Specyfikacja Projektu

Tytuł Projektu: Implementacja Równoległego Wyznaczania Tablic Tęczowych w Zadaniu Łamania Haseł Szyfrowanych Algorytmem Symetrycznym DES **Autorzy**: Artem Omelchenko 324433 | Marta Stankevich 330337

---

1. Wprowadzenie

1.1. Cel Celem projektu jest stworzenie systemu do generowania tablic tęczowych (ang. rainbow tables) oraz ich użycia do łamania haseł szyfrowanych lub hashowanych przy pomocy algorytmu DES. Wybrano algorytm DES bez solenia, ponieważ tylko deterministyczne funkcje hashujące bez losowych modyfikacji (takich jak salt) pozwalają na skuteczne wykorzystanie tablic tęczowych. Solenie zmienia wynik funkcji skrótu nawet dla tego samego hasła, przez co łańcuchy stają się bezużyteczne. Projekt koncentruje się na zastosowaniu metod przetwarzania równoległego w celu znacznego skrócenia czasu generowania tablicy. Implementacja będzie wykorzystywać wiele procesów na jednej maszynie.

1.2. Zakres i Główne Założenia Projekt koncentruje się na:

- Implementacji algorytmu tworzenia tablic tęczowych (łańcuchy hash–reduction),
- Równoległej realizacji procesu generowania tablic z użyciem wielu procesów,
- Demonstracji skuteczności poprzez złamanie hasha z wygenerowanej tablicy.

Założeniem jest użycie uproszczonego algorytmu DES jako funkcji skrótu, bez zastosowania solenia (salt), w celu umożliwienia efektywnego działania tablic tęczowych.

---

2. Opis Funkcjonalny

2.1. Działanie Aplikacji i Interfejs Użytkownika Aplikacja będzie uruchamiana z poziomu wiersza poleceń i będzie obsługiwała dwa tryby:

- **Generowanie tablicy**
- **Łamanie hasha**

2.2. Wejście i Wyjście Systemu

- **Wejście:**
  - Tryb `generate`: liczba łańcuchów, długość łańcucha, liczba procesów, plik wynikowy.

  - Tryb `crack`: wartość hasha, plik z tablicą tęczową.

- **Wyjście:**
  - Plik `.csv` zawierający pary `start_password,end_password`.
  - W przypadku trybu `crack`: oryginalne hasło lub komunikat o niepowodzeniu.

---

3. Aspekty Implementacyjne

3.1. Środowisko i Technologie

- **Język:** Python 3.9
- **Biblioteki/Technologie:**
  - `hashlib` — obsługa funkcji hashujących,
  - `multiprocessing` — uruchamianie wielu procesów równolegle (Pool),
  - `argparse` — obsługa CLI,
  - `csv` — zapis i odczyt tablic.

3.2. Architektura i Algorytm

3.2.1. Generowanie tablicy tęczowej

Dla `N` losowych haseł startowych generowane są łańcuchy o długości `L` według schematu:

`start → hash → reduce → hash → ... → end`

Do pliku wynikowego zapisywane są tylko pary `start_password,end_password`, bez przechowywania całego łańcucha.

3.2.2. Redukcja

Funkcja redukcji odwzorowuje hash na hasło, wykorzystując określony alfabet (np. cyfry i małe litery). Zależność od numeru kroku pozwala uniknąć cykli i powtórzeń.

3.2.3. Proces łamania hasha

W celu złamania danego hasha program generuje potencjalne końcówki łańcuchów, które mogą odpowiadać danemu hashowi. Dla każdej dopasowanej końcówki rekonstruowany jest cały łańcuch od początku, aż zostanie odnaleziony hash wejściowy.

3.3. Strategia Zrównoleglenia

Zasadniczą częścią projektu jest równoległe przetwarzanie danych wejściowych:

- Główny proces generuje listę haseł startowych i dzieli ją na porcje (batch'e),
- Z pomocą `multiprocessing.Pool` przydzielane są te porcje do wielu procesów roboczych,
- Każdy proces niezależnie przetwarza swoją porcję haseł: tworzy łańcuchy i zapisuje wyniki,
- Wyniki mogą być łączone lokalnie lub synchronizowane za pomocą `multiprocessing.Manager.list()` albo zapisywane tymczasowo w plikach i scalane później.

---

4. Weryfikacja i Testowanie

4.1. Testy Funkcjonalne

- Generowanie tablicy dla małych danych testowych (np. łańcuchy długości 10),
- Łamanie znanych haseł (test z przygotowanym hashem),
- Obsługa błędów: nieprawidłowy hash, brak pliku, nieczytelny format.

4.2. Testy Wydajnościowe

- Pomiar czasu działania dla różnych liczby procesów: 1, 2, 4, 8,
- Analiza przyspieszenia i zgodność z prawem Amdahla,
- Weryfikacja poprawności działania równoległości na małych i dużych zestawach danych.