Projet Keccak : Implémentation SHA3

Yahya Ilyas EL BERNOUSSI Luc NGUYEN

19 décembre 2024

Objectif du projet

L'objectif de ce projet était d'implémenter l'algorithme de hachage SHA3, en suivant les spécifications officielles de Keccak. Nous avons intégré les variantes SHA3-224, SHA3-256, SHA3-384 et SHA3-512.

Structure du projet

Le projet est organisé de façon modulaire :
— **include/** : Contient les fichiers d'en-tête (`sha3.h`, `utils.h`).
— **src/** : Contient les implémentations principales (`sha3.cpp`, `utils.cpp`).
— **tests/** : Contient les fichiers de tests.
— **Makefile** : Gère la compilation et l'exécution des tests.

Choix technique

Nous avons choisi d'utiliser C++ pour ce projet car il combine performance et flexibilité. C++ permet une gestion fine des ressources mémoire, ce qui est essentiel pour un algorithme comme SHA3.
De plus, ses bibliothèques standard facilitent la manipulation de fichiers et l'organisation du code en modules. Enfin, nous étions déjà familiers avec ce langage, ce qui nous a permis de nous concentrer sur l'implémentation de l'algorithme plutôt que d'apprendre un nouveau langage.

Optimisation

Lors d'une tentative d'optimisation pour améliorer les performances de SHA3, nous avons rencontré un problème : les résultats obtenus ne correspondaient plus aux valeurs attendues. Pour éviter ces erreurs, nous avons mis en place un script python qui lance les fichiers tests garantissant que chaque modification conserve l'intégrité des résultats.

Grâce au remplacement des allocations dynamiques par des tableaux statiques et à l'utilisation des fonctions `memset` et `memcpy`, nous avons considérablement réduit le temps de traitement. Par exemple, le calcul sur un fichier de 50 Mo est passé à 10 secondes.

Tests et validation

Nous avons conçu une suite de tests pour vérifier la précision et la stabilité de l'implémentation :

- Un script Python (`test_sha3.py`) compare les résultats avec des valeurs de référence pour SHA3-224, SHA3-256, SHA3-384 et SHA3-512.
- Un test spécifique valide la gestion de la mémoire et les performances sur un fichier de 50 Mo (`fichier_50mo.txt`).
- Les tests peuvents être exécutés en une seule commande.

Comment compiler et tester

1. Compilation :

   ```
   make
   ```

2. Exécution sur un fichier :

   ```
   ./sha3 <fichier>
   ```

3. Exécution des test :

   ```
   make test
   ```

   Cette commande exécute le script Python, qui vérifie les résultats des hachages et les performances.