THALES

Password cracking

Etat de l’art et avancées

14 June, 2007

THALES

IPV6 snakes on a plane

Featuring le vote électronique

14 June, 2007

Casser les mots de passe?

Kraal:a721a…

Phil:affa9…

Sid:0decf…

Cancun:7ca44…

Nono:e6967…

Jme:4f596…

Bartavelle:3dc68…

Casser les mots de passe?

Kraal:+back

Phil:*pasgros*

Sid:JamaisSansMonAX

Cancun:easyfr4g

Nono:<3patrick

Jme:faitchier

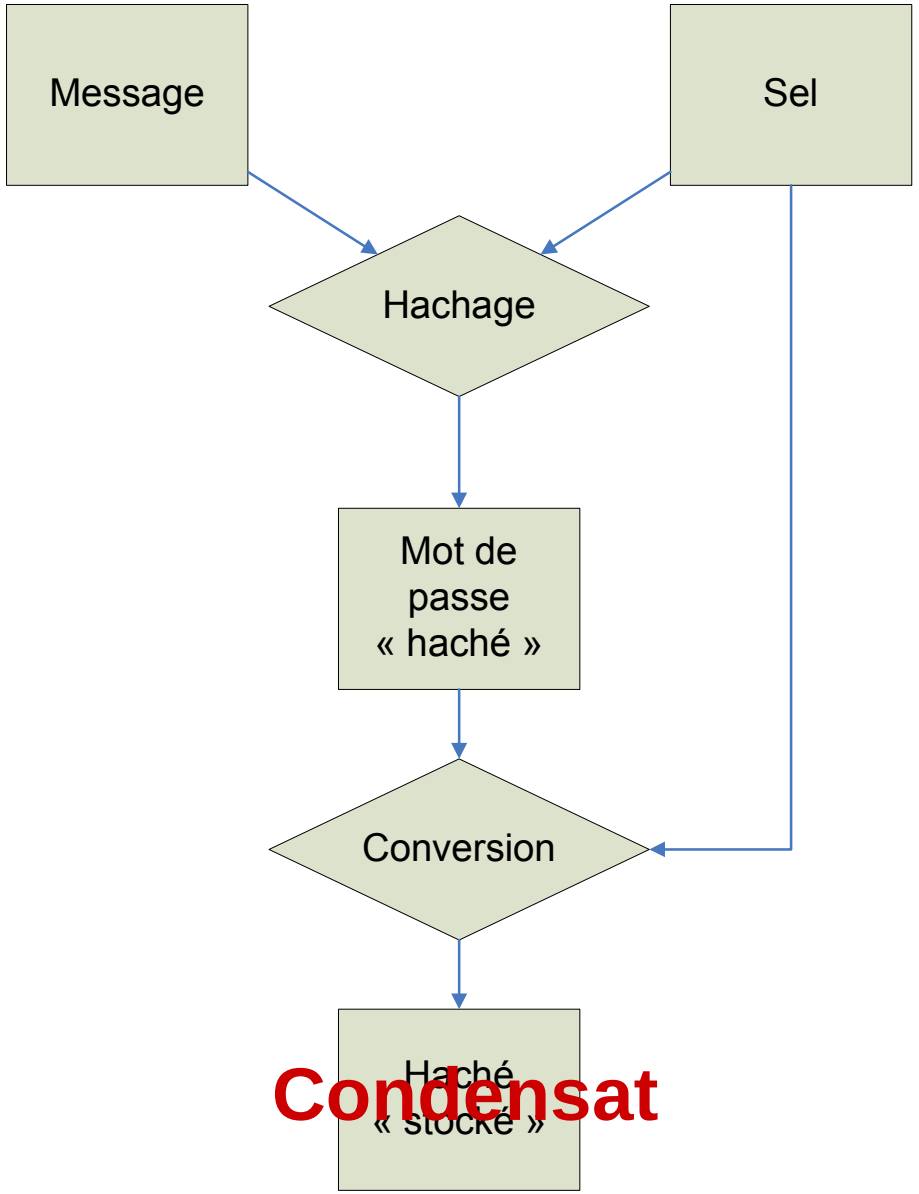
Bartavelle:troutrou

Pourquoi faire?

Pourquoi casser les mots de passe?

- Audit des mots de passe
  - Résistance à une attaque online
- Élévation des privilèges
  - Mots de passe identiques sur différents systèmes
  - Malheureusement, ypcat
- Remplir le rapport
  - Le plus beau trophée
  - Statistiques
- Le fun
  - Mots de passe ridicules

Le stockage des mots de passe



THALES

Techniques de cassage

14 June, 2007

Calcul distribué

Processus parallèle

- Casser N mots de passes à la fois sur N hôtes

- Comment distribuer le travail?
  - Générer le N^{ième} mot de passe

- Optimisations?
  - Multiples travaux de cassage
  - Le problème des graines

Rainbow Tables

Générer des une chaînes

- MDP_{n+1} = Réduction(Hachage (MDP_n))

La fonction de réduction

- Est surjective (de l'ensemble des hachages vers l'ensemble des MDP à tester)
- Consiste à générer le $N^{ième}$ mot de passe d'un ensemble
  - Changement de base
  - Markov

Dictionnaires et mutateurs

De nombreux mots de passe sont construits en faisant varier un mot connu:

- Modification de la casse
- Ajouts de chiffres / caractères spéciaux en fin de mot de passe
- Écriture phonétique

En fonction des cultures d'entreprises, une attaque par dictionnaire permet de révéler entre 30% et 80% des mots de passe (avec un bon dictionnaire).

Attaque par force brute

Consiste à tester tous les arrangements de N caractères dont la taille est inférieure à une certaine valeur

Pas très malin, mais utile dans certaines circonstance

- Il est facile d'ordonner les mots de passe, et rapide de connaître le n^{ième} mot de passe sans générer les précédents
- Permet de garantir la découverte de mots de passe courts
- Dur de faire plus rapide

```
lapik
lapil
lapim
lapin
lapio
lapip
lapiq
lapir
lapis
lapit
lapiu
lapiv
lapiw
lapix
lapiy
lapiz
lapja
lapjb
```

Méthodes « à l'arrache » : john

Utilise une méthode probabiliste fonction de la fréquence d'apparition des lettres

- Très rapide
- Il est presque impossible de générer le nième mot de passe sans calculer les précédents
- « Entraîner » le programme avec des dictionnaires de différents langages ne semble pas avoir beaucoup d'effet
- Plus efficace que la force brute

```
lappy
lappa
lappo
lappi
lappY
lappr
lapp2
lappA
lapp$
lapp3
lappE
lappb
lapin
lapie
lapit
lapip
lapid
lapig
lapi1
```

Méthodes « scientifiques » : chaînes de Markov

Utilise également une méthode probabiliste fonction de la fréquence d'apparition des lettres

- Moyennement rapide
- Il est trivial de générer le nième mot de passe sans calculer les précédents
- Ne permet de générer qu'une quantité de mots de passe finie
- N'est actuellement proposée par aucun outil public

```
lapili
lapill
lapilo
 lapil
lapima
 lapim
lapina
lapind
lapine
lapini
lapinn
lapino
lapint
 lapin
 lapio
 lapip
 lapir
lapisa
lapise
```

Les chaînes de Markov?

Pour une chaîne de degrés *n*, nous supposons que:

- La probabilité d'apparition d'une lettre dans un mot de passe est fonction des *n* précédentes lettres
- La probabilité qu'un mot de passe soit sélectionné est égal au produit des probabilités d'apparition de toutes ses lettres
- Le but du jeu est de ne sélectionner que les mots dont la probabilité d'apparition est inférieure à une certaine valeur

$$P(lapin) = P(l) \,.\, P(a|l) \,.\, P(p|a) \,.\, P(i|p) \,.\, P(n|i)$$

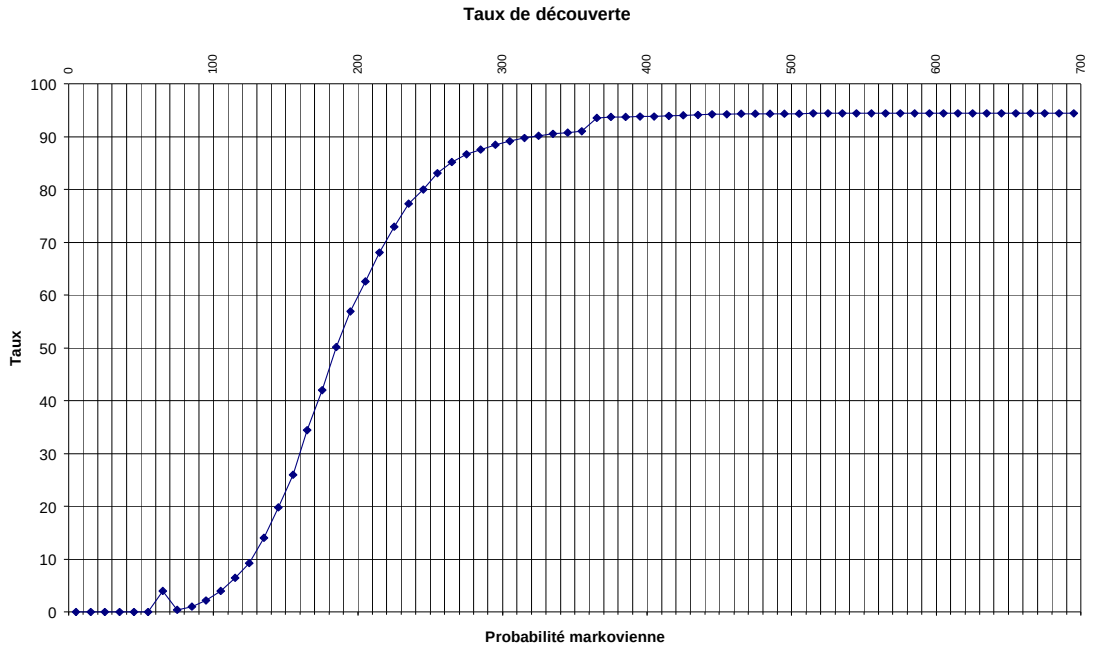
$$\ln(P(lapin)) = \ln(P(l)) + \ln(P(a|l)) + \ln(P(p|a)) + \ln(P(i|p)) + \ln(P(n|i))$$

$$-10.\ln(P(lapin)) = -10.\ln(P(l)) - 10.\ln(P(a|l)) - 10.\ln(P(p|a)) - 10.\ln(P(i|p)) - 10.\ln(P(n|i)) = P'(lapin)$$

Les chaînes de Markov?

Après avoir sélectionné un seuil maximum N, on désigne par E_N l'ensemble des mots de passe *x* tels que P'(*x*)<N

- Calculer la quantité de mots de passe contenus dans E_N
- Ordonner les éléments de E_N
- Extraire le Nième élément de E_N
- Évaluer la quantité de mots de passe à casser contenus dans E_N



Le Graal du consultant

- Il est possible d'assigner une « note » à un mot de passe
- Il est possible de faire des statistiques
- Il est possible d'avoir des indicateurs
- C'est néanmoins un bon critère de force

En pratique

14 June, 2007

Outils utilisés : classiques

- John the Ripper
  - Supporte tous les formats de stockage communs
  - De nombreux patches
    - Amélioration des performances
    - Support de nouveaux ciphers
  - Mutateurs efficaces
    - Langage spécifique (!= brainfuck)
  - Bonnes optimisations

- Bob the Butcher
  - Excellent nom
  - Fonctionne (ou pas)
  - Seulement force brute
  - Roadmap alléchante
  - Projet officieusement mort

Outils utilisés : compromis temps mémoire

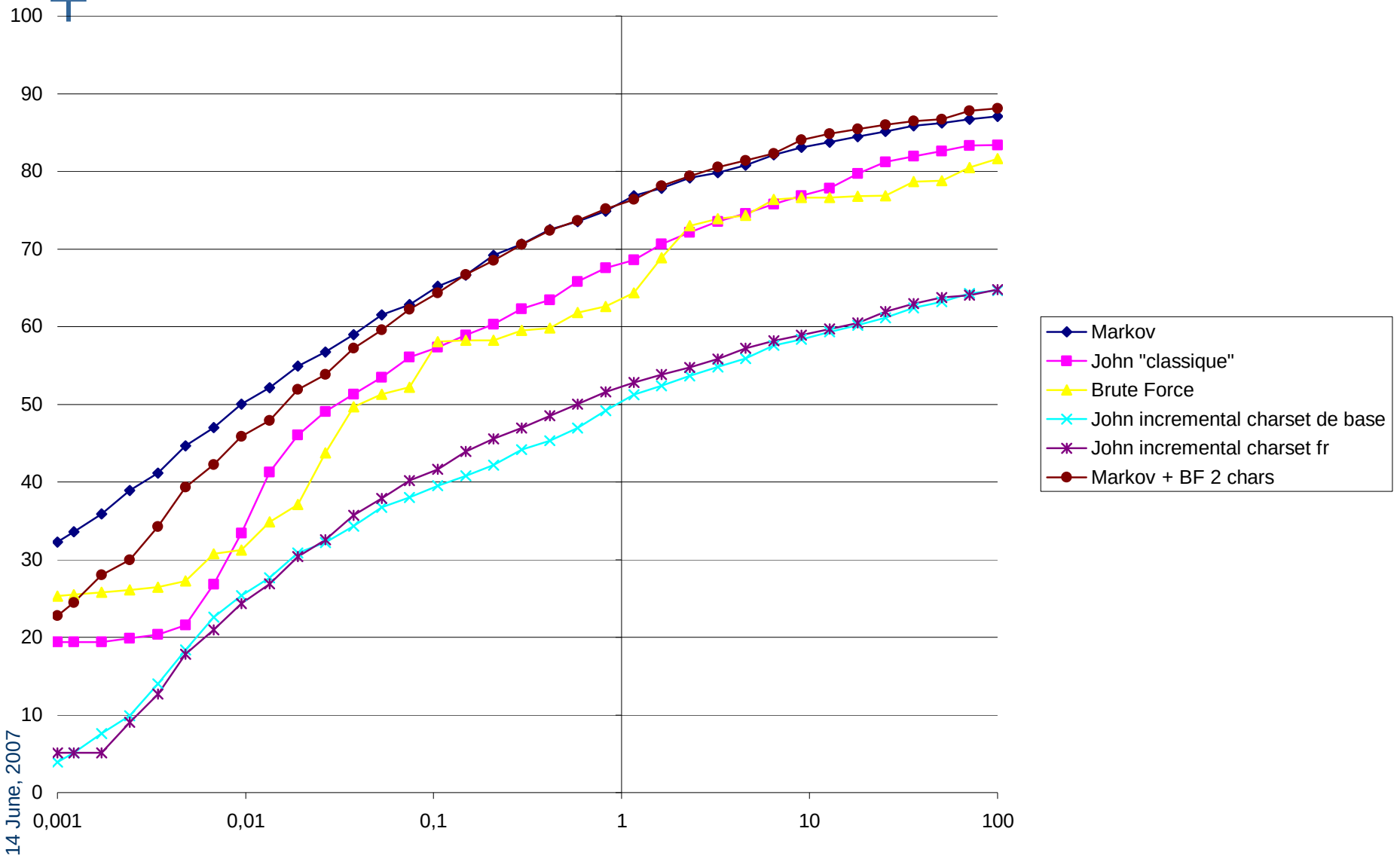
- Rainbowcrack
  - Très populaire
  - Supporte de nombreux types de hachages
  - Ne pas essayer de lire le source
  - Peu efficace
- Ophcrack
  - Efficace
  - Interface graphique
  - Pas très souple

Les biais des outils

Choix de la langue, de la disposition du clavier

- La distribution des caractères varie selon les langues
- Les caractères spéciaux ne sont pas toujours (jamais) supportés
- Les choix de charset ne sont pas toujours heureux
  - RainbowCrack selectionne les caractères spéciaux suivants dans alphanum+14: !@#$%^&*()-_+= et le caractère espace
  - Ça ne couvre que 12.8% des occurrence de caractères spéciaux
  - On peut en couvrir 99% en selectionnant: .-!* et espace
  - Sans parler des caractères non ASCII > 127

Comparaisons des outils



THALES

THALES

Techniques d’optimisation

14 June, 2007

Premier round MD4 - description

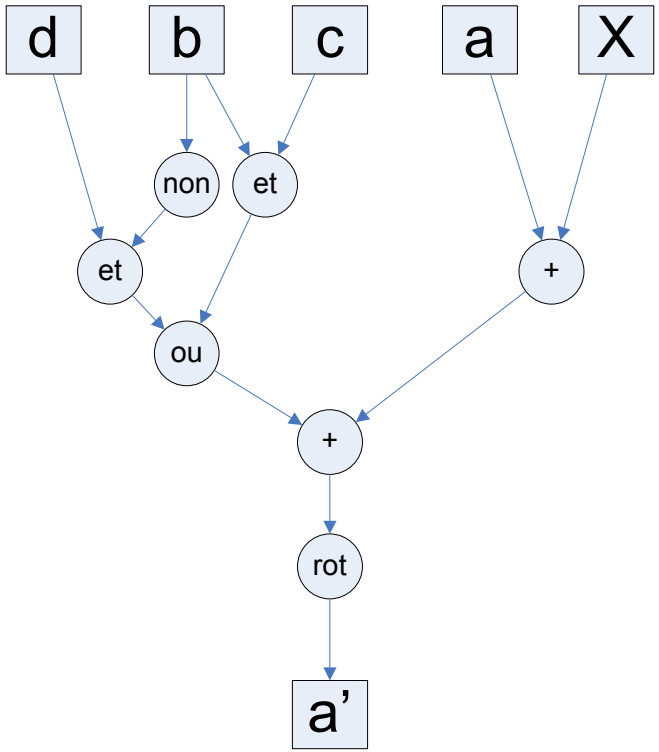
Le premier round effectue l'opération suivante:

a' = RotationGauche(a + ((b & c) | (~b & d)) + X[0], 3)

Avec

a = 0x01234567, b = 0x89abcdef, c = 0xfedcba98, d = 0x76543210

X[0] : 4 premiers octets du message à hacher



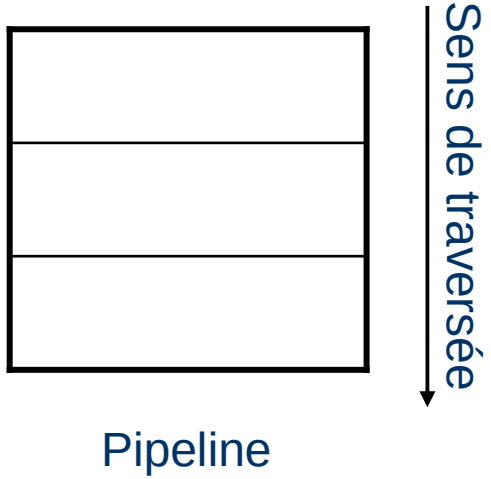

```
and ecx, ebx
not ebx
and ebx, edx
or ecx, ebx
add eax, ecx
add eax, [esi]
ror eax, 3
```

Contraintes d’optimisation

- Ordonnancement, choix des instructions
  - Dépendances entre les instructions
  - Unités d’exécution parallèles
  - Hyperthreading
- Gestion de la mémoire
  - Présence en cache
  - Niveau de cache
  - Vitesse de la mémoire

Premier round MD4 – dans le CPU

Load → and ecx, ebx
not ebx
and ebx, edx
or ecx, ebx
add eax, ecx
add eax, [esi]
ror eax, 3

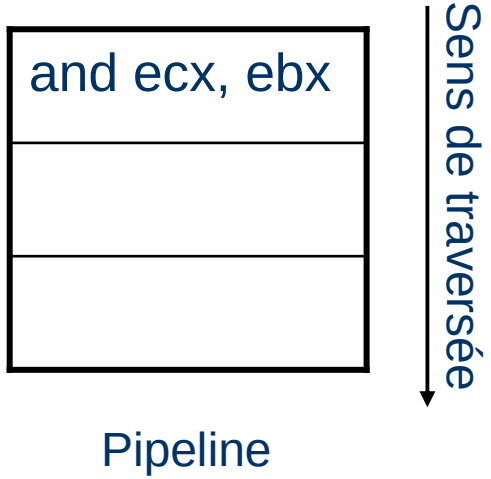


Cycle : 0

Premier round MD4 – dans le CPU

Load →

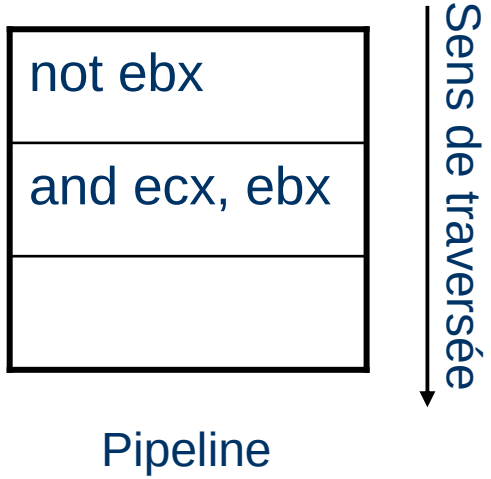
```
and ecx, ebx
not ebx
and ebx, edx
or ecx, ebx
add eax, ecx
add eax, [esi]
ror eax, 3
```



Cycle : 1

Premier round MD4 – dans le CPU

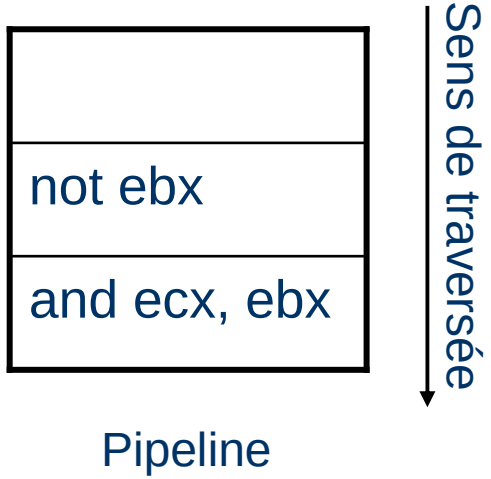
```
                and ecx, ebx
                not ebx
Load ──►        and ebx, edx
                or ecx, ebx
                add eax, ecx
                add eax, [esi]
                ror eax, 3
```



Cycle : 2

Premier round MD4 – dans le CPU

```
          and ecx, ebx
          not ebx
Load ---> and ebx, edx
          or ecx, ebx
          add eax, ecx
          add eax, [esi]
          ror eax, 3
```

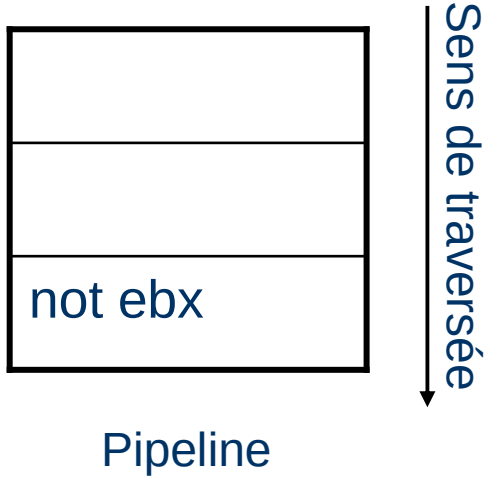


Cycle : 3

Premier round MD4 – dans le CPU

Load →

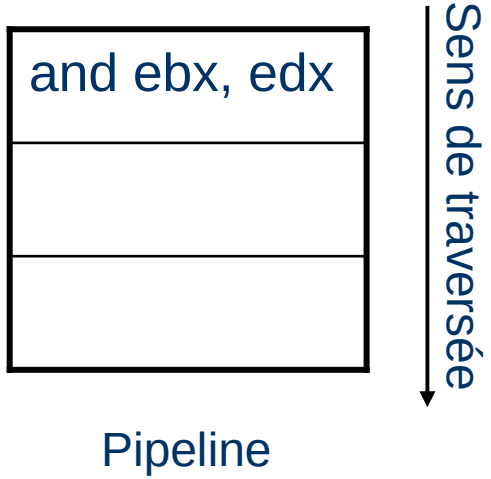
```
and ecx, ebx
not ebx
and ebx, edx
or ecx, ebx
add eax, ecx
add eax, [esi]
ror eax, 3
```



Cycle : 4

Premier round MD4 – dans le CPU

and ecx, ebx
not ebx
and ebx, edx
Load → or ecx, ebx
add eax, ecx
add eax, [esi]
ror eax, 3

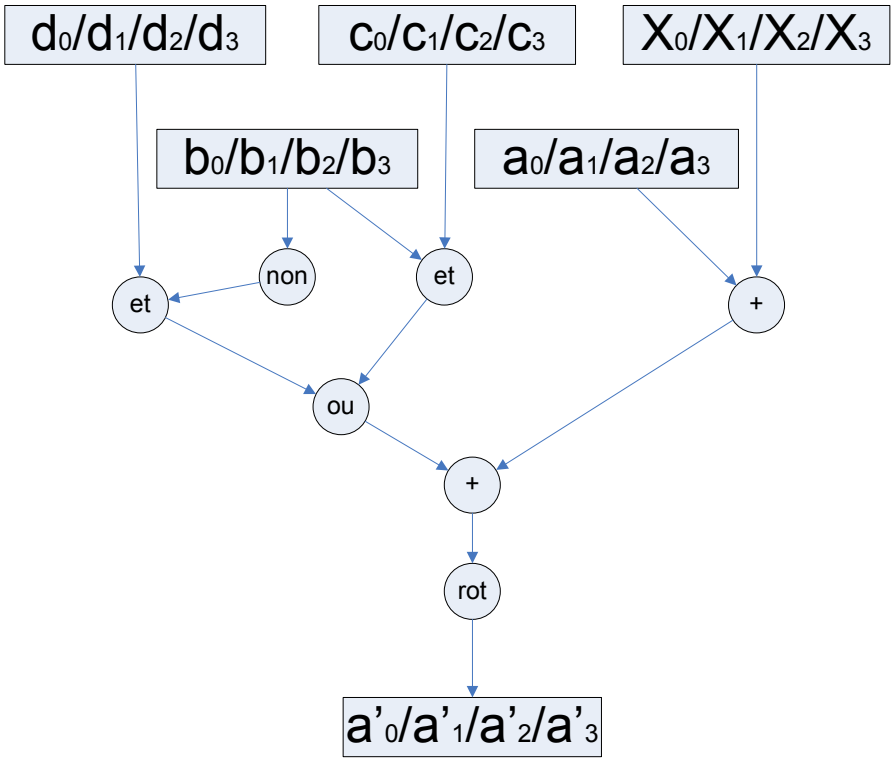


Cycle : 5

Premier round MD4 – SIMD

SIMD : Single Instruction Multiple Data
Une instruction est exécutée sur un vecteur

- MMX : 8 vecteurs de 64 bits (soit 2 entiers 32 bits)
- SSE : 8 vecteurs de 128bits (soit 4 entiers 32 bits)
- SSE 64bits : 16 vecteurs de 128bits (soit 4 entiers 32 bits)

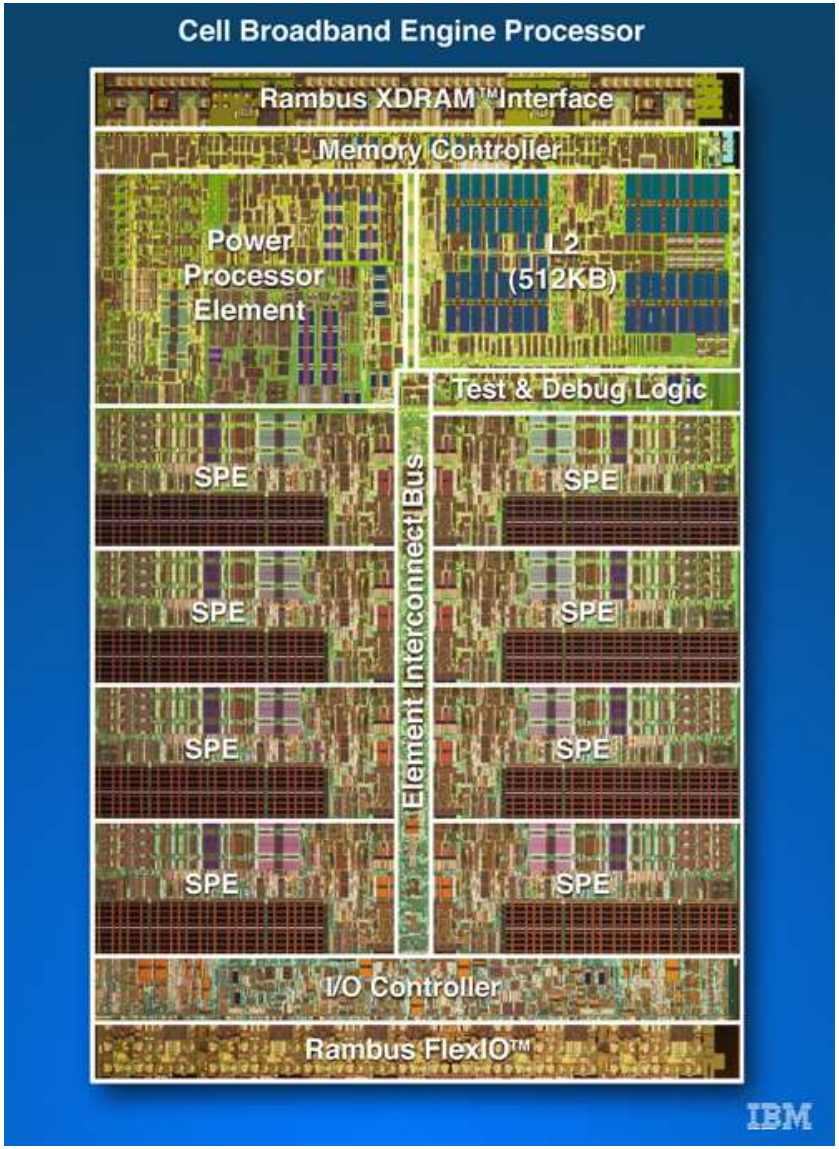


Processeur CELL

Processeur multicore cadencé à 3.2GHz
Un cœur PPC64 classique, mais lent
En théorie 8 cœurs « SPE », dont 6 utilisables sur PS3 sous linux

Spécificités des cœurs SPE:

- Instructions vectorielles
- Jeu d'instructions complet
- 128 registres de 128 bits
- 256ko de mémoire dédiée
- Pas d'accès direct à la RAM centrale



L’enfer du CELL

- Choix des algorithmes
  - La génération des MDP candidats est pénible sur les SPU
  - Casse tête des transferts de mémoire asynchrones
- Écriture du programme
  - Deux assembleurs différents dans un même fichier ELF
  - L’assembleur des SPU est **parfait**
  - Challenge de l’élimination des sauts
  - La PPU est largement plus lente que les SPU

Illustration : interdépendances

```
lqr TMP2_0, (md4_buffer+in*16*PARA+0);
lqr TMP2_1, (md4_buffer+in*16*PARA+16);
lqr TMP2_2, (md4_buffer+in*16*PARA+32);
lqr TMP2_3, (md4_buffer+in*16*PARA+48);
xor TMP1_0, tc0, td0;
xor TMP1_1, tc1, td1;
xor TMP1_2, tc2, td2;
xor TMP1_3, tc3, td3;
and TMP1_0, TMP1_0, tb0;
and TMP1_1, TMP1_1, tb1;
and TMP1_2, TMP1_2, tb2;
and TMP1_3, TMP1_3, tb3;
xor TMP1_0, TMP1_0, td0;
xor TMP1_1, TMP1_1, td1;
xor TMP1_2, TMP1_2, td2;
xor TMP1_3, TMP1_3, td3;
a TMP2_0, TMP2_0, ta0;
a TMP2_1, TMP2_1, ta1;
a TMP2_2, TMP2_2, ta2;
a TMP2_3, TMP2_3, ta3;
a TMP1_0, TMP1_0, TMP2_0;
a TMP1_1, TMP1_1, TMP2_1;
a TMP1_2, TMP1_2, TMP2_2;
a TMP1_3, TMP1_3, TMP2_3;
roti ta0, TMP1_0, rot;
roti ta1, TMP1_1, rot;
roti ta2, TMP1_2, rot;
roti ta3, TMP1_3, rot;
```

Les FPGA

- Qu'est ce que c'est?
- C'est pas mal …
  - Seule la logique nécessaire est « cablée »
  - Ça ne coûte presque rien
  - C'est amusant
- … mais pas top
  - Logique de programmation incomparable aux CPUs
  - Très pénible à débugger
  - Compétence électronique nécessaires pour exploiter pleinement

Performances des FPGA

- Moi
  - MD4
  - 7M mdp/s
  - 75$
- Copacobana
  - 120 puces
  - 4 cœurs par puce
  - 120M DES/s par cœur
  - Casse DES en moins de 14 jours
  - 10.000$

Comparaison entre tout ça

- CPU « standard »
  - Déjà présents
  - Compétence répandue
  - C'est déjà ça
- CELL
  - Console de jeux achetée par la société
  - Plaisir de programmation
  - Compétence plus rare
  - Parfait pour jouer
- FPGA
  - Meilleur rapport vitesse/prix
  - Exotique
  - Demande une compétence rare
  - Parfait pour les machines dédiées

Conclusion

Conseil de choix de mot de passe

- Stockage du mot de passe
  - Inutile d'utiliser un mot de passe très fort en LM
- Caractères spéciaux
  - Un caractère accentué rendra le cassage impossible
  - Sauf par HSC
- Ne ***jamais*** utiliser un mot de passe ridicule
  - Sautera toujours aux yeux dans les rapports d'audit
- Critères habituels
  - Longueur
  - Taille du charset
  - Est impossible à mémoriser

Attaques futures

- Meilleures méthodes de choix de mots de passe candidats
  - Markov degrés supérieur
  - Heuristiques
  - Autres
- Rainbowtables
  - Abandon de RainbowCrack
  - Fonction de réduction markovienne
- Bob the Butcher
  - Solution ultime, sur le papier

THALES

Merci de votre attention

14 June, 2007