ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO PROJECT I

Thiết kế xây dựng chương trình thám mã tệp ZIP

NGUYỄN THẾ ANH
Anh.NT225163@sis.hust.edu.vn

Ngành Kỹ thuật máy tính - IT2

Giảng viên hướng dẫn: ThS. Nguyễn Quốc Khánh _____

Chữ kí GVHD

Khoa: Kỹ thuật máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 1/2025

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong thời công nghệ phát triển, việc số hóa dữ liệu để lưu trữ và chia sẻ ngày càng phổ biến, đặc biệt với việc sử dụng các định dạng nén nhằm làm giảm độ lớn của dữ liệu và tối ưu thời gian chia sẻ, một trong số các định dạng đó chính là ZIP. Để bảo vệ những dữ liệu quan trọng, người dùng còn có thể lựa chọn sử dụng mật khẩu để mã hóa tệp ZIP. Tuy nhiên, việc quên mất mật khẩu sau khi tạo có thể dẫn tới những tình huống khó khăn khi cần truy cập vào dữ liệu.

Vì vậy, việc khôi phục mật khẩu trở thành một vấn đề thiết thực, đặc biệt trong những trường hợp khẩn cấp hoặc khi người dùng không còn khả năng cung cấp được mật khẩu chính xác. Đề tài này được chọn nhằm mục đích xây dựng một công cụ khôi phục mật khẩu của các tệp ZIP một cách hiệu quả và an toàn.

1.2 Mục tiêu và phạm vi đề tài

Mục tiêu chính của đề tài là phát triển một chương trình khôi phục mật khẩu cho các tệp ZIP, phát triển chương trình tập trung vào hai phương pháp chính là **Brute Force Attack** và **Dictionary Attack**. Đề tài chỉ tập trung vào việc khôi phục mật khẩu cho tệp ZIP mã hóa bằng thuật toán AES. Các định dạng tệp nén và mã hóa khác không nằm trong phạm vi nghiên cứu.

1.3 Định hướng giải pháp

Một trong những khó khăn lớn nhất của bài toán là không gian mật khẩu lớn, đặc biệt khi mật khẩu được mã hóa bằng thuật toán mạnh như AES. Điều này làm tăng đáng kể thời gian xử lý và yêu cầu cao về tài nguyên tính toán. Để khắc phục những khó khăn, cần tối ưu hóa hiệu suất và tận dụng được toàn bộ tài nguyên phần cứng khi kiểm tra nhiều tổ hợp mật khẩu, những thuật toán sử dụng phải tối ưu để kết hợp với việc truyển khai xử lý song song đa luồng và đa tiến trình.

1.4 Bố cục đồ án

Phần còn lại của báo cáo này được tổ chức như sau.

Chương 2, trình bày cơ sở lý thuyết của đề tài, bao gồm các khái niệm liên quan đến định dạng tệp ZIP, cách mã hóa và bảo mật dữ liệu bằng thuật toán AES, cũng như các phương pháp dò mật khẩu phổ biến như Brute Force Attack và Dictionary Attack. Đồng thời, giới thiệu về khái niệm xử lý song song, đặc biệt là các kỹ thuật đa luồng và đa tiến trình khi thực thi chương trình. Những lý thuyết này cung cấp nền tảng quan trọng để xây dựng và phát triển công cụ trong đề tài.

Chương 3, tập trung vào phân tích các thư viện và công nghệ được sử dụng trong chương trình. Với nội dung như sau: Làm việc với tệp ZIP thông qua thư viện `pyzipper`, ứng dụng của `multiprocessing` và `threading` trong xử lý song song, cách sử dụng thư viện `tkinter` để xây dựng giao diện đồ họa trực quan cho chương trình. Bên cạnh đó, chương này cũng thảo luận về thư viện `psutil` với việc quản lý tài nguyên của hệ thông. Những nội dung này cho thấy vai trò và sự cần thiết của các thư viện trong việc hiện thực hóa giải pháp.

Chương 4, mô tả chi tiết về chương trình được xây dựng trong đề tài. Chương này trình bày kiến trúc tổng quan của chương trình, bao gồm các thành phần chính như giao diện người dùng, hệ thống quản lý tiến trình và các phương thức tấn công mật khẩu. Cuối cùng, chương này cũng mô tả các tính năng chính như Brute Force Attack, Dictionary Attack và các cơ chế hỗ trợ giao diện như hiển thị trạng thái và tiến trình thực thi.

Chương 5, tập trung đánh giá hiệu năng của chương trình dựa trên các dữ liệu thực tế. Chương này so sánh thời gian thực thi giữa các phương pháp tấn công trong các trường hợp mật khẩu có độ dài và độ phức tạp khác nhau. Bên cạnh đó, hiệu suất của chương trình khi sử dụng xử lý song song cũng được phân tích chi tiết thông qua các biểu đồ và số liệu thực nghiệm. Những kết quả đạt được không chỉ giúp đánh giá khả năng của chương trình mà còn chỉ ra những hạn chế, mở ra hướng cải thiện trong tương lai.

Bố cục này đảm bảo nội dung được trình bày một cách logic, cung cấp cái nhìn toàn diện về cơ sở lý thuyết, công cụ hỗ trợ, quá trình xây dựng và hiệu quả của chương trình.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

Nội dung ở chương trước đã giới thiệu về vấn đề khôi phục mật khẩu cho tệp ZIP và tính cấp thiết của việc phát triển một công cụ hiệu quả. Để xây dựng công cụ này, cần nắm rõ các khái niệm nền tảng liên quan đến định dạng ZIP, mã hóa AES, và các phương pháp tấn công mật khẩu phổ biến. Vì vậy ở chương 2, nội dung tập trung trình bày các nội dung lý thuyết như sau:

2.1 Định dạng ZIP

Định dạng ZIP là một phương pháp nén dữ liệu không mất mát (lossless) nổi bật và phổ biến, chủ yếu được sử dụng để lưu trữ và truyền tải nhiều tệp trong một tệp duy nhất. Phương pháp này không chỉ giúp tiết kiệm không gian lưu trữ mà còn thuận tiện trong việc chia sẻ tệp qua các kênh truyền tải có giới hạn về dung lượng. Cơ chế nén trong ZIP chủ yếu kết hợp hai thuật toán nén là **Lempel-Ziv 77** và **Mã hóa Huffman**.

Thuật toán **Lempel-Ziv 77** hoạt động theo cách tìm kiếm các chuỗi ký tự lặp lại trong dữ liệu và thay thế chúng bằng một tham chiếu đến vị trí và chiều dài của chuỗi đó. Việc này giúp giảm kích thước tệp một cách hiệu quả, bởi vì thay vì lưu trữ các chuỗi lặp lại, chỉ cần lưu trữ thông tin tham chiếu tương đối về vị trí và độ dài chuỗi. Cùng với đó, **Mã hóa Huffman** được sử dụng để tối ưu hóa mã hóa các ký tự xuất hiện với tần suất cao hơn, gán cho chúng các mã ngắn hơn, từ đó giảm dung lượng tệp nén.

Ngoài các thuật toán nén, định dạng ZIP còn hỗ trợ lưu trữ các siêu dữ liệu (Metadata), như tên tệp, thời gian sửa đổi, và mã kiểm tra CRC. Mã kiểm tra CRC giúp kiểm tra tính toàn vẹn của dữ liệu, bảo đảm rằng tệp nén không bị lỗi trong quá trình lưu trữ hoặc truyền tải. Điều này đặc biệt quan trọng trong việc đảm bảo rằng người nhận có thể giải nén tệp một cách chính xác mà không gặp phải các lỗi về dữ liệu.

2.2 Mã hóa ZIP

Trong định dạng ZIP, mã hóa không phải là một phần của cơ chế nén dữ liệu cơ bản nhưng lại là một tính năng quan trọng đối với bảo mật sau khi thực hiện cơ chế nén. Việc mã hóa trong ZIP chủ yếu được sử dụng để bảo vệ các tệp nén khỏi truy cập trái phép. Tuy nhiên, tính năng mã hóa trong ZIP cũng có sự khác biệt đáng kể về phương thức và mức độ bảo mật, tùy thuộc vào thuật toán mã hóa được áp dụng. Có hai phương thức mã hóa chính được sử dụng để bảo vệ tệp dữ liệu ZIP đó là: **Mã hóa XOR** (Truyền thống) và **Mã hóa AES** (Advanced Encryption Standard).

2.2.1 Mã hóa XOR

Mã hóa XOR là một phương pháp rất đơn giản và cơ bản, trong đó mỗi bit của dữ liệu gốc sẽ thực hiện phép toán logic XOR với một bit trong khóa mật khẩu. Quá trình này có thể được đảo ngược bằng cách áp dụng lại phép toán XOR với cùng một khóa.

Đây không phải là một phương pháp mã hóa tệp tin an toàn trong thực tế. Do tính chất đơn giản của thuật toán, nếu kẻ tấn công biết hoặc đoán được một phần dữ liệu, họ có thể dễ dàng khôi phục khóa mật khẩu và giải mã dữ liệu. Mã hóa XOR không có tính bảo mật mạnh mẽ và thường được xem là không đủ an toàn cho việc mã hóa thông tin.

Hiện nay, hầu hết mã hóa XOR chỉ được sử dụng trong các ứng dụng đơn giản và có tuổi đời lâu dài, và hầu như không bao giờ được sử dụng trong các hệ thống bảo mật hiện đại. Do tính dễ bị phá vỡ của nó, XOR chỉ thích hợp cho các trường hợp yêu cầu bảo mật thấp hoặc khi không cần bảo vệ dữ liệu nghiêm ngặt.

2.2.2 Mã hóa AES

AES là một thuật toán mã hóa đối xứng, nghĩa là cùng một khóa được sử dụng cho cả mã hóa và giải mã. Dạng mã hóa này được thiết kế dựa trên một cấu trúc gọi là **sơ đồ Ma trận** (Substitution-Permutation Network - SPN), trong đó dữ liệu gốc được biến đổi qua một loạt các vòng mã hóa.

Mã hóa AES làm việc trên từng khối dữ liệu có kích thước cố định là 128-bit, chia khối dữ liệu thành 16 byte và xử lý từng byte thông qua các phép toán mã hóa. Ban đầu, khóa gốc được người dùng lựa chọn và có độ dài (bit) cố định tùy thuộc vào mỗi phiên bản (AES-128, AES-192, AES-256). Khóa gốc sau khi xác định sẽ được chuyển đổi thành một loạt các khóa con (round keys). Tất cả các vòng mã trong AES sau đó đều sẽ sử dụng các khóa con này. Số lượng của khóa con được tạo ra phụ thuộc vào độ dài của khóa gốc và số vòng mã hóa, cụ thể:

- AES-128: Thực hiện 10 vòng mã hóa, yêu cầu 11 khóa con.
- AES-192: Thực hiện 12 vòng mã hóa, yêu cầu 13 khóa con.
- AES-256: Thực hiện 14 vòng mã hóa, yêu cầu 15 khóa con.

Trong mỗi vòng, khóa gốc được chia thành nhiều từ (words) 32-bit và thực hiện qua các phép toán. Quy trình mỗi vòng bao gồm bốn bước chính: **SubBytes**, **ShiftRows**, **MixColumns** và **AddRoundKey**. Những bước này nhằm loại bỏ sự tuyến tính liên quan đến việc có thể tìm ra mối quan hệ trực tiếp giữa các thành phần của khóa (hoặc mật khẩu) và kết quả mã hóa thông qua các phép toán tuyến tính.

1. **SubBytes**: Đây là một phép thay thế phi tuyến tính được thực hiện thông qua S-Box, một bảng thay thế dựa trên một thuật toán cụ thể, giúp tạo ra các giá trị ngẫu nhiên và phi tuyến tính.
2. **ShiftRows**: Phép này dịch chuyển các dòng trong ma trận dữ liệu, tạo sự thay đổi không đều giữa các phần dữ liệu, từ đó làm giảm tính dự đoán của các bit.
3. **MixColumns**: Thao tác này thực hiện trộn các cột trong ma trận, giúp làm phức tạp thêm quá trình mã hóa, không cho phép rút ra các mối quan hệ tuyến tính đơn giản giữa các byte của khối dữ liệu.
4. **AddRoundKey**: Đây là phép toán đơn giản, nhưng việc thêm khóa vòng vào dữ liệu sau mỗi vòng mã hóa cũng giúp phá vỡ mọi mối liên hệ tuyến tính có thể có giữa dữ liệu ban đầu và kết quả mã hóa.

Tại vòng mã hóa cuối, tất cả các bước đều được thực hiện ngoại trừ MixColumns.

Điểm mạnh của AES trong bảo mật nằm ở sự kết hợp của các phép toán tuyến tính và không tuyến tính, cùng với sự thay đổi khóa trong mỗi vòng mã hóa, làm cho việc rút ra các mối quan hệ giữa dữ liệu và khóa trở nên vô cùng khó khăn.

Sự kết hợp giữa 2 loại tính toán này khiến việc tấn công bằng các phương pháp phân tích tuyến tính (Linear cryptanalysis) trở nên không khả thi trong thời gian thực tế.

2.3 Giải mã ZIP

Giải mã tệp ZIP được bảo vệ bằng mã hóa XOR hoặc AES đòi hỏi các phương pháp và mức độ bảo mật khác nhau tùy thuộc vào cơ chế mã hóa được sử dụng.

2.3.1 Mã hóa XOR

Quá trình giải mã tệp ZIP sử dụng XOR có thể được tóm tắt qua các bước cơ bản sau. Đầu tiên, tệp ZIP sẽ yêu cầu mật khẩu người dùng để tạo ra chuỗi khóa. Chuỗi khóa này được tạo ra từ việc kết hợp mật khẩu với một giá trị ban đầu (`key0`), và có thể sử dụng các phép toán bổ sung như XOR để tạo ra hai khóa phụ (`key1`, `key2`).

Tiếp theo, các giá trị byte của tệp mã hóa sẽ được XOR với các byte của chuỗi khóa, qua đó phục hồi lại dữ liệu gốc. Điều này cho phép phục hồi tệp ZIP mà không cần phải sử dụng thêm bất kỳ thông tin nào ngoài chuỗi khóa đã được sinh ra từ mật khẩu vì XOR là một phép đối xứng.

2.3.2 Mã hóa AES

Về cơ bản, quy trình giải mã trong AES là quá trình đảo ngược các phép toán đã thực hiện trong quá trình mã hóa. Quá trình này bao gồm các bước sau:

1. **AddRoundKey**: Bước đầu tiên trong giải mã là thực hiện phép toán XOR giữa dữ liệu mã hóa và khóa vòng hiện tại. Khóa vòng được tạo ra bằng cách áp dụng hàm mở rộng khóa từ khóa ban đầu.
2. **InvShiftRows**: Phép đảo ngược của bước ShiftRows trong quá trình mã hóa. Trong mã hóa, ShiftRows dịch chuyển các dòng của ma trận dữ liệu (mỗi dòng sẽ được dịch chuyển một số vị trí nhất định). Trong giải mã, phép đảo ngược này sẽ dịch chuyển lại các dòng về vị trí ban đầu để khôi phục lại dữ liệu.
3. **InvSubBytes**: Đây là bước đảo ngược của phép toán SubBytes, nơi các byte dữ liệu trong ma trận được thay thế bằng giá trị khác từ bảng thay thế (S-Box). Để giải mã, chúng ta sẽ tra cứu các giá trị ngược lại trong bảng thay thế để khôi phục lại các byte ban đầu.
4. **InvMixColumns**: Đây là phép toán đảo ngược của MixColumns trong quá trình mã hóa. MixColumns là phép toán giúp trộn các cột trong ma trận dữ liệu, làm cho mỗi cột của ma trận dữ liệu trở nên phụ thuộc vào tất cả các byte của cột đó. Để giải mã, ta phải áp dụng một phép toán ngược lại, nhằm tách các byte ra khỏi mối quan hệ phức tạp đã tạo ra trong bước mã hóa.

Số vòng trong quá trình thực hiện trong quá trình giải mã giống với thực hiện mã hóa. Trong vòng cuối cùng, bước InvMixColumns được bỏ qua, tương tự như cách mã hóa bỏ qua bước MixColumns ở vòng cuối.

2.4 Tổng kết

Việc tấn công vào thuật toán AES thông qua phương pháp phân tích tuyến tính không khả thi trong thực tế, bởi vì cấu trúc của AES được thiết kế để loại bỏ mối liên hệ tuyến tính giữa dữ liệu và khóa mã hóa. Cụ thể, các bước mã hóa phi tuyến như SubBytes, kết hợp với các phép hoán vị phức tạp như ShiftRows và MixColumns, đã tạo ra sự biến đổi dữ liệu không thể dự đoán được theo bất kỳ mô hình tuyến tính nào. Điều này khiến phân tích tuyến tính không đủ khả năng để khai thác lỗ hổng trong thiết kế của AES.

Thay vì sử dụng phân tích tuyến tính, một cách tiếp cận thực tế hơn để giải mã tệp ZIP mã hóa AES là xây dựng không gian mật khẩu khả dĩ và thử từng mật khẩu trong danh sách này. Việc xây dựng không gian mật khẩu cần dựa trên các đặc điểm thực tế của mật khẩu mà người dùng có thể sử dụng, như độ dài, ký tự phổ biến, hoặc các mẫu mật khẩu thông thường. Quy trình thử mật khẩu sẽ được triển khai như sau:

Trước tiên, từ danh sách mật khẩu đã xây dựng, từng mật khẩu sẽ được sử dụng để tái tạo khóa AES thông qua quá trình mở rộng khóa tiêu chuẩn. Sau đó, khóa

được áp dụng để giải mã tệp ZIP. Dữ liệu giải mã được kiểm tra dựa trên cấu trúc và thông tin của tệp, chẳng hạn như header tệp ZIP hoặc mã kiểm tra CRC. Nếu dữ liệu giải mã hợp lệ, có thể xác nhận mật khẩu đã được tìm thấy; nếu không, quá trình sẽ tiếp tục với mật khẩu tiếp theo trong danh sách.

Việc triển khai phương pháp này phụ thuộc vào sự tối ưu hóa trong việc tạo và thử mật khẩu, nhằm giảm thời gian xử lý và tăng khả năng thành công. Một yếu tố quan trọng là tính hiệu quả của thuật toán kiểm tra tính hợp lệ của dữ liệu sau khi giải mã, bởi điều này ảnh hưởng trực tiếp đến tốc độ của toàn bộ quy trình.

CHƯƠNG 3. THƯ VIỆN SỬ DỤNG

Với chiến lược thám mã đã đề ra, nội dung của chương 3 đi sâu vào các công cụ và thư viện Python cụ thể được sử dụng để hiện thực hóa giải pháp này. Đặc biệt kể đến các thư viện như pyzipper, multiprocessing, threading, tkinter và psutil, đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất xử lý và nâng cao trải nghiệm người dùng. Chương này sẽ phân tích cách thức hoạt động của từng thư viện và ứng dụng thực tế của chúng trong chương trình khôi phục mật khẩu ZIP.

3.1 Thư viện pyzipper

Pyzipper là một thư viện mở rộng của Python's zipfile, cung cấp các tính năng mã hóa và giải mã hóa nâng cao cho các tệp ZIP. Thư viện hỗ trợ mã hóa AES với các độ dài khóa 128, 192, và 256 bit, giúp đảm bảo tính bảo mật cao cho các tệp ZIP, điều mà thư viện zipfile không hỗ trợ.

Thư viện này còn được thiết kế để tương thích với các công cụ nén phổ biến khác như 7-Zip và WinZip, giúp người dùng dễ dàng xử lý tệp ZIP đã được mã hóa trên nhiều nền tảng mà không gặp phải các vấn đề tương thích. Với giao diện lập trình (API) giống với zipfile, pyzipper cho phép các lập trình viên dễ dàng tích hợp vào các dự án hiện có mà không gặp phải khó khăn trong việc học lại các cú pháp hay phương thức mới.

Tuy nhiên, mặc dù pyzipper là một công cụ mạnh mẽ, thư viện này đã ít được duy trì trong vài năm qua và không nhận được cập nhật mới thường xuyên. Điều này đặt ra vấn đề về sự bền vững khi lựa chọn pyzipper cho các dự án dài hạn. Mặc dù đã có tuổi đời khoảng 6 năm nhưng pyzipper vẫn được sử dụng rộng rãi trong cộng đồng Python nhờ vào tính năng mã hóa AES mạnh mẽ mà nó cung cấp.

Trong chương trình sử dụng thư viện pyzipper, ba hàm chủ yếu được triển khai để thao tác với các tệp ZIP mã hóa AES, bao gồm `setpassword()`, `testzip()`, và `extract()`. Những hàm này đóng vai trò quan trọng trong việc kiểm tra và giải nén các tệp ZIP bảo vệ bằng mật khẩu, đồng thời đảm bảo tính toàn vẹn và bảo mật của dữ liệu.

Hàm `setpassword()` trong pyzipper được sử dụng để cung cấp mật khẩu cho việc mở tệp ZIP. Phục vụ thực hiện các thao tác tiếp theo như kiểm tra mật khẩu đúng hay sai, và giải nén dữ liệu. Mặc dù `setpassword()` chỉ là một phương thức đơn giản, nhưng nó là bước không thể thiếu trong quá trình làm việc với các tệp bảo vệ bằng mật khẩu.

Sau khi thiết lập mật khẩu thông qua `setpassword()`, một trong những

phương thức quan trọng trong việc kiểm tra tính toàn vẹn của tệp ZIP là hàm `testzip()`. Phương thức này được sử dụng để xác định từng tệp trong kho lưu trữ ZIP có bị hỏng hay không thông qua việc kiểm tra mã CRC-32. Nếu mật khẩu đã được cung cấp đúng, việc kiểm tra sẽ thành công và trả về kết quả là `None`. Ngược lại, nếu mật khẩu sai hoặc tệp bị hỏng, chương trình sẽ báo lỗi hoặc trả về tên tệp bị lỗi. Hàm `testzip()` là một công cụ mạnh mẽ giúp xác minh tính toàn vẹn của dữ liệu trong tệp ZIP trước khi tiếp tục giải nén.

Hàm `extract()` trong pyzipper cho phép giải nén một thành phần (tệp) từ kho lưu trữ ZIP đã được mã hóa. Sau khi mật khẩu được xác thực và tính toàn vẹn của tệp được kiểm tra thông qua `testzip()`, có thể sử dụng hàm này để giải nén các tệp vào thư mục đích. Với việc hỗ trợ mã hóa AES, `extract()` cũng đảm bảo rằng chỉ khi mật khẩu đúng, quá trình giải mã và giải nén mới diễn ra thành công.

Tổng kết lại, thư viện pyzipper là một công cụ giúp thao tác với tệp ZIP mã hóa AES, xác thực mật khẩu, kiểm tra tính toàn vẹn của tệp, và giải nén dữ liệu một cách hiệu quả và an toàn.

3.2 Thư viện multiprocessing

Trong Python, thư viện `multiprocessing` là một công cụ mạnh mẽ giúp thực hiện các tác vụ song song, khai thác khả năng xử lý đa lõi của các bộ vi xử lý hiện đại. Đây là một phần quan trọng của Python Standard Library, giúp tối ưu hóa các chương trình xử lý tính toán nặng hoặc các tác vụ cần thực hiện đồng thời. Thư viện này hỗ trợ việc chia nhỏ các công việc thành nhiều tiến trình (process), cho phép chương trình xử lý các công việc một cách hiệu quả hơn, giảm thời gian thực thi tổng thể.

Một trong những lợi thế chính của `multiprocessing` là nó giúp giải quyết vấn đề hạn chế của đa luồng trong Python. Do Global Interpreter Lock (GIL) trong Python, các tiến trình đồng thời không thể tận dụng tốt khả năng đa lõi khi sử dụng thư viện `threading`. Tuy nhiên, `multiprocessing` khởi động các tiến trình riêng biệt, mỗi tiến trình có bộ nhớ riêng biệt, giúp vượt qua hạn chế này và tận dụng tối đa sức mạnh của hệ thống đa lõi.

Trong chương trình sử dụng thư viện `multiprocessing`, có hai hàm quan trọng cần được đề cập là `Process()`, và `Queue()`. Những hàm này đóng vai trò quan trọng trong việc quản lý các tiến trình và trao đổi dữ liệu giữa các tiến trình.

Hàm `Process()` cho phép tạo và quản lý các tiến trình riêng biệt. Mỗi tiến trình được khởi tạo và thực hiện một công việc độc lập, không chia sẻ bộ nhớ với

các tiến trình khác, điều này giúp tránh được vấn đề xung đột dữ liệu khi chạy song song.

Hàm `Queue()` giúp trao đổi dữ liệu giữa các tiến trình. Dữ liệu được đưa vào hàng đợi chung và có thể được các tiến trình khác lấy ra để xử lý. Đây là một cách hiệu quả để giao tiếp và đồng bộ hóa các tiến trình, giúp chúng có thể chia sẻ thông tin và hợp tác trong quá trình làm việc.

Tổng kết lại, thư viện `multiprocessing` là một công cụ hữu ích để thực hiện đa tiến trình trong Python, giải quyết được vấn đề GIL của xử lý đa luồng, giúp tăng hiệu quả và giảm thời gian thực thi chương trình. Thư viện này đặc biệt phù hợp với các ứng dụng cần xử lý tính toán song song hoặc các tác vụ yêu cầu độ trễ thấp và hiệu suất cao.

3.3 Thư viện threading

Thư viện `threading` trong Python là một công cụ mạnh mẽ giúp thực hiện các tác vụ đồng thời trong một chương trình, mà không làm gián đoạn các hoạt động khác, đặc biệt hữu ích trong các ứng dụng giao diện người dùng (GUI) hoặc các ứng dụng yêu cầu xử lý nhiều tác vụ vào-ra song song. Thư viện này cho phép tạo và quản lý các luồng, mỗi luồng có thể thực hiện một công việc độc lập nhưng cùng tồn tại trong một chương trình, giúp tối ưu hóa hiệu suất và trải nghiệm với chương trình.

Trong chương trình xây dựng, thư viện `threading` có hai tác vụ chủ yếu được thực hiện trong các luồng riêng biệt: chạy quá trình tấn công mật khẩu và cập nhật tiến trình của giao diện người dùng. Việc sử dụng các luồng này không chỉ giúp giảm thiểu độ trễ mà còn duy trì tính mượt mà cho giao diện người dùng trong khi các công việc nặng đang được thực thi.

3.4 Thư viện tkinter

Thư viện `tkinter` là một công cụ phổ biến trong Python, dùng để xây dựng các ứng dụng giao diện đồ họa người dùng (GUI). Với khả năng cung cấp các thành phần giao diện cơ bản như cửa sổ, nhãn, nút bấm, ô nhập liệu và thanh tiến trình, `tkinter` cho phép phát triển các ứng dụng dễ dàng và hiệu quả mà không cần phải cài đặt thêm phần mềm bên ngoài. Điều này giúp tiết kiệm nhiều thời gian và công sức cho lập trình viên. Đồng thời, `tkinter` còn có một cộng đồng lớn và tài liệu phong phú, hỗ trợ tốt cho việc phát triển ứng dụng.

3.5 Thư viện psutil

Thư viện `psutil` (Python System and Process Utilities) là một công cụ mạnh mẽ trong việc truy xuất và quản lý thông tin về các tiến trình đang chạy cũng như

tài nguyên hệ thống. `psutil` cung cấp các API hữu ích để giám sát việc sử dụng CPU, bộ nhớ, và các tài nguyên khác của hệ thống, đồng thời cho phép lập trình viên kiểm soát tiến trình và phân phối tài nguyên hiệu quả.

Thư viện `psutil` có những ứng dụng quan trọng, đặc biệt là trong việc tối ưu hóa việc sử dụng tài nguyên CPU khi thực hiện các tác vụ xử lý song song. Các tính năng chính của `psutil` được sử dụng bao gồm:

Quản lý CPU Affinity: Thông qua phương thức `cpu_affinity()` của lớp `psutil.Process`, chương trình có thể gán các tiến trình đang cho các lõi CPU cụ thể. Điều này giúp tận dụng tối đa tài nguyên của hệ thống, đặc biệt khi hệ thống có nhiều lõi xử lý, nâng cao hiệu suất của các tác vụ song song.

Giám sát tài nguyên: Thư viện này cung cấp khả năng theo dõi việc sử dụng tài nguyên của hệ thống và các tiến trình, giúp đảm bảo rằng không có tiến trình nào chiếm dụng quá nhiều tài nguyên, điều này rất quan trọng trong các ứng dụng yêu cầu hiệu suất cao như ứng dụng tấn công mật khẩu.

Ưu điểm của thư viện `psutil` là khả năng cung cấp thông tin chi tiết về hệ thống và các tiến trình đang chạy, giúp lập trình viên tối ưu hóa việc sử dụng tài nguyên và kiểm soát hiệu suất. `psutil` cũng hỗ trợ nhiều nền tảng, từ Windows đến Linux và macOS, giúp tăng tính linh hoạt và tính tương thích cho ứng dụng. Việc sử dụng thư viện này đặc biệt quan trọng trong các ứng dụng cần xử lý song song hoặc tính toán nặng, vì nó giúp đảm bảo hệ thống vẫn hoạt động ổn định trong suốt quá trình thực thi.

CHƯƠNG 4. MÔ TẢ CHƯƠNG TRÌNH

4.1 Thiết kế kiến trúc

4.1.1 Lựa chọn kiến trúc phần mềm

Chương trình được lựa chọn phát triển dựa trên kiến trúc phần mềm **MVC (Model-View-Controller)** nhằm tối ưu hóa việc phân tách trách nhiệm giữa các thành phần chính. Kiến trúc này không chỉ hỗ trợ cải thiện khả năng bảo trì và mở rộng, mà còn nâng cao tính rõ ràng trong việc quản lý mã nguồn. Trong đó:

Model bao gồm các lớp như **ProgressManager** và **Cracker**. Lớp **Progress-Manager** chịu trách nhiệm quản lý trạng thái của quá trình giải mã, bao gồm thông tin về số lượng mật khẩu đã thử, độ dài mật khẩu hiện tại, và tệp wordlist đang sử dụng. Lớp **Cracker** tập trung xử lý logic chính của chương trình, bao gồm hai phương pháp tấn công là *Brute Force* và *Dictionary Attack*. Các lớp này hoàn toàn độc lập và không tương tác trực tiếp với giao diện người dùng.

View được đại diện bởi lớp **GUI**, chịu trách nhiệm hiển thị giao diện người dùng. Sử dụng thư viện Tkinter, **GUI** cung cấp các thành phần giao diện như ô nhập liệu, nút bấm, thanh tiến độ và thông báo trạng thái, hỗ trợ người dùng tương tác với chương trình một cách thuận tiện và trực quan.

Controller cũng được tích hợp vào lớp **GUI**, đóng vai trò điều phối các sự kiện giữa giao diện người dùng và logic xử lý. Controller chịu trách nhiệm nhận yêu cầu từ giao diện (như nhấn nút bắt đầu hoặc dừng lại), sau đó điều khiển luồng xử lý trong lớp **Cracker** và cập nhật kết quả trở lại giao diện.

4.1.2 Tổng quan thiết kế

Ba module chính bao gồm: `gui`, `cracker`, và `progress_manager`.

Module `gui` chứa giao diện đồ họa và chức năng điều khiển sự kiện. Module `cracker` chịu trách nhiệm thực thi logic tấn công và quản lý tiến trình tấn công. Cuối cùng, module `progress_manager` đảm nhiệm việc lưu trữ, khôi phục và theo dõi trạng thái của quá trình giải mã mật khẩu.

Trong quá trình hoạt động, GUI phụ thuộc vào hai module còn lại để thực hiện nhiệm vụ. Khi người dùng gửi yêu cầu, GUI truyền thông tin tới Cracker, đồng thời yêu cầu ProgressManager lưu trạng thái hiện tại nếu cần. Mối quan hệ giữa các module được thiết kế theo nguyên tắc giảm thiểu sự phụ thuộc, giúp chương trình dễ dàng mở rộng trong tương lai.

4.2 Thiết kế chi tiết

4.2.1 Thiết kế giao diện

Giao diện trực quan được thiết kế với kích thước khung chính cố định là 550x500 pixel, chia thành bốn phần chức năng chính: chọn tệp tin ZIP, cấu hình chế độ tấn công, cài đặt số lượng luồng xử lý và hiển thị trạng thái tiến trình.

Giao diện sử dụng các thành phần cơ bản của Tkinter như ô nhập liệu (Entry), nút bấm (Button), thanh tiến độ (Progressbar), và nhãn (Label). Bố cục được tối ưu hóa để đảm bảo tính thẩm mỹ và dễ sử dụng. Tất cả các thành phần đều tuân thủ tiêu chuẩn giao diện người dùng, bao gồm màu sắc trung tính, khoảng cách hợp lý giữa các thành phần và phông chữ Arial kích thước 10 đến 12 pixel.

4.2.2 Thiết kế lớp

Ba lớp chính trong chương trình là `Cracker`, `GUI`, và `ProgressManager`, được thiết kế với trách nhiệm rõ ràng như sau:

Lớp Cracker: chịu trách nhiệm xử lý hai phương pháp tấn công, với các thuộc tính chính như `progress_manager` để quản lý trạng thái tiến trình, các phương thức chính như `brute_force()` để thực hiện kiểm tra toàn bộ tổ hợp mật khẩu, và `dictionary_attack()` để kiểm tra mật khẩu từ danh sách có sẵn.

Lớp GUI: chịu trách nhiệm tạo giao diện đồ họa và điều phối luồng xử lý. Các thuộc tính quan trọng bao gồm các thành phần giao diện như nút bấm, ô nhập liệu và thanh tiến độ. Phương thức `setup_gui()` thiết lập bố cục giao diện, trong khi `start_attack()` chịu trách nhiệm khởi động tiến trình giải mã.

Lớp ProgressManager: lưu trữ và khôi phục trạng thái của tiến trình giải mã, đảm bảo rằng người dùng có thể tạm dừng và tiếp tục công việc mà không bị mất dữ liệu. Các phương thức quan trọng bao gồm `save_progress()` để lưu trạng thái hiện tại vào tệp JSON và `load_progress()` để khôi phục từ tệp lưu trữ.

4.3 Luồng xử lý

Chương trình được thiết kế với hai luồng xử lý chính, tương ứng với hai phương pháp tấn công phổ biến: Brute Force và Dictionary Attack. Mỗi phương pháp này có quy trình vận hành riêng biệt nhưng được điều phối thông qua các tiến trình con, sử dụng thư viện multiprocessing để tối ưu hóa hiệu suất của CPU. Quy trình xử lý được khởi chạy khi người dùng lựa chọn phương pháp tấn công thông qua giao diện đồ họa (GUI). Các thông số cấu hình do người dùng nhập sẽ được chuyển đến lớp xử lý chính `Cracker` để bắt đầu quá trình tấn công.

4.3.1 Quy trình xử lý của phương pháp Brute Force

Trong phương pháp Brute Force, luồng xử lý được tổ chức thành hai loại tiến trình chính: tiến trình cung cấp và tiến trình xử lý. Ban đầu, tiến trình cung cấp được khởi tạo để sinh ra các chỉ số đại diện cho tổ hợp mật khẩu và chia chúng thành từng lô nhỏ rồi đẩy vào hàng đợi. Kích thước của mỗi lô có thể được tùy chỉnh, nhưng cần được thiết kế hợp lý để giảm thiểu chi phí vào-ra và tránh làm tràn dung lượng bộ nhớ.

Song song với quá trình này, nhiều tiến trình con xử lý được khởi tạo, với số lượng tùy thuộc vào cấu hình của người dùng. Mỗi tiến trình con sẽ lấy một lô mật khẩu từ hàng đợi, sau đó thực hiện việc tạo mật khẩu từ các chỉ số và kiểm tra tính hợp lệ của mật khẩu đó với tệp ZIP mục tiêu. Khi một mật khẩu đúng được tìm thấy, tiến trình con tương ứng sẽ thiết lập cờ dừng toàn cục, từ đó thông báo kết quả cho người dùng và chấm dứt toàn bộ quá trình kiểm tra.

4.3.2 Quy trình xử lý của phương pháp Dictionary Attack

Phương pháp Dictionary Attack có cách tiếp cận khác biệt nhưng luồng xử lý vẫn được cấu trúc tương tự như Brute Force. Tiến trình cung cấp thay vì sinh chỉ số thì sẽ đọc lần lượt danh sách mật khẩu từ tệp dữ liệu đầu vào. Việc đọc diễn ra tuần tự đảm bảo rằng chương trình không tiêu tốn quá nhiều bộ nhớ, đặc biệt khi danh sách mật khẩu có kích thước lớn. Các tiến trình xử lý sẽ lấy các lô từ hàng đợi và tiến hành kiểm tra từng mật khẩu với tệp ZIP theo cách tương tự như phương pháp Brute Force.

4.3.3 Chiến lược tối ưu hóa

Như nội dung đã phân tích trong chương Cơ sở lý thuyết, việc tấn công bằng phương pháp phân tích tuyến tính gần như là bất khả thi với mã hóa AES. Vậy nên quá trình thám mã được triển khai theo hướng tạo ra khóa từ danh sách tổ hợp và thử với dữ liệu đã mã khóa để tìm ra được mật khẩu đúng. Tuy nhiên, nếu dữ liệu của tệp ZIP quá lớn, hướng tiếp cận này không đạt hiệu quả. Hướng tối ưu được triển khai là kiểm thử với Metadata của tệp ZIP như lớp sàng lọc ban đầu, sau đó mới kiểm thử với tệp nhỏ nhất trong tệp ZIP để xác định được mật khẩu chính xác.

Ở cả hai chế độ, việc phân bổ tài nguyên CPU được tối ưu hóa bằng cách sử dụng p.cpu_affinity() từ thư viện psutil, cho phép kiểm soát cách các tiến trình sử dụng nhân CPU. Điều này đảm bảo hiệu suất cao và cân bằng tải giữa các tiến trình.

4.3.4 Theo dõi tiến trình và lưu trạng thái

Trong cả hai phương pháp, việc theo dõi tiến trình được thực hiện thông qua các biến dùng chung của multiprocessing như passwords_tried và total_password. Các giá trị này được cập nhật liên tục và hiển thị trên giao diện người dùng thông qua thanh tiến trình và nhãn trạng thái. Để đảm bảo tính nhất quán của dữ liệu, các biến dùng chung này được bảo vệ bằng cơ chế khóa khi chúng được cập nhật.

Khi người dùng yêu cầu tạm dừng, chương trình lưu trạng thái hiện tại vào một tệp JSON thông qua lớp ProgressManager. Trạng thái lưu trữ bao gồm số lượng mật khẩu đã thử, chỉ số hiện tại trong phương pháp Brute Force, hoặc vị trí trong danh sách mật khẩu khi thực hiện Dictionary Attack. Khi chương trình được khởi động lại, trạng thái này sẽ được đọc và khôi phục, đảm bảo tính liên tục trong quá trình xử lý.

4.3.5 Xử lý lỗi và đảm bảo ổn định

Chương trình được thiết kế để xử lý lỗi trong các tiến trình con nhằm đảm bảo tính ổn định. Khi một tiến trình con gặp lỗi, thông báo sẽ được gửi đến các tiến trình khác để dừng an toàn, giúp tránh tình trạng treo hoặc "deadlock". Điều này đảm bảo rằng ngay cả trong trường hợp xảy ra sự cố, chương trình vẫn có thể phục hồi và tiếp tục hoạt động mà không gây gián đoạn đáng kể.

4.4 Kết quả đạt được

4.4.1 Chức năng chính

Chương trình đã được phát triển thành công với các chức năng chính như sau:

Brute Force Attack: Chương trình cho phép thực hiện phương pháp tấn công thử toàn bộ tổ hợp mật khẩu dựa trên tập ký tự (charset) và độ dài mật khẩu được chỉ định. Tính năng này được thiết kế để xử lý cả mật khẩu có độ dài lớn, đảm bảo kiểm tra toàn bộ tổ hợp một cách tuần tự hoặc song song, tùy thuộc vào cấu hình số luồng xử lý.

Dictionary Attack: Với phương pháp tấn công dựa trên từ điển, chương trình hỗ trợ người dùng kiểm tra mật khẩu bằng cách sử dụng danh sách các từ được cung cấp trong tệp wordlist. Chương trình đảm bảo khả năng xử lý các danh sách có kích thước lớn bằng cách phân chia công việc thành các lô nhỏ để xử lý.

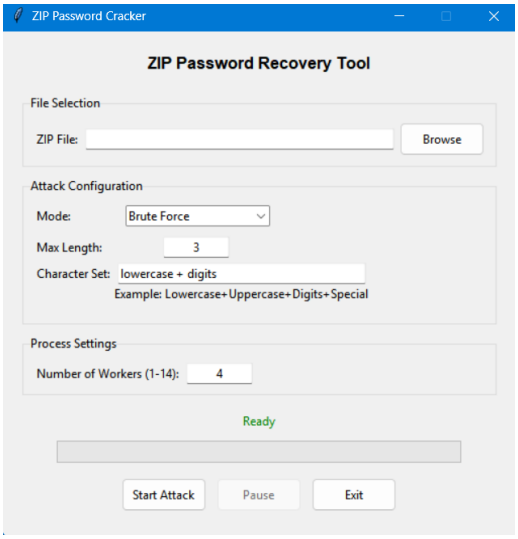
Pause/Resume Functionality: Người dùng có thể tạm dừng tiến trình tấn công mật khẩu bất kỳ lúc nào và tiếp tục lại từ vị trí đã dừng.

Progress Tracking: Thanh tiến độ hiển thị trong giao diện cung cấp thông tin trực quan về trạng thái hiện tại của tiến trình tấn công, bao gồm tổng số mật khẩu đã thử, độ dài mật khẩu đang xử lý, và thời gian ước tính để hoàn thành.

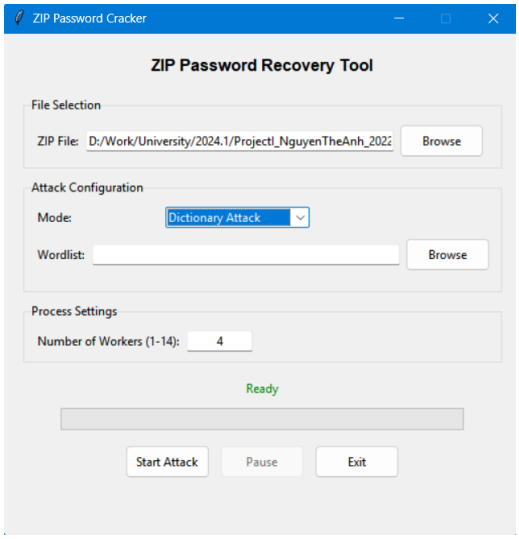
Multi-processing Support: Chương trình sử dụng thư viện multiprocessing để tận dụng tối đa khả năng của CPU. Số lượng luồng xử lý có thể được cấu hình bởi người dùng, giúp tăng hiệu suất và giảm thời gian thực hiện, đặc biệt khi xử lý khối lượng công việc lớn.

4.4.2 Minh họa chương trình

Giao diện chính của chế độ tấn công **Brute Force**, yêu cầu nhập đường dẫn tệp ZIP, lựa chọn độ dài mật khẩu, bộ ký tự và số tiến trình thực hiện xử lý song song. Với chế độ tấn công **Dictionary Attack**, đường dẫn từ điển mật khẩu được yêu cầu thay thế cho độ dài mật khẩu và bộ ký tự.

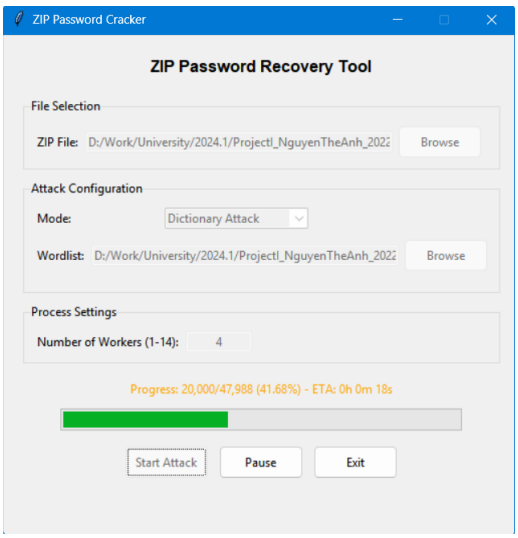


Hình 4.1: GUI Brute Force

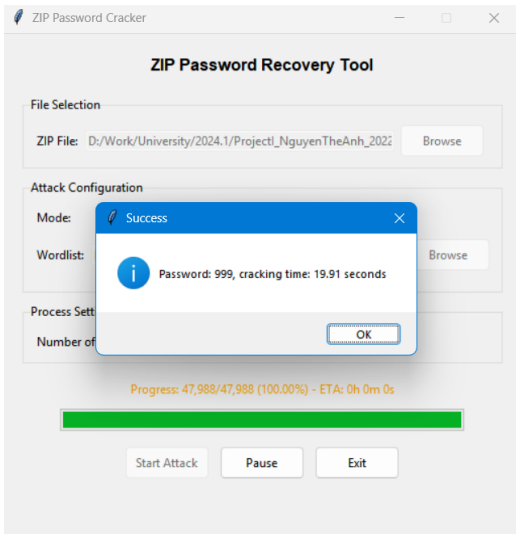


Hình 4.2: GUI Dictionary Attack

Khi chương trình bắt đầu thực hiện tấn công, thanh tiến trình được cập nhật để hiển thị tiến độ và thời gian dự kiến. Thông báo kết thúc sẽ được gửi tới người dùng khi chương trình tìm thấy mật khẩu hoặc khi tấn công mọi tổ hợp nhưng không tìm thấy mật khẩu đúng.



Hình 4.3: Đang tấn công



Hình 4.4: Thông báo thành công

CHƯƠNG 5. KIỂM THỬ VÀ ĐÁNH GIÁ

5.1 Mục tiêu và phương án kiểm thử

Quá trình kiểm thử nhằm đánh giá các yếu tố cốt lõi của chương trình, bao gồm hiệu suất, tính ổn định và độ chính xác trong việc thực hiện các phương pháp tấn công mật khẩu trên các tệp ZIP có cấu trúc và dung lượng khác nhau. Thông qua đó, kiểm thử cung cấp dữ liệu định lượng và định tính để phân tích khả năng đáp ứng các tiêu chí thiết kế, đảm bảo chương trình hoạt động đúng như kỳ vọng trong các kịch bản thực tế.

Các mục tiêu cụ thể bao gồm:

Hiệu suất: Xác định tốc độ thực thi của các phương pháp tấn công mật khẩu trên từng loại tệp ZIP, bao gồm việc sử dụng hiệu quả tài nguyên phần cứng như CPU và tối ưu hóa trong môi trường đa tiến trình.

Tính ổn định: Kiểm tra khả năng duy trì hoạt động liên tục của chương trình, bao gồm xử lý lỗi, phục hồi từ trạng thái tạm dừng và đảm bảo không xảy ra tình trạng treo hoặc deadlock trong quá trình thực thi.

Độ chính xác: Đảm bảo chương trình luôn tìm ra mật khẩu chính xác của tệp ZIP, bất kể độ phức tạp của mật khẩu hoặc kích thước danh sách từ điển.

Để đảm bảo các mục tiêu này, kiểm thử sẽ được tiến hành trên bốn tệp ZIP với các đặc điểm khác nhau, bao gồm:

Tệp ZIP 0 (1KB): Chứa một tệp duy nhất với dung lượng rất nhỏ, được thiết kế để kiểm tra khả năng hoạt động của chương trình trong điều kiện đơn giản nhất. Đây là đơn vị kiểm thử cơ bản nhằm loại trừ các yếu tố gây nhiễu từ tài nguyên hệ thống hoặc cấu trúc phức tạp của tệp.

Tệp ZIP 1 (100KB): Mô phỏng tình huống thực tế với một tệp đơn lẻ có kích thước nhỏ, thường gặp trong các trường hợp sử dụng cá nhân hoặc học thuật.

Tệp ZIP 2 (200KB, 300KB, 500KB): Đại diện cho các tệp có cấu trúc phức tạp hơn, bao gồm nhiều tệp con với kích thước trung bình, thường xuất hiện trong các môi trường doanh nghiệp hoặc lưu trữ tài liệu quan trọng.

Tệp ZIP 3 (1000KB): Mô phỏng trường hợp xử lý dữ liệu khối lượng lớn, đòi hỏi chương trình phải tối ưu hóa khả năng quản lý tài nguyên và đảm bảo hiệu suất trong các điều kiện tải nặng.

Việc kiểm thử được tổ chức một cách có hệ thống nhằm phân tích khả năng của chương trình trong việc đáp ứng các yêu cầu hiệu năng và độ tin cậy. Kết quả kiểm

thử sẽ đóng vai trò là cơ sở để đánh giá và cải thiện chương trình, từ đó đảm bảo rằng nó có thể áp dụng hiệu quả trong các tình huống thực tế, như khôi phục mật khẩu cho các tệp ZIP bị quên hoặc thất lạc.

5.2 Kết quả kiểm thử

Quá trình thực hiện trên không gian mật khẩu bao gồm chữ thường và số, độ dài mật khẩu tối đa 3 ký tự và mật khẩu chính xác được đặt là tổ hợp cuối cùng được sinh. Máy tính thực hiện kiểm thử sử dụng CPU AMD Ryzen 7 5800H 8 Cores 16 Threads. Kết quả kiểm thử được trình bày chi tiết dưới đây.

5.2.1 Kết quả kiểm thử trên tệp ZIP 0 (1KB)

Tệp ZIP 0 được sử dụng để kiểm tra hiệu năng cơ bản và xác minh tính chính xác của chương trình trong điều kiện đơn giản. Bảng dưới đây trình bày thời gian hoàn thành (đơn vị: giây) của các phương pháp Brute Force và Dictionary Attack với số lượng tiến trình khác nhau.

Số tiến trình	BruteForce	DictionaryAttack
1	53.23	52.01
2	43.07	42.14
3	25.02	23.99
4	22.32	21.71
5	17.33	16.53
6	16.47	15.35
7	13.97	12.24
8	12.93	11.80

Bảng 5.1: Thời gian kiểm thử trên tệp ZIP 0

Kết quả cho thấy thời gian thực hiện giảm đáng kể khi tăng số lượng tiến trình, đặc biệt từ 1 đến 4 tiến trình. Tuy nhiên, sau 4 tiến trình, tốc độ cải thiện giảm dần, cho thấy dấu hiệu bão hòa trong hiệu suất đa luồng.

5.2.2 Kết quả kiểm thử trên tệp ZIP 1 (100KB)

Bảng sau đây trình bày thời gian hoàn thành kiểm thử với tệp ZIP 1 chứa một tệp đơn lẻ có dung lượng 100KB.

Số tiến trình	**BruteForce**	**DictionaryAttack**
1	53.25	52.12
4	23.44	21.88
8	14.18	12.28

Bảng 5.2: Thời gian kiểm thử trên tệp ZIP 1

So với tệp ZIP 0, thời gian xử lý của tệp ZIP 1 tăng nhẹ khi chỉ sử dụng một tiến trình. Tuy nhiên, khi tăng số tiến trình, thời gian xử lý giảm mạnh, khẳng định tính hiệu quả của việc áp dụng đa luồng.

5.2.3 Kết quả kiểm thử trên tệp ZIP 2 (200KB, 300KB, 500KB)

Đây là trường hợp tệp ZIP phức tạp hơn, bao gồm ba tệp con có kích thước khác nhau. Kết quả kiểm thử được trình bày dưới đây.

Số tiến trình	BruteForce	DictionaryAttack
1	55.45	54.53
4	26.62	25.51
8	17.51	15.92

Bảng 5.3: Thời gian kiểm thử trên tệp ZIP 2

Dữ liệu cho thấy thời gian xử lý tăng lên đáng kể khi kích thước tổng của tệp ZIP tăng. Tốc độ xử lý được cải thiện đáng kể với đa luồng, nhưng sự bão hòa hiệu suất vẫn xuất hiện ở số lượng lớn tiến trình.

5.2.4 Kết quả kiểm thử trên tệp ZIP 3 (1000KB)

Kịch bản này mô phỏng việc xử lý dữ liệu lớn. Bảng dưới đây ghi nhận thời gian hoàn thành kiểm thử.

Số tiến trình	BruteForce	DictionaryAttack
1	60.43	59.53
4	31.49	30.49
8	22.96	21.95

Bảng 5.4: Thời gian kiểm thử trên tệp ZIP 3

Với tệp ZIP dung lượng lớn, thời gian xử lý tăng đáng kể, đặc biệt khi sử dụng ít tiến trình. Mặc dù vậy, việc tối ưu hóa đa luồng vẫn giúp cải thiện đáng kể hiệu suất, giảm gần ba lần thời gian thực hiện khi sử dụng 8 tiến trình so với 1 tiến trình.

5.3 Đánh giá

5.3.1 Độ chính xác

Chương trình đã đạt độ chính xác cao với cả hai phương pháp trong quá trình kiểm thử. Ở tất cả các kịch bản, chương trình đều tìm ra mật khẩu của tệp ZIP, kể cả với độ phức tạp của mật khẩu tăng lên hoặc danh sách từ điển có dung lượng lớn. Điều này khẳng định tính đúng đắn và hiệu quả trong thiết kế thuật toán.

5.3.2 Tốc độ thực thi và hiệu quả của đa tiến trình

Trong hai chế độ, Brute Force yêu cầu tính toán liên tục để sinh các tổ hợp mới, trong khi Dictionary Attack chỉ cần đọc các mật khẩu sẵn có từ danh sách. Nên điều này làm tăng chi phí xử lý của Brute Force so với Dictionary Attack, đặc biệt với mật khẩu dài và bộ ký tự lớn.

Tốc độ thực thi đã được cải thiện đáng kể khi áp dụng đa tiến trình xử lý song song. Kết quả kiểm thử cho thấy thời gian thực hiện giảm mạnh khi số lượng tiến trình tăng từ 1 đến 4, với tốc độ giảm trung bình gần 50% mỗi lần số tiến trình tăng gấp đôi. Tuy nhiên, khi số tiến trình tăng lên từ 4 đến 8, tốc độ cải thiện giảm dần và xuất hiện hiện tượng bão hòa hiệu suất.

5.3.3 Giải thích hiện tượng bão hòa

Hiện tượng bão hòa xảy ra khi tăng số lượng tiến trình không còn đem lại cải thiện đáng kể về tốc độ thực thi. Nguyên nhân chính bao gồm:

Thực hiện hai vòng kiểm tra: Vòng kiểm tra cuối thực hiện giải mã và giải nén một tệp có kích thước nhỏ nhất trong tệp ZIP nhằm đảm bảo mật khẩu hoàn toàn chính xác. Quá trình này không được áp dụng xử lý song song nên tốc độ không giảm khi tăng số tiến trình sử dụng mà phụ thuộc hoàn toàn vào kích thước của tệp đó.

Chi phí đồng bộ hóa: Khi số tiến trình tăng, chi phí quản lý và đồng bộ hóa giữa các tiến trình cũng tăng, đặc biệt với các hàng đợi chia sẻ trong chương trình.

Tăng độ phức tạp xử lý dữ liệu: Các tiến trình phải liên tục kiểm tra trạng thái ngắt và tạm dừng , điều này làm tăng thêm chi phí tính toán.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Báo cáo đã hoàn thành việc nghiên cứu toàn diện về chương trình tấn công mật khẩu trên các tệp ZIP, từ cơ sở lý thuyết đến triển khai và kiểm thử thực tế. Mở đầu, các phương pháp tấn công mật khẩu như Brute Force và Dictionary Attack được phân tích chi tiết, làm rõ ưu điểm, hạn chế, và tính ứng dụng. Tiếp đó, hệ thống kiểm thử được thiết kế khoa học với các tệp ZIP có kích thước và cấu trúc đa dạng, đảm bảo việc đánh giá toàn diện hiệu năng, độ chính xác và tính ổn định của chương trình.

Kết quả kiểm thử cho thấy chương trình đạt độ chính xác cao, xử lý hiệu quả trong hầu hết các tình huống kiểm thử, đặc biệt khi sử dụng tối ưu hóa đa tiến trình. Thời gian xử lý được rút ngắn đáng kể khi tăng số lượng tiến trình, tuy nhiên hiện tượng bão hòa hiệu suất xuất hiện khi vượt quá 4 tiến trình, phản ánh những hạn chế trong việc tối ưu hóa đồng bộ và xử lý song song. Sự so sánh giữa Brute Force và Dictionary Attack đã làm rõ rằng, trong khi Dictionary Attack hiệu quả hơn về tốc độ, Brute Force vẫn giữ vai trò quan trọng trong việc xử lý các trường hợp không có dữ liệu từ điển phù hợp.

6.2 Hướng phát triển

Dựa trên kết quả kiểm thử, báo cáo đã đưa ra những kết luận quan trọng về khả năng và giới hạn của chương trình tấn công mật khẩu trên các tệp ZIP. Chương trình đã chứng minh tính hiệu quả và độ chính xác cao khi xử lý các kịch bản kiểm thử với nhiều loại tệp ZIP có cấu trúc và kích thước khác nhau. Tuy nhiên, hiện tượng bão hòa hiệu suất khi tăng số lượng tiến trình và thời gian xử lý lớn với các mật khẩu phức tạp đã chỉ ra những điểm cần cải thiện. Để nâng cao hiệu suất và mở rộng khả năng ứng dụng, một số định hướng phát triển quan trọng đã được đề xuất.

Thứ nhất, việc xây dựng chiến lược tối ưu hóa xử lý đa tiến trình là một giải pháp tiềm năng. Bằng cách giảm thiểu chi phí đồng bộ hóa và khai thác triệt để tài nguyên phần cứng, chương trình có thể duy trì hiệu suất cao ngay cả khi xử lý dữ liệu lớn.

Thứ hai, tích hợp GPU vào hệ thống là một hướng đi hứa hẹn nhằm tăng tốc độ xử lý. GPU, với khả năng thực thi hàng ngàn luồng song song, có thể giảm đáng kể thời gian thực hiện, nhất là trong các kịch bản yêu cầu xử lý một không gian khóa rộng lớn. Việc tận dụng sức mạnh tính toán của GPU không chỉ cải thiện tốc độ mà

còn giảm áp lực lên CPU, từ đó tối ưu hóa toàn bộ hệ thống.

Thứ ba, việc tăng cường thuật toán quản lý tài nguyên sẽ giúp giảm thiểu hiện tượng bão hòa hiệu suất, đặc biệt với các kịch bản xử lý dữ liệu lớn hoặc đòi hỏi thời gian dài. Bằng cách tối ưu hóa cách thức phân bổ và sử dụng tài nguyên, chương trình có thể đạt hiệu quả tốt hơn trong việc sử dụng đa tiến trình mà không gặp phải các giới hạn hiện tại.

Cuối cùng, việc nghiên cứu thêm các phương pháp kiểm thử trên các tệp ZIP có cấu trúc phức tạp hơn sẽ giúp đánh giá toàn diện hơn hiệu suất của chương trình. Điều này đảm bảo rằng chương trình không chỉ hiệu quả trong môi trường thử nghiệm mà còn có thể đáp ứng được các yêu cầu thực tế ngày càng đa dạng.

Những định hướng này không chỉ nhằm khắc phục các hạn chế hiện tại mà còn mở ra cơ hội để chương trình trở thành một công cụ mạnh mẽ, linh hoạt và có tính ứng dụng cao trong nhiều tình huống thực tế khác nhau. Qua đó, chương trình có thể tiếp tục phát triển và đáp ứng tốt hơn các yêu cầu của người dùng trong tương lai.

TÀI LIỆU THAM KHẢO

[1] IJC-SMC Editorial Team, “A Survey on the Advanced Encryption Standard (AES): A Pillar of Modern Cryptography," *International Journal of Computer Science and Mobile Computing*, 2024. [Online]. Available: https://ijcsmc.com/docs/papers/April2024/V13I4202418.pdf.

[2] APPNOTE, “.ZIP File Format Specification," 2022. [Online]. Available: https://www.pkware.com/appnote.

[3] Python Software Foundation, “threading — Thread-based parallelism," [Online]. Available: https://docs.python.org/3/library/threading.html. (visited on 01/16/2025).

[4] Python Software Foundation, “multiprocessing — Process-based parallelism," [Online]. Available: https://docs.python.org/3/library/multiprocessing.html. (visited on 01/16/2025).

[5] Python Software Foundation, “tkinter — Python interface to Tcl/Tk," [Online]. Available: https://docs.python.org/3/library/tkinter.html. (visited on 01/16/2025).

[6] Python Enhancement Proposal, “PEP 371 – Addition of the multiprocessing package to the standard library," [Online]. Available: https://peps.python.org/pep-0371/. (visited on 01/16/2025).