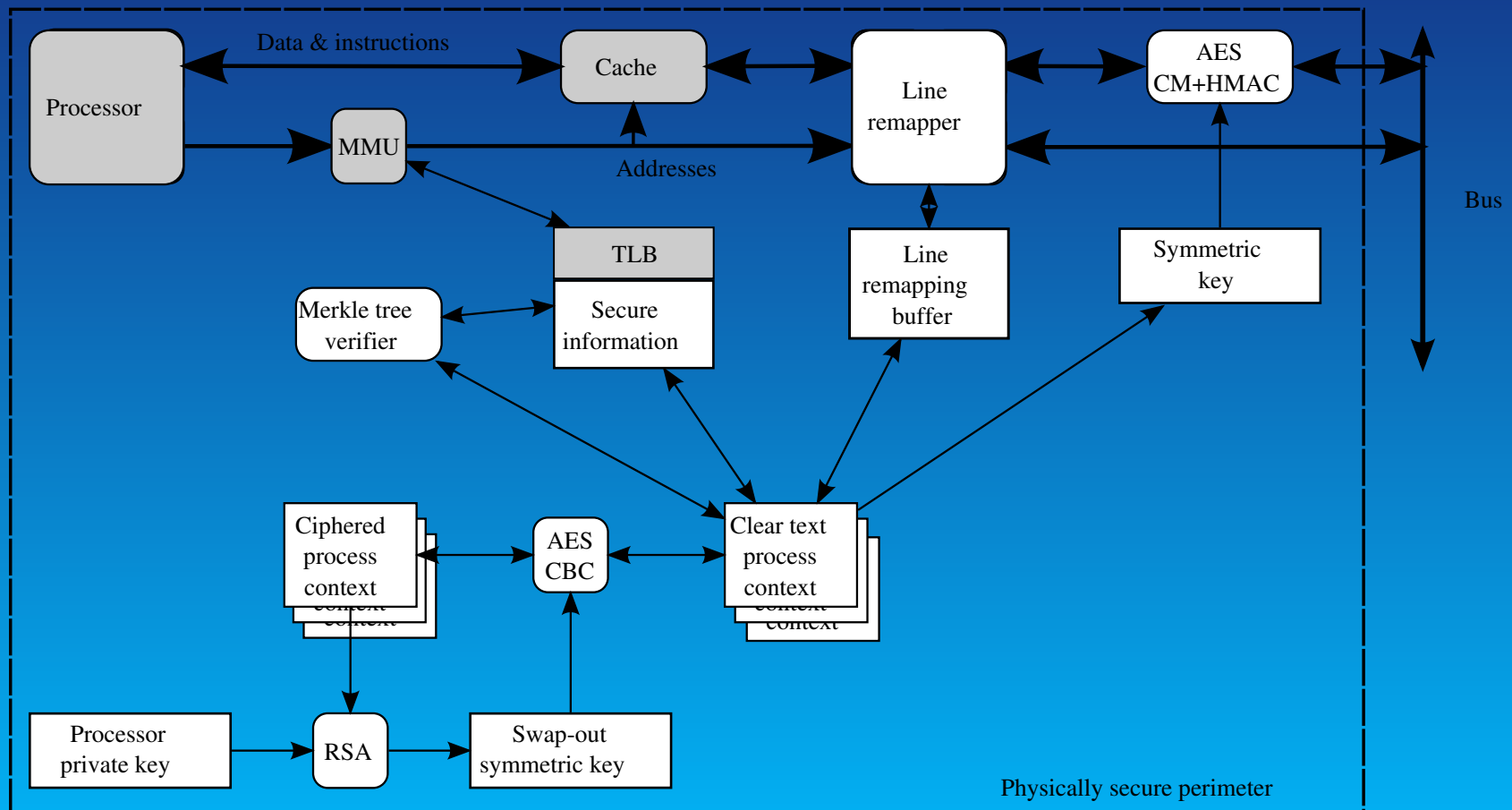
Recouvrement en lecture



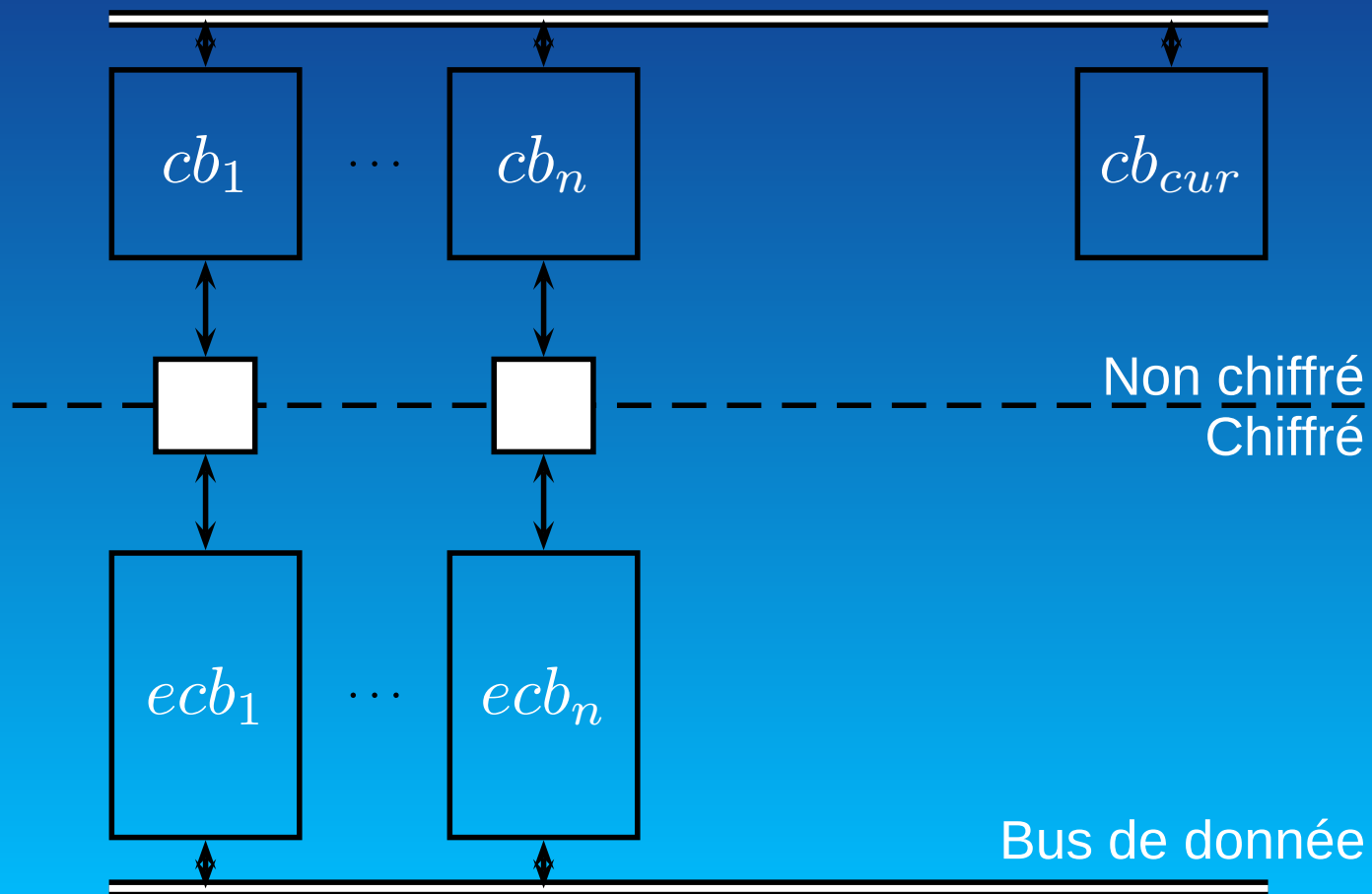
- Chaque processeur est unique avec clé publique/privée
- Rajout de chiffrement AES CBC entre cache et mémoire externe
- Chiffrement adresses de pages et mélange adresses intra-page
- Linux modifié pour gérer crypto-processus opaques



Architecture CryptoPage globale



Rajout de registres de contexte pour accélérer les changements de processus

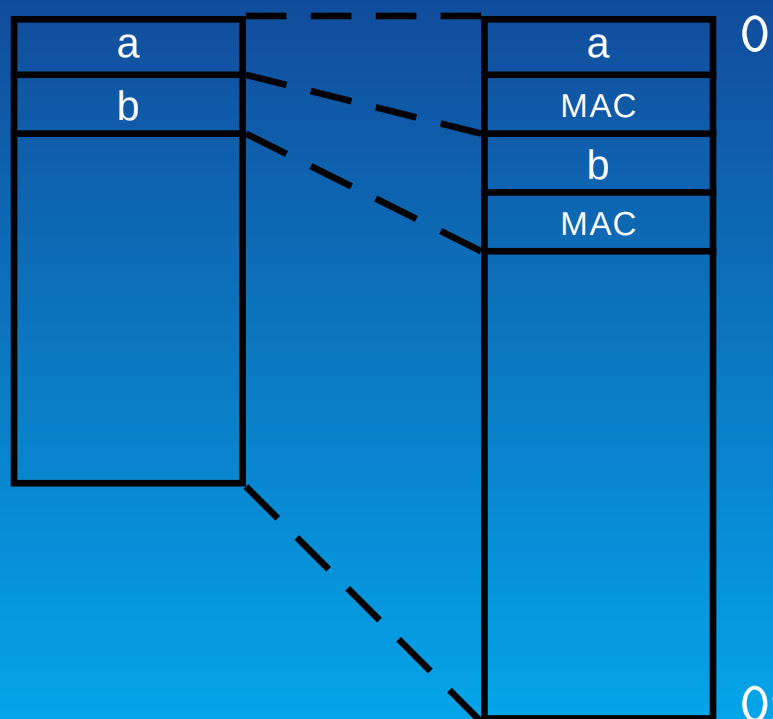


Instruction	Description
`IRETS`	Retour d'interruption en mode sécurisé
`SIG`	Déclenchement d'un signal
`SIGEND` cb_i	Fin d'un signal
`MOV` $src, ecb_{i,j}$	Charge un contexte matériel
`MOV` $ecb_{i,j}, dest$	Sauvegarde un contexte matériel
`MOV` cb_i, cb_j	Déplacement d'un contexte matériel
`MOVNE` $reg, dest$	Écriture mémoire non sécurisée
`MOVNE` src, reg	Lecture mémoire non sécurisée
`MOV` $reg, cpcr_i$	Manipulation des registres de contrôle
`MOV` $cpcr_i, reg$	Manipulation des registres de contrôle

Où stocker données d'authentification ?

- Problème : stockage transparent des MAC ?
- Modification de la MMU afin de cacher ce stockage :
  - Manipulation d'adresses *virtuelles* par processus sécurisés
  - Adresses *virtuelles* dilatées en adresses *logiques* par la MMU
  - Allocation de mémoire logique supplémentaire afin de stocker les MAC
  - Manipulation des adresses logiques par OS (sans distinction données/MAC)

Stockage des données d'authentification ?



$$A_v \rightarrow \text{MMU} \rightarrow A_l \rightarrow \text{MMU} \rightarrow A_\varphi$$

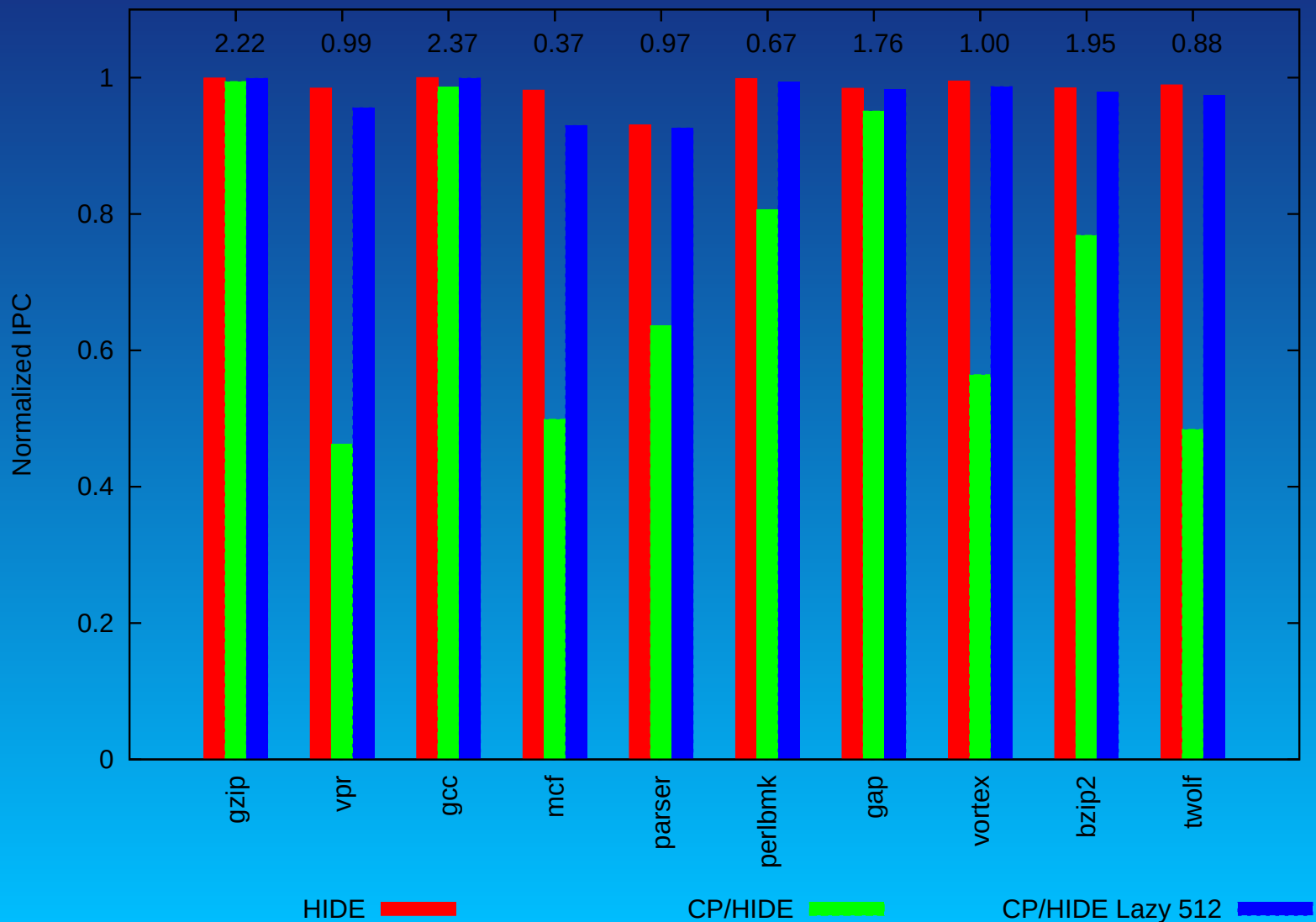
- Appels systèmes : besoin de passer/récupérer arguments ⇝ mécanisme d'effacement et de restauration sélectif des registres
- Signaux UNIX : mécanisme d'autorisation de branchement d'un processus à une adresse particulière à la demande du système d'exploitation
- Chargement initial d'un programme sécurisé : contexte initial du programme chiffré à l'aide de la clé publique du processeur

Programmes fonctionnent chiffrés dans BOCHS avec noyau LINUX 2.6.10 adapté pour tourner sur CRYPTOPAGE

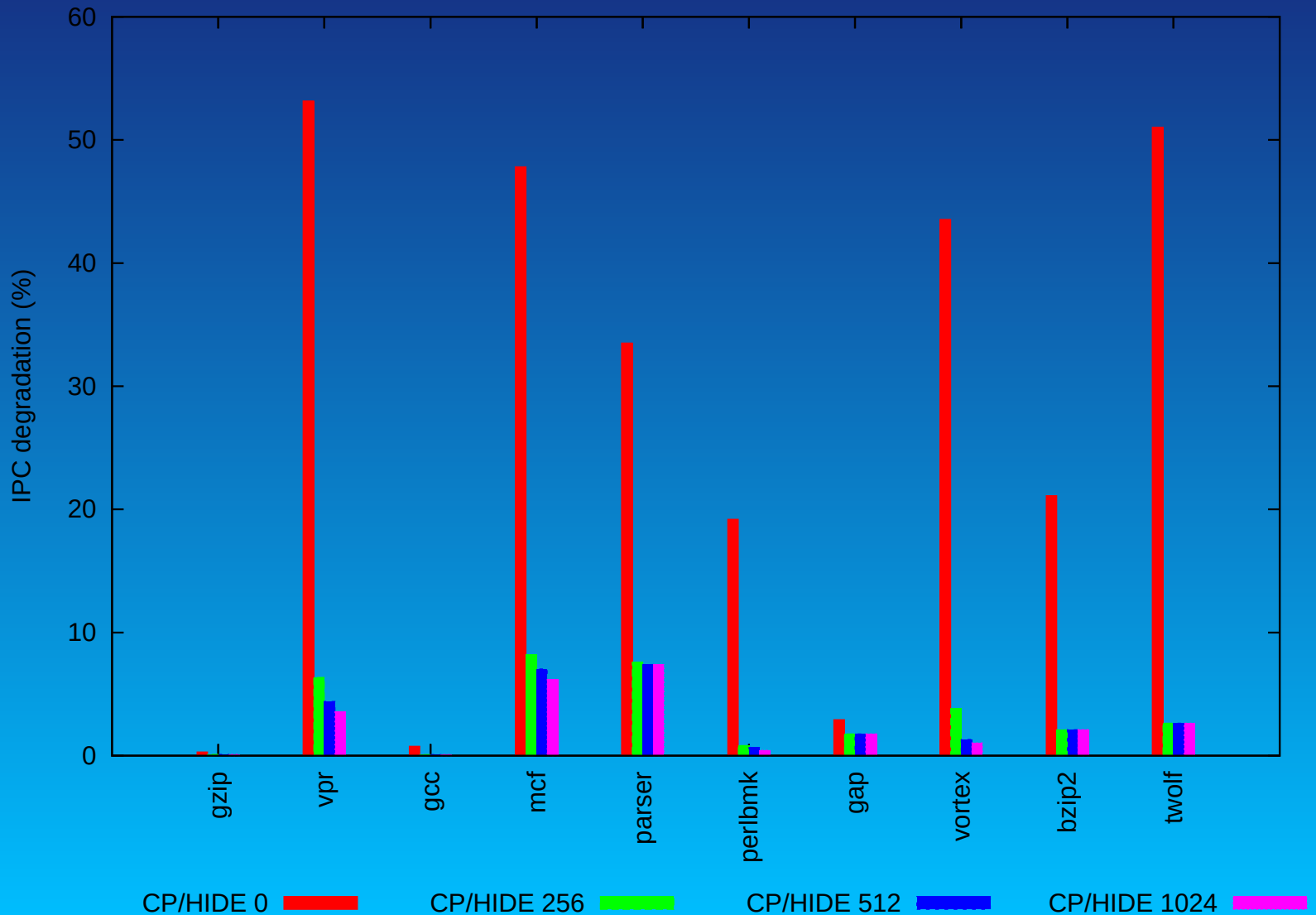
- BOCHS (100 000 lignes en tout) :
  - x86 virtuel dans simulateur PC complet Bochs : permet une simulation qualitative mais non quantitative ☹
  - + 800 lignes pour le cache
  - + 2 000 lignes pour CRYPTOPAGE
- Noyau LINUX (plusieurs millions lignes en tout) : + 500 lignes
- *dietlibc* : + 150 lignes
- Chiffrement des fichiers ELF : 1 500 lignes

- Implantation processeur style Alpha dans SimpleScalar : mesures quantitatives mais pas qualitatives (pas de problématique OS, processus...) ☹
- SIMICS ? Pas libre... ☹

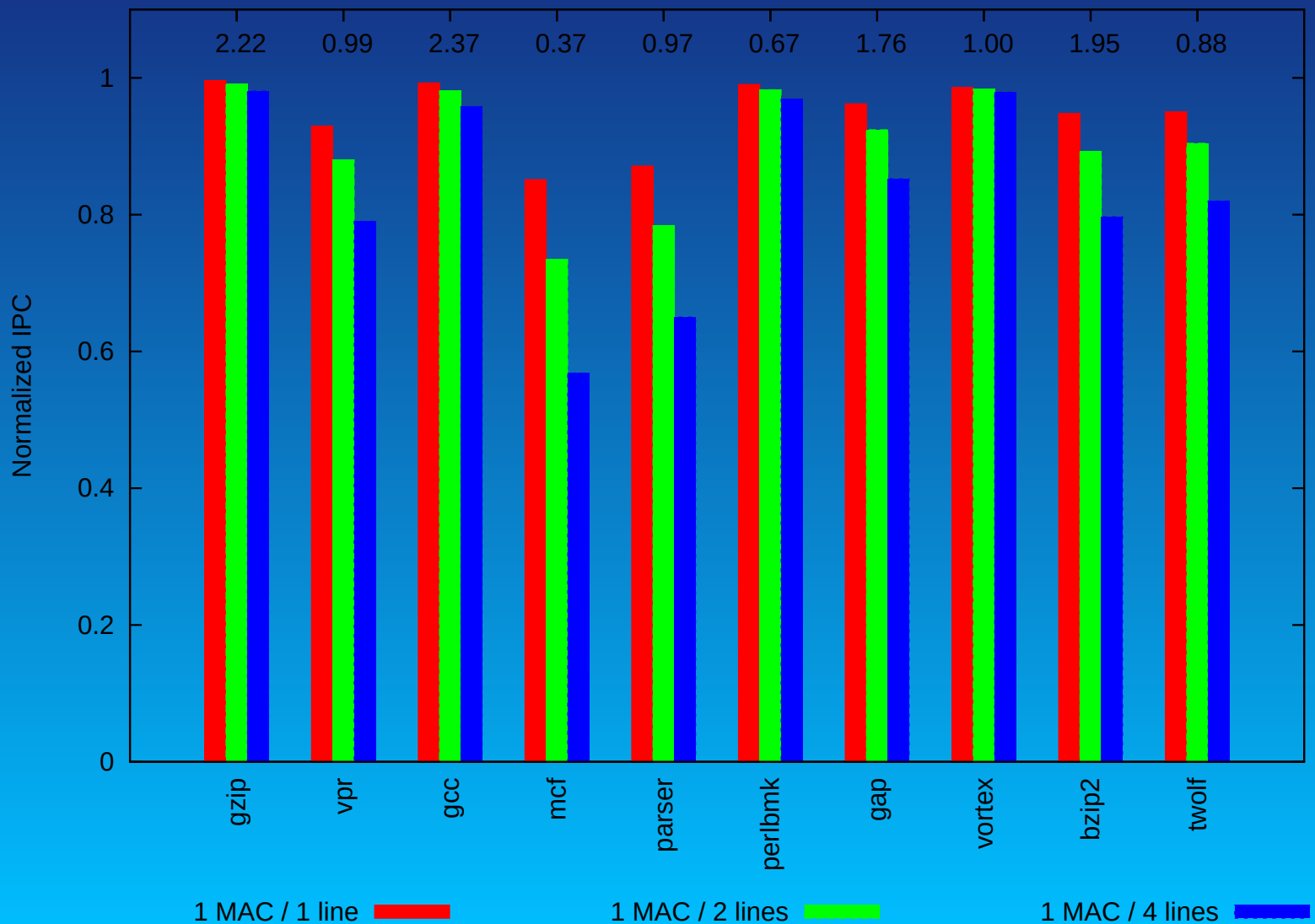
Comparaison avec HIDE



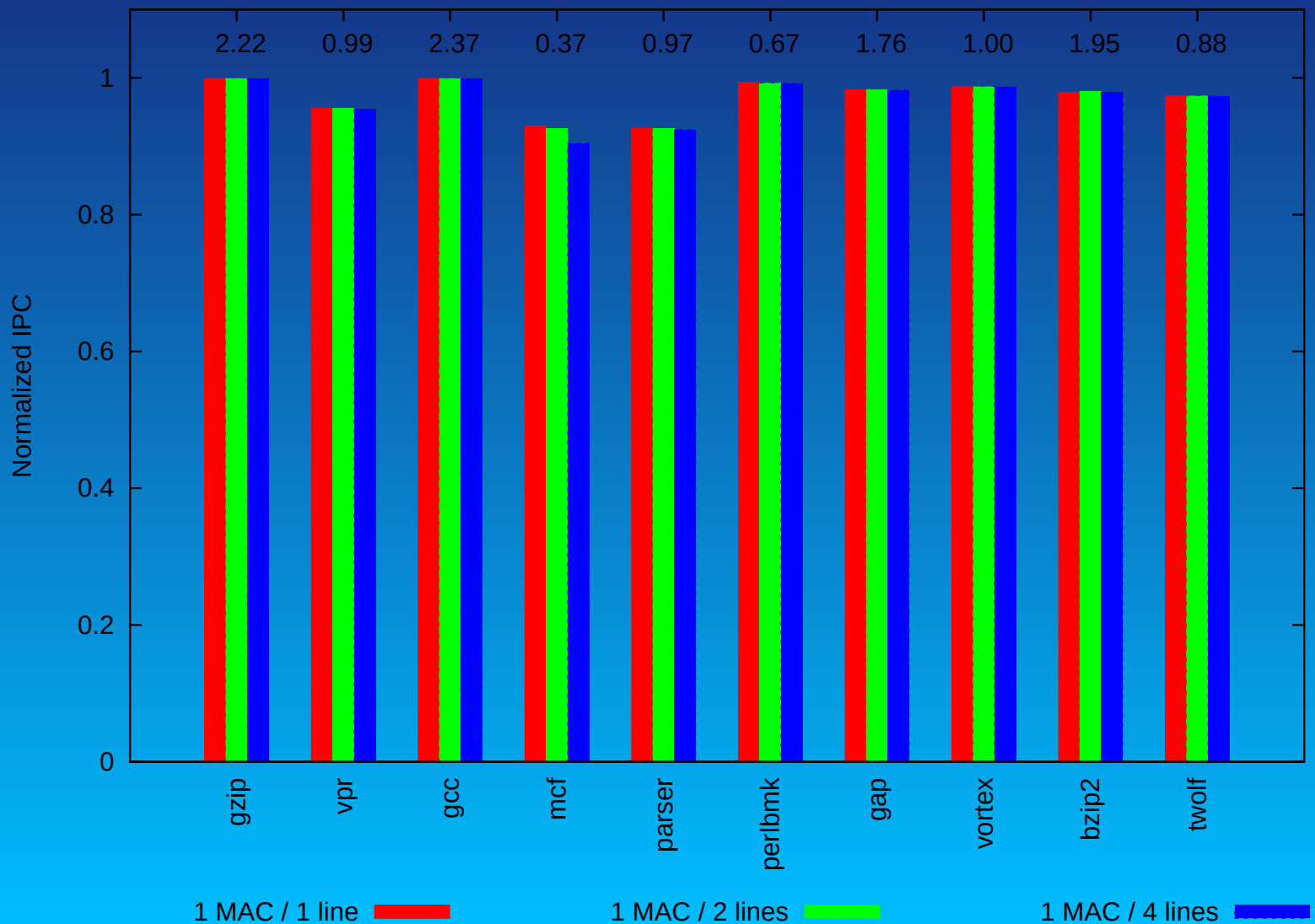
Impact de la taille du cache d'arbre de Merkle



Impact ratio lignes/MAC

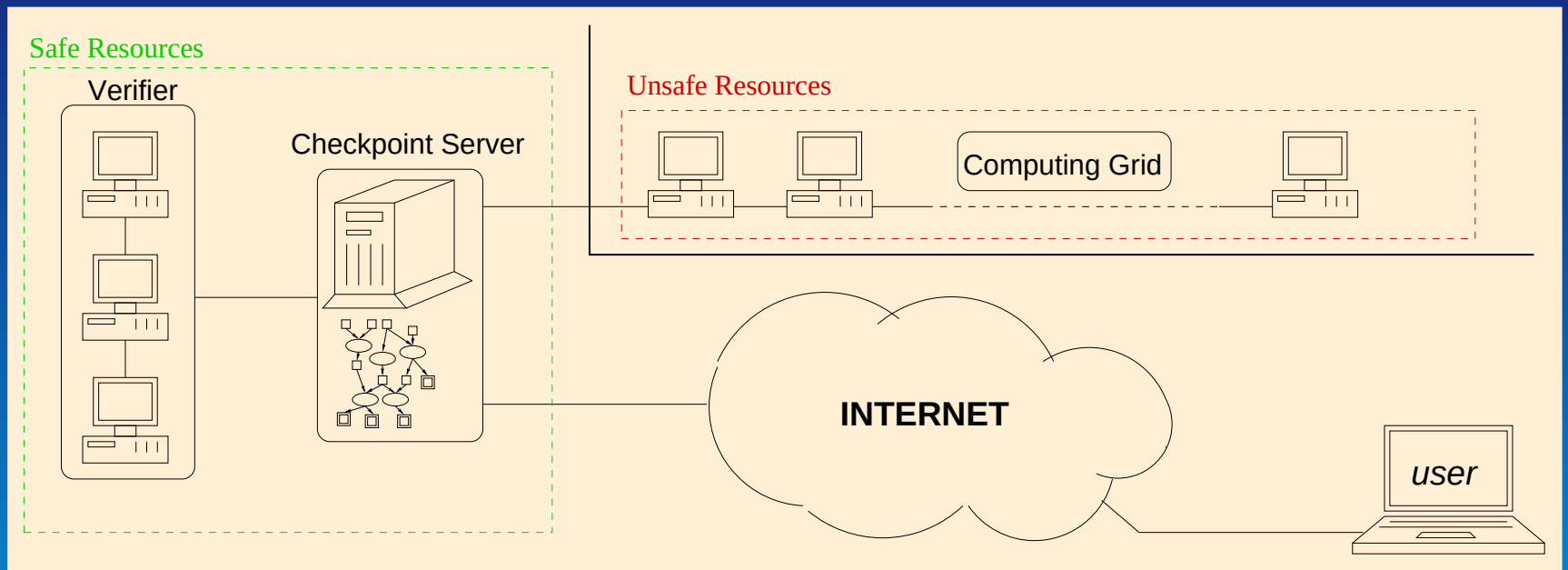


Impact ratio lignes/MAC (exécution spéculative)



- Nombreux benchmarks de `spec.org` à faire tourner
- Un essai d'architecture $\equiv$ 6 ans de calcul $\rightsquigarrow$ besoin de grosse puissance de calcul
- Utilisation de Condor (20 ans d'âge !) pour faire calcul distribué avec reprise en cas de panne
  - Permet utilisation nombreuses machines élèves
  - Mécanisme de checkpointing pour reprise en cas de panne ou reboot d'ordinateur
  - Ne fonctionne qu'en dehors des heures de cours/TP
- 60 machines style bi-cœur Pentium 4 2-3 GHz depuis 6 mois : déjà 150 000 heures de calcul...
- Effet de bord : faire profiter ENST Bretagne de cette technologie écologique

- Security And Fault-tolerance to Exploit Safety ambient Computing in lArge scaLe Environments
- ANR/ARA SSIA avec Paris XIII, Cryptalpes, ENSTB/Info, LIG/IMAG ID & LSR, IRISA
- Calculs très distribués avec nœuds de confiance (style CryptoPage) ou pas (la vraie vie)
- Applications de type graphe de tâches
  - Travailleurs
  - Serveurs de stockage d'états
  - Serveurs de vérification

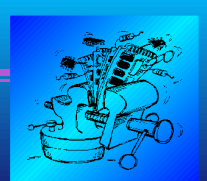


- Étudier mécanismes matériels nécessaires à exécution sûre
  - Certification exécution
  - Chiffrement code et donnée
  - Anti-rejeu...
  - Simulateur du système
  - Adaptation de Linux 2.6 à CryptoPage

- Développement de bibliothèques pour fournir services nécessaires
- Si pas ordinateur de confiance, vérification probabiliste
  - Rejouer certaines tâches sur machines de confiance et vérifier résultat
  - Minimiser rejeux en maximisant probabilité de détection de mensonge
  - Utilisation de systèmes à la CryptoPage pour éviter besoins de vérification
  - Utilisation graphe de dépendance de PIPS pour mieux placer vérification (nœuds plus dominateurs...)
  -  Quid des applications très orientées « pointeur » ? ☹
  - Utilisation parallélisation automatique ou mode interactif de PIPS pour découper en tâches. Rajouter des directives ?

- Utilisation analyse de régions de tableau dans PIPS pour générer automatiquement du code de check-pointing incrémental : autre projet...
- Utiliser analyse sémantique précise pour minimiser communications entre tâches

- Mise en place infrastructure tolérante aux pannes
- Cryptanalyse de l'approche
- Études d'applications intrinsèquement tolérantes aux mensonges ou pannes à un certain taux (arithmétiques sur corps finis...)



- Le grand retour des machines virtuelles à la IBM 370 (utilisées pour la multiprogrammation à l'origine)
- AMD V (Pacifica), Intel VMX (Virtual Machine eXtensions), Power LPAR (Logical Partitions)...
  - Plusieurs systèmes d'exploitations exécutés de manière transparente dans une vraie fausse machine
  - Pas besoin de modifier OS (contrairement à Xen...)
  - Gros centres de calculs & hébergement virtualisé, tolérance aux pannes, SOA (Service Oriented Architecture)
  - Permet de virtualiser matériel et SoC pour recycler logiciel (VirtualLogix et téléphones portables ou autres), vieux code DSP...

- Difficile de prouver de vraies applications et vraies OS
- ⇝ Segmenter applications et OS dans compartiments étanches
- Simplifier gestion des compartiments avec OS de paravirtualisation plus simple, voire prouvé
- Confiance dans les compartiments...

Travaux futurs...

- Peut-on déléguer la virtualisation de manière sécurisée à la CryptoPage
- Comment déléguer une machine virtuelle sécurisée CryptoPage à un OS de paravirtualisation qui n'est pas de confiance ?
- Permettrait d'aller avec son CryptoPage/V dans un cybercafé... modulo problèmes d'entrées-sorties
- Quels mécanismes matériels rajouter ?

Conclusion sur sécurité haute performance

- Informatique & architecture haute performance : sujet chaud
- ∃ Besoins d'informatique distribuée sûre (santé...)
- Domaine d'application qui dépasse de loin domaine informatique scientifique : systèmes embarqués, SoC...
- Domaine en explosion avec les progrès de l'intégration
  - Architectures plus efficaces (performance, coût & consommation électrique)
  - Utiliser surplus de transistors pour de la sécurité (enfin !)
  - Permet de faire exécution certifiée d'OpenSource en mode chiffré (vote électronique ou... DRM OpenSource ☺)
  - Sécurité matérielle : débarque dans le grand public (par la mauvaise porte ☹, Xbox, DRM) ⇝ applications enfin sécurisées

- Réfléchir aux bonnes et mauvaises utilisations d'un tel système, implications sociales ⇝ usages...

- Permet une exécution de confiance : protection d'exécution et des données même avec attaques physiques externes
- Ne présuppose pas d'OS de confiance
- Permet distribution de la confiance (grilles de calcul, dossier médical, postes de travail confidentiel défense...)
- Solution avec ou sans mélange des lignes de cache selon MMU ou pas
- Faible impact sur les performances en moyenne, meilleure que solutions concurrentes même sans mélange des lignes

- Concevoir une infrastructure globale sécurisée de calcul distribué : SAFESCALE
- Extensions des mécanismes proposés à des systèmes multi-processeurs et à des processeurs multi-cœurs
- Modèle de sécurité et jeu d'instructions minimal mais expressif
- Description interaction fine avec OS et faire une version de Linux adaptée à dernière version de CryptoPage
- Certification d'exécution sans chiffrement
- Gestion de la virtualisation sécurisée des machines (retour mode IBM370...)
- Rajout de mécanisme anti-rejeu absolu (applications bancaires, notaire...) via mémoire permanente à accès restreint
- Offrir nouveaux mécanismes à l'approche TCG

Sécurité matérielle plus globale

- Résistance aux attaques physique (Télécom Paris)
  - Analyses de consommation
  - Logique asynchrone
  - Routage symétrique
- OS de confiance et Bus sécurisés dans les SoC (ENST Sophia)
- Processeur de confiance (ENST Bretagne)

☕ Offre une large gamme coût/sécurité