Collisions explicites pour 76 tours de la fonction de compression de SHA-1

Pierre Karpman

Inria et École polytechnique, France
Nanyang Technological University, Singapour

Travail commun avec Thomas Peyrin & Marc Stevens

JC2, La Londe-les-Maures
2015–10–05

Fonctions de hachage

Fonction de hachage

Une fonction de hachage (binaire) est une application
$\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$

- Très utile en crypto : signatures hachées, construction de MAC, chiffres par flot
- C'est une primitive sans clef
- Comment définir une bonne fonction de hachage ?

Trois notions (informelles) de sécurité

Résistance aux (premières) préimages

Soit t, trouver m tel que $\mathcal{H}(m) = t$
Meilleure attaque générique en $\mathcal{O}(2^n)$

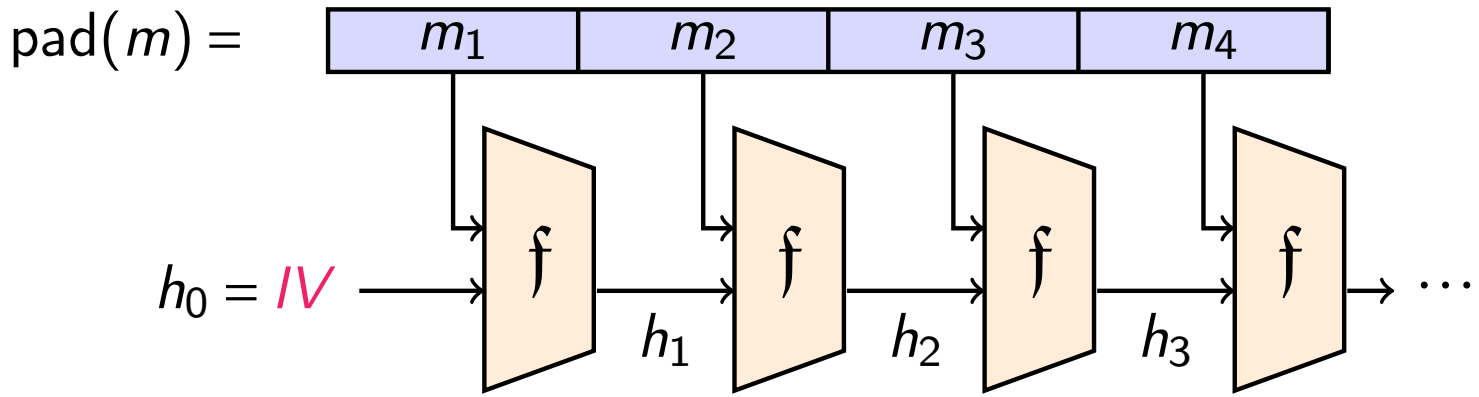
Résistance aux (secondes) préimages

Soit m, trouver $m' \neq m$ tel que $\mathcal{H}(m) = \mathcal{H}(m')$
Meilleure attaque générique en $\mathcal{O}(2^n)$

Résistance aux collisions

Trouver $m, m' \neq m$ tels que $\mathcal{H}(m) = \mathcal{H}(m')$
Meilleure attaque générique en $\mathcal{O}(2^{\frac{n}{2}})$

Fonctions de hachage à la Merkle-Damgård



- $\mathsf{A}(\mathcal{H}) \Rightarrow \mathsf{A}(\mathfrak{f})$
- $\neg\mathsf{A}(\mathfrak{f}) \Rightarrow \neg\mathsf{A}(\mathcal{H})$
- $(\mathsf{A}(\mathfrak{f}) \Rightarrow ???)$
  - Casse la réduction de sécurité

Notions de sécurité supplémentaires pour MD

Collisions à initialisation semi-libre

L'attaquant peut choisir l'IV, mais celui-ci doit être identique pour m et m'

Collisions à initialisation libre (colibri)

Aucune restrictions sur l'IV

- La même notion existe pour les attaques en préimage

Colibri (variante)

Attaquer $\mathfrak{f}$ à la place de $\mathcal{H}$

Qu'est-ce qu'on a fait ?

- Ces travaux : collisions pour 76/80 tours de la fonction de compression de SHA-1 (95% de SHA-1)
- De complexité pratique (on peut calculer l'attaque)
- Une unique carte graphique pas chère suffit pour des résultats rapides

La fonction de hachage SHA-1

- Conçue par la NSA en 1995 pour réparer SHA-0
- Membre de la famille MD4
- Hachés de 160 bits ⇒ résistance théorique aux collisions de 80 bits
- Blocs de messages de 512 bits

Fonction de compression de SHA-1

Chiffre par bloc en mode Davies-Meyer

Feistel « ARX » à 5 branches

$$A_{i+1} = A_i^{\circlearrowleft 5} + \phi_{i \div 20}(A_{i-1}, A_{i-2}^{\circlearrowleft 2}, A_{i-3}^{\circlearrowleft 2}) + A_{i-4}^{\circlearrowleft 2} + W_i + K_{i \div 20}$$

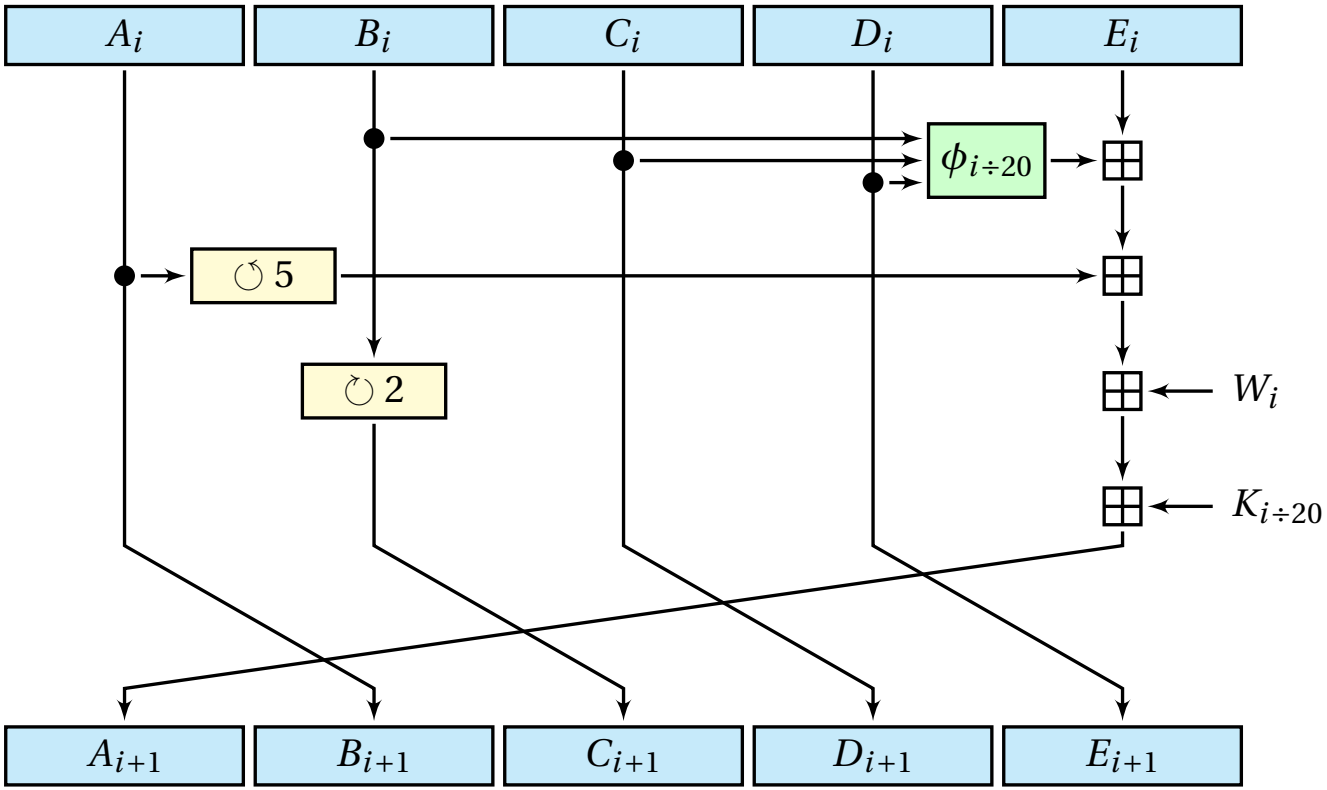
l'expansion de message est linéaire

$$W_{0\ldots15} = M_{0\ldots15},\ W_{i \geq 16} = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})^{\circlearrowleft 1}$$

$\leftsquigarrow$ La seule différence entre SHA-0 et SHA-1

80 tours au total

Fonction de tour en une image



Collisions de Wang

SHA-1 ne résiste pas aux collisions (Wang, Yin, Yu, 2005)

Attaque en collisions différentielle

- Trouver une différence de message qui induit un bon chemin différentiel *linéaire*
- Construire un chemin *non-linéaire* pour enchaîner l'*IV* avec le chemin linéaire
- Utiliser des *modifications de message* pour accélérer l'attaque
- Nécessite une paire de messages de deux blocs

Complexité de l'attaque $\equiv 2^{69}$

Améliorée jusqu'à $\equiv 2^{61}$ (Stevens, 2013)

Détour par les préimages

SHA-1 résiste bien plus aux préimages

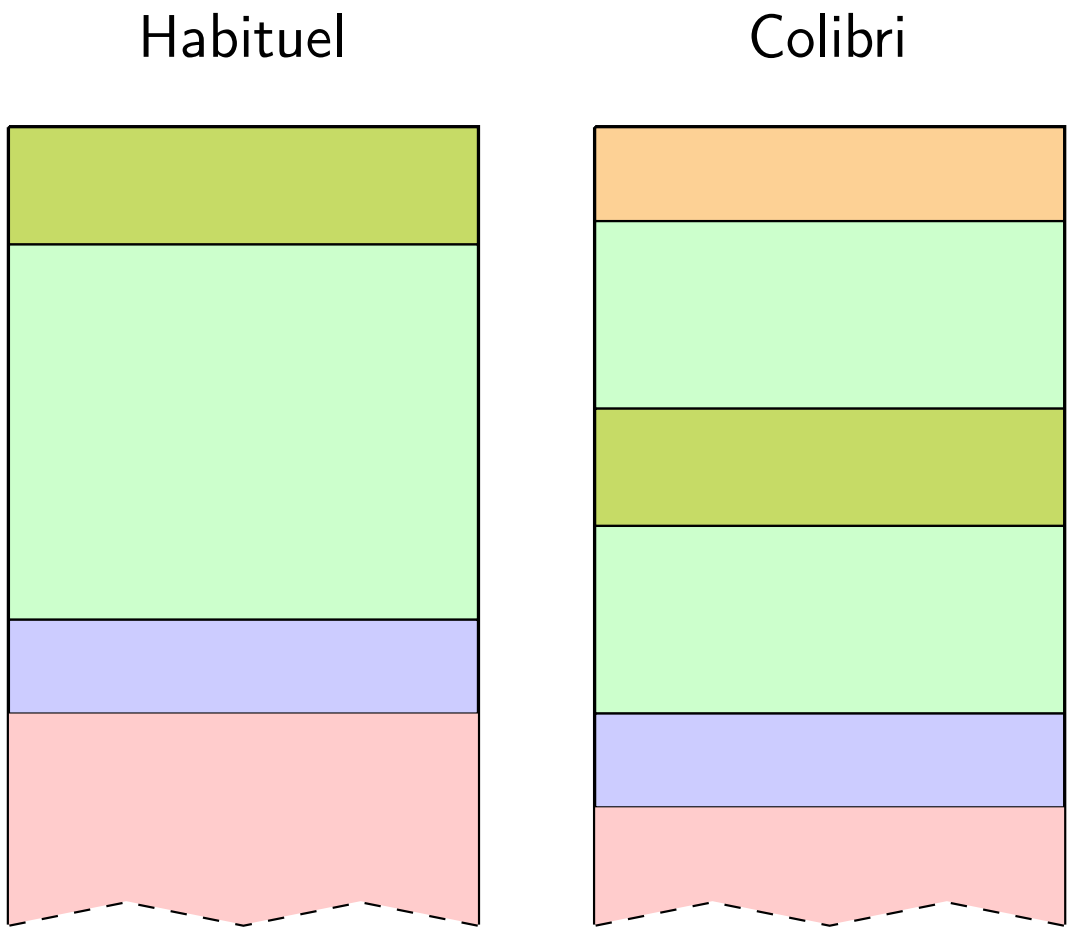
- Aucune attaque sur la fonction complète
- Attaques pratiques jusqu'à $\lessapprox$ 30 tours ($\lessapprox$ 37.5% de SHA-1)
  (De Cannière & Rechberger, 2008)
- Attaques théoriques jusqu'à 62 tours (77.5% de SHA-1)
  (Espitau, Fouque, Karpman, 2015)

Let's break stuff !

L'intérêt des colibris

- Raison principale : commencer depuis un état du "milieu" + décaler le message
- ⇒ Utiliser la liberté dans le message jusqu'à un tour plus avancé
- ⇒ Mais on ne contrôle plus l'*IV*
- ⇒ On doit contrôler la propagation arrière

L'intérêt des colibris (en image)



Qu'est-ce qu'il faut faire, alors ?

1. Trouver un bon chemin linéaire
2. Construire un bon chemin non-linéaire avec décalage
3. Trouver des techniques d'accélération

Faisons ça pour 76 tours !

- La meilleure attaque pratique est à 75 (on veut faire mieux)
- (Premier # tours avec des résultats visibles pour SHA-1 en entier)

Choix de la partie linéaire

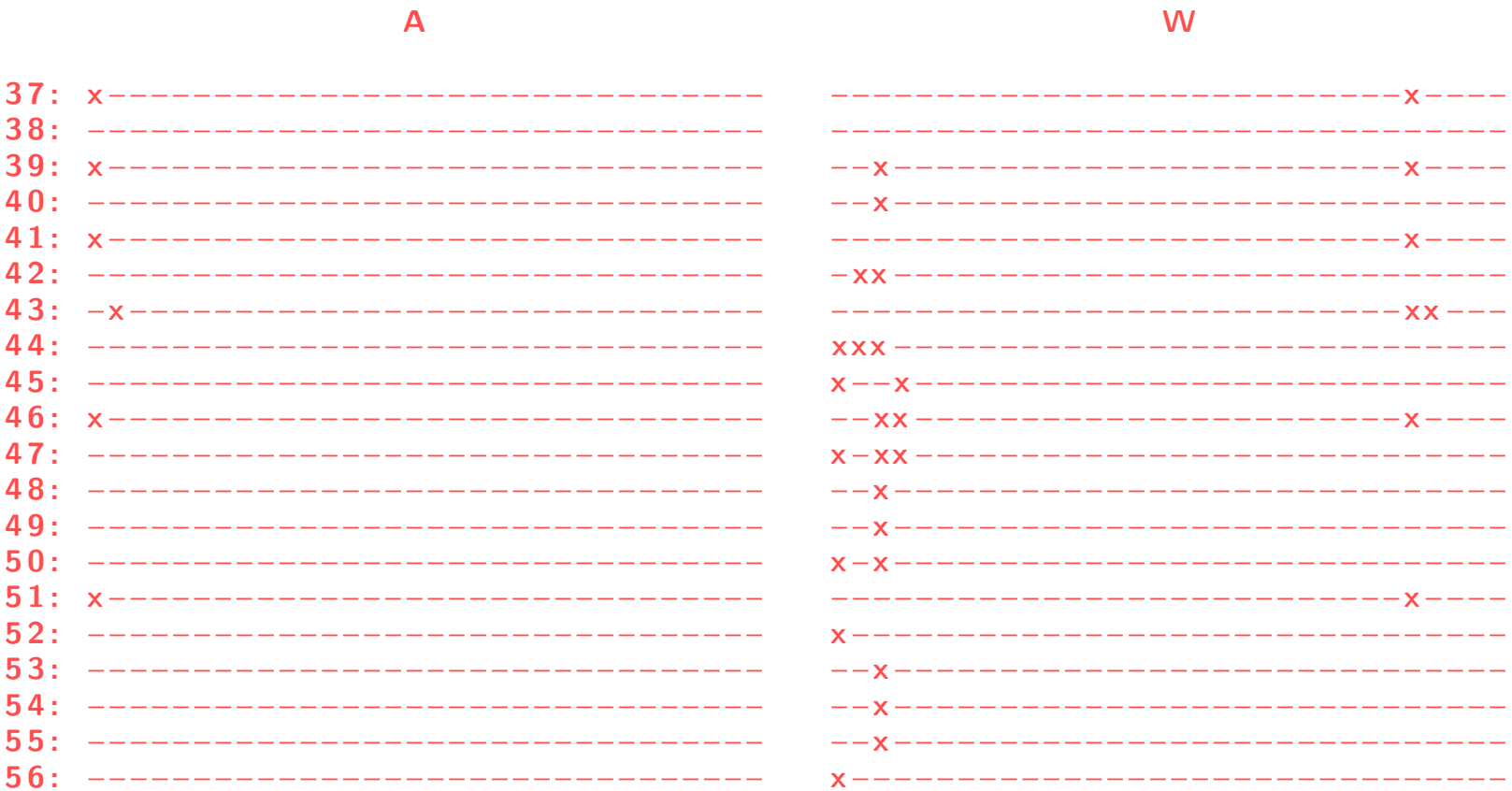
Critères :

- Probabilité élevée
- Aucune (ou peu) de différences dans les 5 derniers tours (= différences dans l'IV)
- Peu de différences dans les premiers mots de message

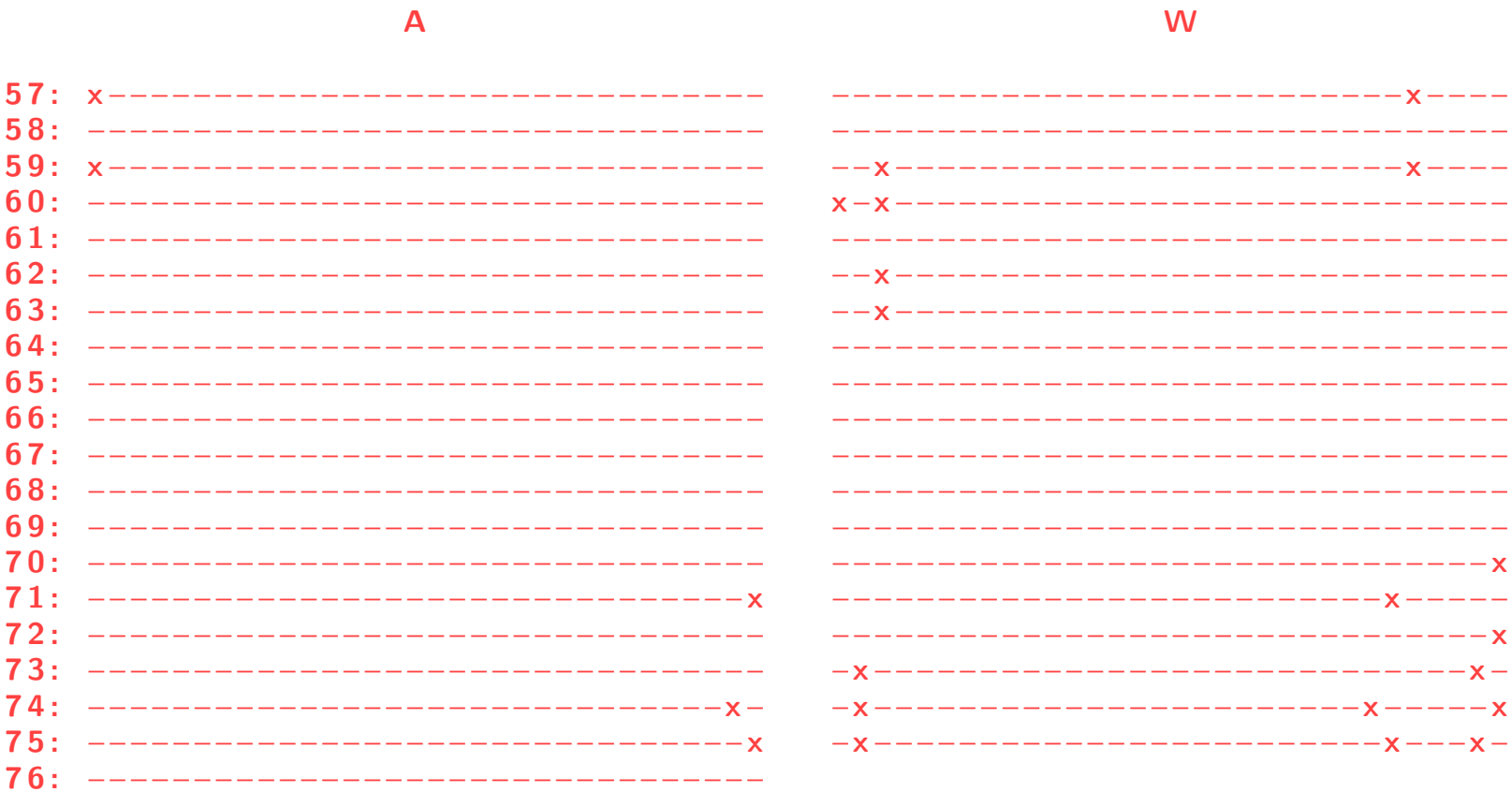
$\Rightarrow$ Pas beaucoup de candidats

On a choisi $\mathrm{II}(55,0)$ (notation de Manuel, 2011)

Chemin linéaire en image (1/2)



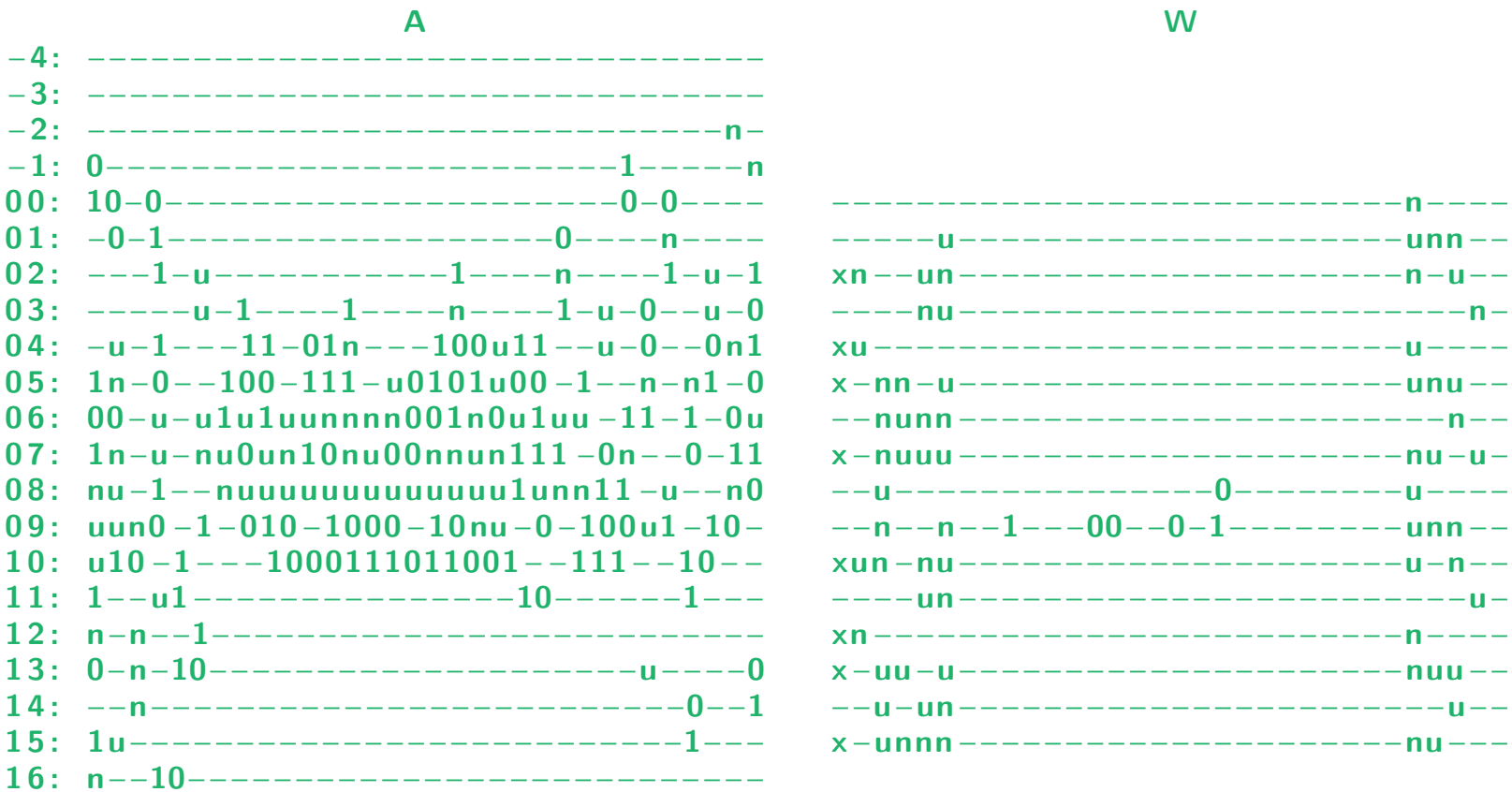
Chemin linéaire en image (2/2)



Construction du chemin non-linéaire

- Débute par un préfixe de forte probabilité arrière pour les 5 premiers tours
- Utilise une JLCA améliorée pour le reste
- ⇒ Bon chemin avec “peu” de conditions (236 jusqu’à #36)

Chemin non-linéaire en image



Techniques d'accélération

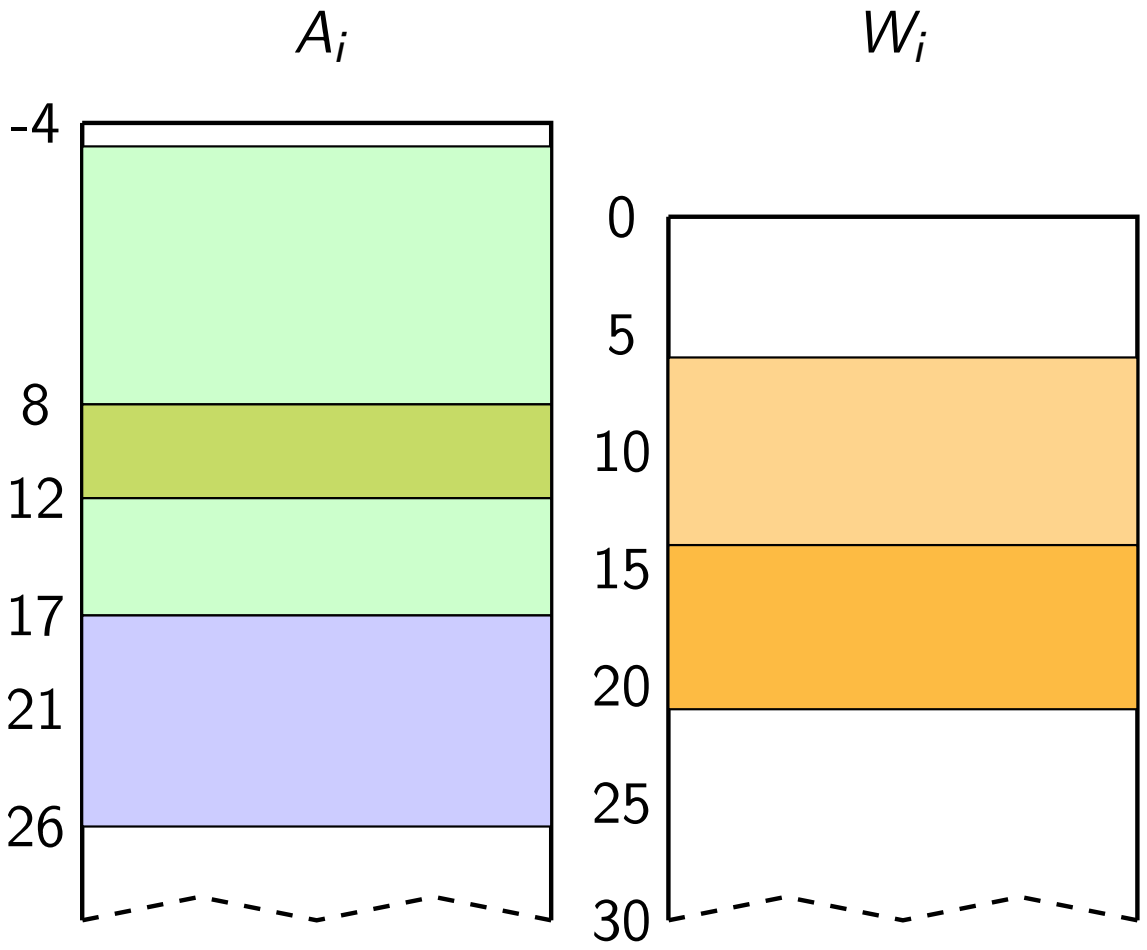
- Modification de message : corriger les mauvais instances
- Bits neutres : générer plus de bonnes instances quand une est trouvée
- On a choisi les BNs :
  - Faciles à trouver
  - Faciles à implémenter
  - Bon potentiel de parallélisation (on en reparlera)
- BNs utilisé avec décalage = 6
- Messages libres = W6...21 au lieu de W0...15

Pour résumer

- Initialiser l'état avec un décalage
- Initialiser le message avec un décalage
- Utiliser les bits neutres avec un décalage
- ⇒ beaucoup de bits neutres jusqu'à des tours tardifs (youpi)
- ⇒ on connait pas l'*IV* en avance (bof)

- Chemin linéaire ⇒ différences dans l'*IV*
- Tout est fait en un bloc
- ⇒ Attaque sur la fonction de compression

Pareil en image



Dites le avec des cartes graphiques !

- Nvidia GTX-970
- Récent, haut de gamme, bon rapport prix/performance
- $13 \times 128 = 1664$ cœurs @ $\propto 1\,\text{GHz}$
- Programmation de haut niveau avec CUDA
- Débit pour arithmétique 32 bits : tout à 1/cycle/cœur sauf ⟳
- $\approx$ S$ 500

Impératifs d'architecture

- L'exécution est groupée en warps de 32 threads
- Single Instruction Multiple Threads :
  La divergence de flot de contrôle est sérialisée ⇒ minimiser le branchement
- Cacher la latence en groupant les threads par gros blocs
- Attention à l'utilisation des registres / mémoire

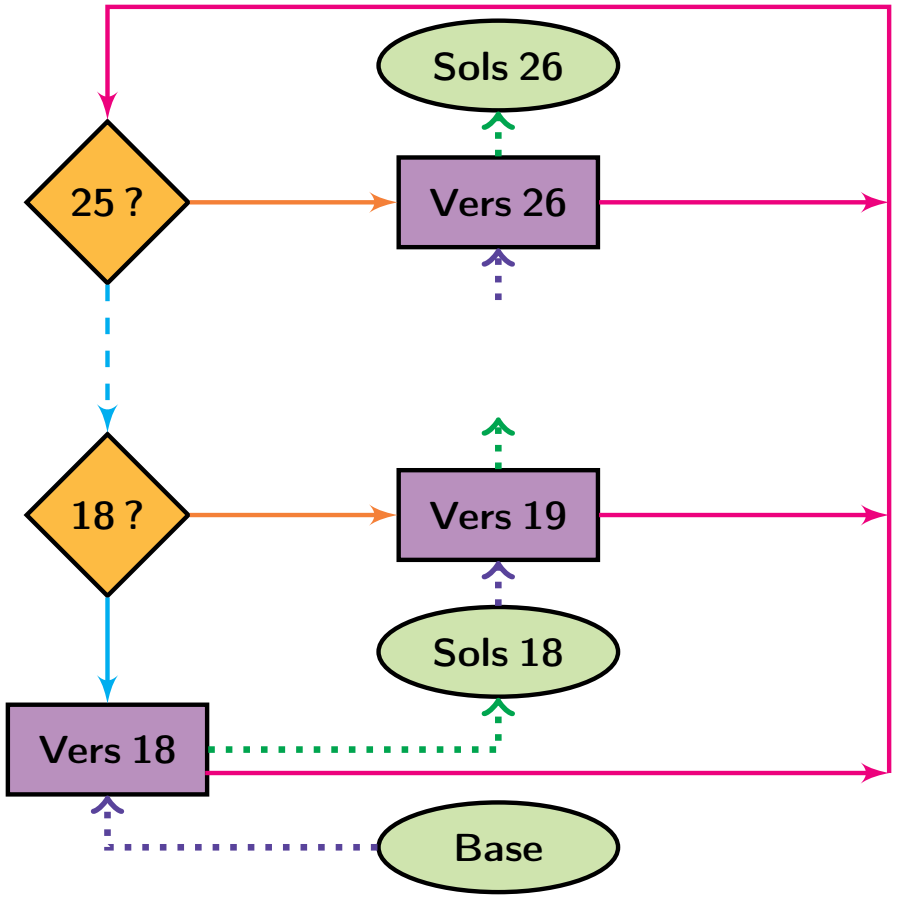
Notre approche par fragments

1. Stocker des solutions partielles jusqu'à un certain tour dans des buffers partagés
2. Tous les threads de chaque bloc prennent une solution
3. ... essayent tous les bits neutres pour ce tour
4. ... stockent les bons candidats dans le prochain buffer

Notre approche par fragments (suite.)

- Solutions de base jusqu'à #17 générées sur CPU
- Bits neutres jusqu'à #26 sur GPU
- Vérification jusqu'à #56 sur GPU
- Vérification finale sur CPU

Fragments en image



Résultats graphiques

- Matériel : une GTX-970 (S$500)
- Une solution partielle jusqu'à #56 par minute en moyenne
- $\Rightarrow$ Temps estimé pour trouver un colibri $\lessapprox$ 5 jours (moins en Guyane)

- Complexité $\equiv 2^{50.25}$ SHA-1

GPU contre CPU

- Sur un cœur de CPU @ 3.2 GHz, l'attaque prends ≈ 606 jours
- ⇒ Un GPU ≡ 140 cœurs
- (À comparer avec ≡ 40 (Grechnikov & Adinetz, 2011))
- Pour des calculs SHA-1 bruts, le rapport est 320
- ⇒ Perd seulement ×2.3 à cause des branchements (pas mal)

Tout est fini !