ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ASSIGNMENT

MÔN Mật mã và An ninh mạng (CO3069)

ĐỀ TÀI:

Bài tập lớn số 2

GVHD: Nguyễn Cao Đạt

Sinh viên thực hiện: Phan Minh Trọng – 2213676 – L01

Tạ Nguyễn Tiến Dũng – 2110965 – L01

Đồng Mạnh Cường – 2210420 – L01

Thành phố Hồ Chí Minh – 2024

MỤC LỤC

Chương 1: Giới thiệu

1.1 Yêu cầu bài tập lớn

Chủ đề đã chọn: Chủ đề số 2

Đối với bài tập lớn lần này, nhóm đã chọn chủ đề số 2, là một chủ đề liên quan đến một số phương pháp mã hóa, giải mã. Cụ thể nhóm sẽ thực hiện các yêu cầu của bài tập lớn như sau:

- Nghiên cứu về các hệ mã hóa thay thế (Caesar) và mã hóa hoán vị (Rail fence), tìm hiểu cơ chế, cách thức hoạt động của các hệ mã hóa trên.
- Từ những gì nghiên cứu được, nhóm sẽ sử dụng một ngôn ngữ lập trình mà nhóm đã chọn (Python) để hiện thực chương trình giải quyết bài toán liên quan đến việc sử dụng các hệ mã hóa này thông qua các phương thức được yêu cầu:
  - Phương thức mã hóa "plaintext" bằng hệ mã hóa thay thế đơn ký tự.
  - Phương thức mã hóa "plaintext" bằng hệ mã hóa hoán vị.
  - Phương thức mã hóa nhân sử dụng cả 2 phương thức trên.
- Cuối cùng, nhóm sẽ tìm hiểu và thiết kế ra các phương pháp để tìm ra "plaintext" tương ứng với "ciphertext", gồm 2 công việc tuần tự:
  - Cố gắng lấy các "plaintext" từ "ciphertext".

Cố gắng lấy được các khóa "key" đã sử dụng.

1.2 Nội dung chính các chương tiếp theo

Sau khi đã xác định được chủ đề, hiểu rõ luồng hoạt động cần phải hoàn thành trong bài tập lớn lần này, nhóm đã chia rõ và phân luồng cụ thể cho các hoạt động tiếp theo, cụ thể:

- **Chương 2**: **Trong chương này, nhóm tập trung tìm hiểu về cơ sở lý thuyết của các kiến thức liên quan trong bài tập lớn:**

- Kiến thức cơ bản, cốt lõi của việc mã hóa/giải mã bằng các hệ mã hóa; kiến thức nền tảng khi bắt tay làm bất cứ một bài toán liên quan đến lĩnh vực này.
- Các hệ mã hóa được đề cập (hệ mã hóa thay thế đơn ký tự Caesar, hệ mã hóa hoán vị Rail fence); tìm hiểu cách chúng hoạt động, các ưu điểm, nhược điểm của các hệ mã hóa này trong thực tế.
- Nghiên cứu, tìm hiểu về các phương pháp khác khả thi để có thể giải mã khi không có thông tin về khóa cũng như phương pháp hiện thực mã hóa.

- **Chương 3: Trong chương này, nhóm sẽ tập trung vào phân tích và thiết kế các bước để hiện thực bài toán:**
  - Đầu tiên, nhóm bắt tay vào phân tích kỹ hơn yêu cầu của bài toán, chia bài toán thành các module nhỏ hơn và nghiên cứu tuần tự và song song (tuần tự cho từng module và nghiên cứu song song giữa các thành viên).
  - Sau khi phân tích và chia nhỏ theo từng module, nhóm sẽ thiết kế bước giải module đó cho phù hợp với kiến thức.
- **Chương 4: Sau bước phân tích và thiết kế xong, nhóm bắt đầu công việc hiện thực, sau đó là đánh giá:**
  - Sử dụng ngôn ngữ lập trình đã chọn bắt đầu hiện thực cho từng module nhỏ.
  - Việc hiện thực từng module sẽ được thực hiện song song với việc viết báo cáo, làm tới đâu viết tới đó.
  - Song song với đó là nghiên cứu và thiết kế các phương pháp khả thi để truy vấn ngược “plaintext”.
  - Đánh giá các phương thức hiện thực, (performance có tốt không, có đáp ứng được tất cả các bài toán chưa,…)
- **Chương 5: Bước cuối cùng, kết luận:**

- Nhóm trình bày những kết quả mà mình đã hoàn thiện được trong bài tập lớn.
- Trình bày những module chưa làm được, và nêu rõ lý do chưa hoàn thành được.
- Trình bày những điểm thuận lợi, khó khăn khi thực hiện bài tập lớn này.
- Hướng phát triển để có thể hoàn thiện hơn, mở rộng hơn dự án này.
- Hoàn thiện báo cáo

Chương 2: Cơ sở lý thuyết

2.1 Cơ sở lý thuyết cho việc mã hóa, giải mã

Mã hóa và giải mã là hai quá trình cơ bản để tạo ra bảo mật thông tin giữa người dùng, giúp bảo vệ dữ liệu khỏi sự truy cập trái phép. Cụ thể:

- Mã hóa (Encrytion): Là quá trình và chuyển đổi thông điệp gốc (được gọi là "plaintext") có ý nghĩa thành một dạng thông tin khác không có ý nghĩa (được gọi là bản mã – "ciphertext") thông qua một số khóa bí mật. Điều này giúp thông tin được bảo mật trong quá trình truyền tải, chỉ những người có quyền truy cập (có "key") mới có thể truy cập và đọc nội dung gốc (plaintext).
- Giải mã (Decrytion): Là quá trình ngược lại của encryption, nghĩa là quá trình chuyển đổi từ "ciphertext" về "plaintext" thông qua khóa giải mã, giúp người có quyền truy cập đọc được thông tin mà người gửi truyền tải.
- Và đương nhiên, để có thể hiện thực được 2 quá trình trên, phải thông qua các thuật toán mã hóa (hay là hệ mã hóa). Và trong bài tập lớn lần này, chúng ta sẽ tìm hiểu và nghiên cứu rõ hơn về 2 hệ mã hóa được đề cập ở phần tiếp theo.

2.2 Hệ mã hóa thay thế đơn ký tự Caesar

2.2.1 Hệ mã hóa thay thế

Hệ mã hóa thay thế (Substitution Cipher) là một trong những phương pháp mã hóa cổ điển trong lĩnh vực mật mã học. Phương pháp này thay thế mỗi ký tự trong văn bản gốc (plaintext) bằng một ký tự khác từ một bảng chữ cái đã được thay thế (cipher alphabet), tạo thành văn bản mã hóa (ciphertext). Đây là một trong những hệ mã đơn giản nhưng có thể được sử dụng để hiểu về cách thức mã hóa và giải mã dữ liệu

2.2.2 Hệ mã hóa thay thế đơn ký tự Caesar

Hệ mã hóa thay thế đơn ký tự Caesar là một trong những phương pháp mã hóa cổ điển nhất, được đặt theo tên của Julius Caesar, nhà lãnh đạo La Mã, người đã sử dụng hệ thống này để bảo vệ các thông điệp quân sự của mình. Phương pháp này là một dạng đặc biệt của *mã hóa thay thế*, trong đó mỗi ký tự trong văn bản gốc (plaintext) sẽ được thay

thế bằng một ký tự khác trong bảng chữ cái theo một số bước dịch chuyển cố định. Đây là một ví dụ đơn giản nhưng có thể được sử dụng để minh họa các nguyên lý cơ bản của mật mã học.

Một ví dụ đơn giản đối với một mã hóa đơn ký tự Caesar:

Với khóa key = 3 và plaintext letter là A ➔ Mã hóa của letter A là letter A được dịch đi 3 ký tự trên bảng chữ cái alphabet, nghĩa là D. Nếu plaintext dài hơn, chúng ta sẽ thực hiện cho từng ký tự và sẽ ghép lần lượt cipher letter để được ciphetext cuối cùng.

Nhận xét: Rõ ràng với một cách mã hóa thay thế đơn ký tự và quá đơn giản như vậy, dẫn tới việc về bảo mật an toàn của hệ mã hóa không được đánh giá cao, dễ bị tấn công và lấy cắp thông tin.

2.3 Hệ mã hóa hoán vị Rail fence

Hệ mã hóa hoán vị Rail fence: là một mã chuyển vị khá dễ để áp dụng, nó giúp đảo trật tự của các ký tự trong bản rõ một cách nhanh chóng, thuận tiện. Về cơ bản, hệ mã hóa này hoạt động bằng ách viết plaintext theo các dòng thay thế trên trang, sau đó sẽ đọc tuần tự theo các dòng để có được "ciphertext". Cụ thể hơn, để bắt đầu mã hóa một thông điệp, chúng ta sẽ xác định số hàng sẽ có trong trang thông qua khóa "key" (giá trị của key chính là số hàng). Tiếp theo, viết các ký tự trong plaintext theo các đường chéo liên tiếp trên các hàng (zig-zag), điều này tiếp tục cho đến khi viết hết được plaintext trên trang. Trong trường hợp còn dữ các ký tự còn dư và chưa điền hết hàng trên trang theo quy tắc trên, ô đó sẽ được thay thế bằng X (có ý nghĩa là null và hoạt động như một trình giữ chỗ).

Để hiểu rõ hơn về hệ mã hóa này, chúng ta sẽ đi qua một ví dụ:

Yêu cầu: Mã hóa thông điệp "**ALL STUDENTS COMPLETE THE ASSIGMENT ON TIME**" với khóa "key" = 4:

- Đầu tiên chúng ta sẽ xác định plaintext sau khi lược bỏ các ký tự khoảng trắng: **ALLSTUDENTSCOMPLETETHEASSIGMENTONTIME**
- Tiếp theo chúng ta xác định được số hàng khi mã hóa là 4 (vì key = 4), như vậy chúng ta sẽ có trang mã hóa với số hàng là 5.
- Bước kế tiếp, chúng ta sẽ lần lượt điền các ký tự plaintext theo đường chéo cho đến khi điền hết các ký tự trong plaintext:

A	L	L	S	T	U	D	E	N	T	S	C	O	M	P	L	E	T	E	T	H	E	A	S	S	I	G	N	M	E	N	T	O	N	T	I	M	E

A							N								E							S							O				
	L					E		T						L		T						S		I						T		N			
		L				D			S				P				E				A				G				N				T			
			S		U					C		M						T		E						N		E					I		E
				T							O								H							M								M		

- Bước cuối cùng, ta đọc các ký tự theo hàng tuần tự để ra kết quả ciphertext cuối cùng: "**ANESOLETLTSITNLDSPEAGNTSUCMTENEIETOHMM**"

  Nhận xét: Hệ mã hóa Rail Fence mặc dù đã tăng cương được độ mã hóa so với hệ mã hóa thay thế đơn ký tự Cipher, tuy nhiên rõ ràng độ bảo mật vẫn chưa cao:

- Nếu biết được số hàng (số rails sử dụng) và cách thức mã hóa, kẻ xấu có thể truy ra được thông điệp gốc rất dễ dàng. Thậm chí, kẻ tấn công còn có thể thực hiện việc thử nghiệm với tất cả các key trung bình, nhỏ dễ dàng.
- Do đặc trưng của việc hoán vị, kẻ gian dễ dàng tái tạo lị trang giải mã mà không cần biết khóa.

2.4 Các phương pháp sử dụng để cố gắng suy luận khóa

Về cơ bản, rõ ràng việc mã hóa để bảo mật an toàn thông tin truyền tải đi, và nếu người nhận muốn đọc, bắt buộc phải biết được các quy tắc mã hóa cũng như khóa "key". Câu hỏi đặt ra là nếu như không có thông tin về khóa, liệu có cách nào để có thể cố gắng suy luận khóa hoặc các thông tin bảo mật được gửi đi hay không? Câu trả lời là có.
Trong lĩnh vực an ninh mạng, và cụ thể trong việc mã hóa và giải mã, vì các thông điệp gửi đi là có ý nghĩa nên việc có một xác suất nhỏ thành công trong việc phân tích, suy luận khóa. Trong trong các tài liệu nghiên cứu, có một số phương pháp được đề cập tới phương pháp suy luận, cụ thể:

2.4.1 Sử dụng việc phân tích tần số của các ký tự xuất hiện

Theo các phân tích, nghiên cứu của các chuyên gia trong ngôn ngữ tự nhiên, đặc biệt tiếng Anh (là ngôn ngữ phổ biến nhất thế giới), mỗi ký tự trong bảng chữ cái có một tần số xuất hiện. Chính vì thế, có thể lấy bảng tần suất xuất hiện này đối chiếu với các ký tự trong ciphertext để có thể suy luận ngược plaintext.

a	b	c	d	e	f	g	h	i	j	k	l	m
8, 05	1, 62	3, 2	3, 65	12, 31	2, 28	1, 61	5, 14	7, 18	0, 1	0, 52	4, 03	2, 25
n	o	p	q	r	s	t	u	v	w	x	y	z
7, 19	7, 94	2, 29	0, 20	6, 03	6, 59	9, 59	3, 1	0, 93	2, 03	0, 2	1, 88	0, 09

2.4.2 Phân tích cặp ký tự hoặc ba ký tự

"Ngoài việc tần số của các ký tự bảng chữ cái đơn lẻ, việc xác định tần suất gặp lại của các cặp từ hay ba ký tự trong tiếng anh cũng có thể là gợi ý cho việc suy luận mã. Rõ ràng trong các thông điệp bằng tiếng anh, việc lặp lại của cặp ký tự như "th", "er", "in", "on",… hoặc ba ký tự như "the", "ate",… cũng có thể là gợi ý để có thể suy luận khóa."[1]

2.4.3 Phân tích khoảng cách giữa các ký tự lặp lại

"Đây là một phương pháp đặc biệt hữu ích với các mã thay thế polyalphabetic, chẳng hạn mã Vigenère. Phương pháp Kasiski xem xét khoảng cách giữa các chuỗi ký tự lặp lại trong ciphertext. Khoảng cách này có thể giúp suy ra độ dài của khóa. Khi biết độ dài khóa, bạn có thể áp dụng phân tích tần số trên từng phần của ciphertext."[2]

2.4.4. Các kỹ thuật thử khóa brute-force

Đây là một phương pháp khá là phổ biến trong việc suy luận khóa. Bằng cách thử rất nhiều các key khác nhau cho các hệ mã hóa phổ biến, có thể kể tới như các thuật toán Cipher (hệ mã hóa thay thế). Tuy xác suất và tính hiệu quả không hề cao, tuy nhiên cũng là một cách để cố gắng suy luận khóa.

[1] What is a Trigraph? - Trigraph Examples and Definition. Truy cập ngày 15/11/2024, từ https://www.twinkl.com.vn/teaching-wiki/trigraph

[2] Kasiski examination - Wikipedia. Truy cập ngày 15/11/2024, từ https://en.wikipedia.org/wiki/Kasiski_examination

Chương 3: Phân tích và thiết kế

3.1 Hệ mã Caesar

Theo hệ mã caesar, khi ta mã hóa bản rõ, mỗi ký tự trong bản rõ được dịch chuyển theo một số cố định trong bảng kí tự, có thể là ASCII hoặc UTF-8. Tuy rằng có nhiều thuật toán có thể giải mã được hệ mã Caesar như bruteforce, phần tích từ điển,... nhưng nhóm đề xuất sử dụng phân tích tần số để xác định bản rõ vì một số lí do sau:

a) Mỗi ngôn ngữ có phân bố tần suất đặc trưng:

Trong bảng chữ cái tiếng Anh:

a	b	c	d	e	f	g	h	i	j	k	l	m
8, 05	1, 62	3, 2	3, 65	12, 31	2, 28	1, 61	5, 14	7, 18	0, 1	0, 52	4, 03	2, 25
n	o	p	q	r	s	t	u	v	w	x	y	z
7, 19	7, 94	2, 29	0, 20	6, 03	6, 59	9, 59	3, 1	0, 93	2, 03	0, 2	1, 88	0, 09

b) Hệ mã Caesar bảo toàn tần số xuất hiện của ký tự: Vì đây là một phép dịch chuyển đơn giản, tần suất của các ký tự trong bản mã giống hệt với bản rõ, chỉ bị dịch chuyển vị trí.

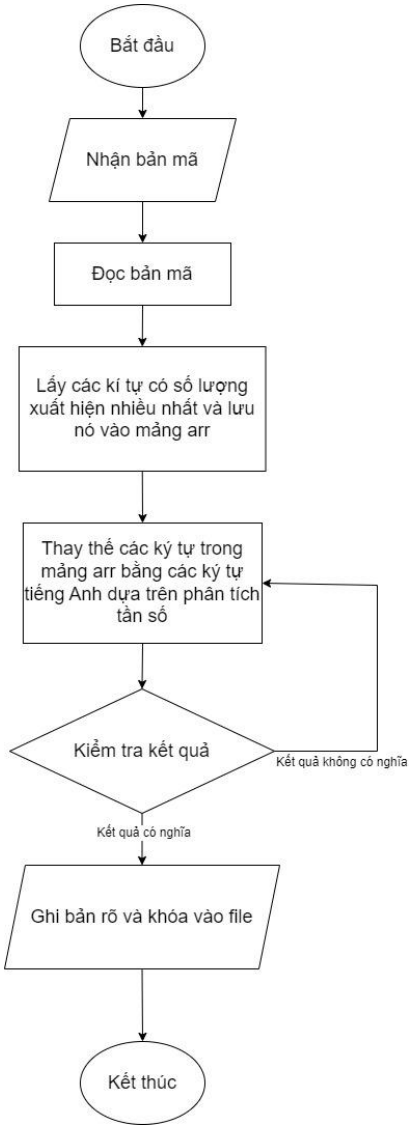
c) Độ dài bản rõ có kích thước lớn: vì độ dài bản rõ lớn nên ta mới có thể đảm bảo được tần số của các kí tự.

d) Số lượng key quá lớn: bruteforce sẽ gây tốn thời gian dựa trên việc thử toàn bộ khóa khả thi.

Phương pháp tấn công này dựa trên việc khai thác đặc điểm tự nhiên của văn bản tiếng Anh, nơi mà các chữ cái không xuất hiện với tần suất đồng đều. Một số chữ cái như E, T, và A thường xuất hiện rất phổ biến, trong khi những chữ cái như Q, X, và Z lại hiếm gặp hơn. Bằng cách phân tích tần suất xuất hiện của các ký tự trong bản mã và so sánh với phân bố tần suất điển hình trong tiếng Anh, ta có thể suy luận ra các mẫu chữ hoặc khóa mã hóa.[3]

[3] Frequency Analysis. Truy cập ngày 20/11/2024, từ https://mathstats.uncg.edu/sites/pauli/112/HTML/secfrequency.html

Sơ đồ thiết kế cho phân tích tần số



3.2 Hệ mã Rail fence

Rail Fence Cipher có ưu điểm là đơn giản, dễ thực hiện, phù hợp để minh họa khái niệm mã hóa cơ bản. Tuy nhiên, hệ mã này có nhiều nhược điểm: không thêm bất kỳ sự phức tạp nào vào nội dung văn bản, dễ bị phá mã bằng cách thử nghiệm (brute force), và có không gian key (keyspace) nhỏ do key bị giới hạn bởi độ dài văn bản.

Rail Fence Cipher không an toàn vì tất cả các ký tự trong văn bản gốc vẫn xuất hiện trong chuỗi mã hóa, chỉ bị hoán vị. Điều này giúp kẻ tấn công dễ dàng nhận diện mẫu zigzag

đặc trưng và khôi phục key. Một tấn công brute force đơn giản, thử lần lượt tất cả các giá trị key khả dĩ, sẽ phá mã thành công. Với chuỗi mã hóa dài n, số lượng key cần thử là n−2, khiến hệ mã dễ bị tấn công ngay cả khi văn bản gốc dài.

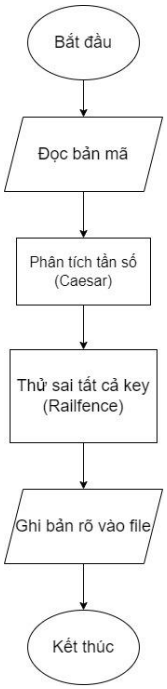
Sơ đồ thiết kế thử sai trong railfence

3.3 Hệ mã nhân (Caesar kết hợp với railfence)

Sự kết hợp mã Caesar và Rail Fence Cipher tạo ra một hệ mã phức tạp hơn, tăng cường độ bảo mật so với việc sử dụng từng hệ mã riêng lẻ. Sự kết hợp này thuộc loại mã hóa đa lớp, trong đó mỗi lớp thực hiện một vai trò khác nhau: mã Caesar thay thế các ký tự dựa trên một phép dịch vòng, trong khi Rail Fence Cipher hoán vị thứ tự của các ký tự.

Hệ mã nhân này cải thiện đáng kể độ phức tạp so với từng hệ mã đơn lẻ, nhưng vẫn không đảm bảo mức độ bảo mật cao trong thực tế. Hệ mã này vẫn dễ bị phá bởi các tấn công brute force kết hợp. Vì vậy, sự kết hợp này phù hợp để minh họa ý tưởng mã hóa đa lớp, nhưng không đủ mạnh để sử dụng trong các ứng dụng bảo mật hiện đại.

Sơ đồ thiết kế bẻ khóa hệ mã nhân (Caesar kết hợp với railfence)



Chương 4: Hiện thực và đánh giá

4.1 Giải mã Caesar

4.1.1 Hiện thực

Cấu trúc mã nguồn

- **Hàm condition(c)**

- Kiểm tra xem một ký tự có hợp lệ không (bao gồm chữ cái, khoảng trắng và các ký tự dấu câu cơ bản).

- **Hàm decrypt_caesar (file_input, file_output)**

- Đọc file_input để lấy ciphertext
- Ghi kết quả vào file_output
- Set start_time và end_time để tính toán thời gian thực thi

- **Hàm decrypt_caesar_help(ciphertext)**

- Tính tần suất ký tự trong văn bản và lấy danh sách các kí tự có tần suất cao nhất.
- So sánh với tần suất chữ cái tiếng Anh để xác định key.
- Giải mã văn bản bằng cách thử các key khả thi và kiểm tra kết quả.
- Nếu có bất kì kí tự đặc biệt nào xuất hiện thì loại bỏ key đó ngay lập tức.
- Vì bảng chữ cái tiếng Anh bao gồm cả viết hoa, viết thường nên cần xử lý các trường hợp có sự lẫn lộn hai kiểu viết sao cho plaintext trả về đúng cả chữ hoa, chữ thường.

Mã nguồn triển khai

```python
def __init__(self):
    self.valid_chars = {' ', '.', ',', '?', ':', '!', ';', '"', '(', ')', "'", '_', '@', '#', '$', '+', '-', '=', '*', '/', '%', '\n'}

def condition(self, c:chr):
    return c.isalpha() or c.isdigit() or c in self.valid_chars

def decrypt_caesar_help(self, ciphertext):
    english_frequency = [' ', 'e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f', 'g', 'y', 'p', 'b', 'v', 'k', 'j', 'x', 'q', 'z']

    counter = Counter(ciphertext)

    max_frequency = max(counter.values())

    highest_frequency_chars = [char for char, freq in counter.items() if freq == max_frequency]
    processed_keys = set()
    results = []
    decrypted = ""
    for i in (english_frequency):
        for j in highest_frequency_chars:
            decrypted = ""
            key = (ord(j) - ord(i)) % 0x110000
            flag = True
            char = 0
            while char < len(ciphertext):
                new_char_1 = chr((ord(ciphertext[char]) - key) % 0x110000)
                if not (self.condition(new_char_1)):
                    new_char_2 = chr((ord(new_char_1)+32) % 0x110000)
                    new_char_3 = chr((ord(new_char_1)-32) % 0x110000)
                    if (self.condition(new_char_2) and flag):
                        flag = False
                        key = (key - 32) % 0x110000
                        char = 0
                    elif (self.condition(new_char_3) and flag):
                        flag = False
                        key = (key + 32) % 0x110000
                        char = 0
                    else: break
                    decrypted=""
                else:
                    decrypted+=new_char_1
                    char += 1
            if(len(decrypted)==len(ciphertext)):
                if key not in processed_keys:
                    processed_keys.add(key)  # Đánh dấu key đã xử lý
                    results.append((key, decrypted))
    return results
```

4.1.2 Kết quả thử nghiệm

```
PS D:\Monhoc\MMANM\BTL> python Decrypt.py
Thời gian chạy: 0.000559 giây
Ghi kết quả vào ./result_1.txt thành công!
Tất cả các phép tính đã hoàn thành!
PS D:\Monhoc\MMANM\BTL>
```

Dữ liệu đầu vào: File

input.txt

```
Pk|*pk|*k□k□6*1orsxn*~ro*□y|n*wy□x~ksx|6*pk|*p|yw*~ro*my□x~|so|*`yukvsk*kxn*Myx|yxkx~sk6*~ro|o*vs□o*~ro*1vsxn*~o□~|8*|ozk|k~on*~ro□*vs□o*sx*Lyyuwk|u|q|y□o*|sqr~*k~*~ro*myk|~*yp*~ro*|owkx~sm|6*k*vk|qo8
```

Kết quả đầu ra: File result_1.txt

Key: 10, Decrypted: Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large.

Hiệu suất: Với ciphertext dài khoảng 200 kí tự thì chương trình chạy khoảng 0.000559 giây.

Với chuỗi ciphertext dài khoảng 4400 kí tự thì chương trình chạy khoảng 0.009377 giây. Đối với ciphertext lớn hơn thì hiệu suất giảm dần nên nếu được thì cần tối ưu thêm.

4.1.3 Đánh giá giải pháp

Ưu điểm

- Phương pháp dựa trên phân tích tần suất ký tự hiệu quả với các văn bản dài hoặc tuân theo ngữ pháp tiếng Anh thông thường.
- Dễ triển khai, có khả năng thử nhiều key nhanh chóng.

Nhược điểm

- Không hoạt động tốt với ciphertext ngắn (không đủ thông tin tần suất ký tự).
- Hiệu suất chưa tối ưu với dữ liệu rất lớn.

Giới hạn

- Nếu văn bản mã hóa không phải tiếng Anh hoặc chứa nhiều ký tự đặc biệt, kết quả có thể không chính xác vì chỉ chấp nhận một số kí tự phổ biến trong tiếng Anh.
- Phụ thuộc vào chất lượng của danh sách english_frequency.

Tỉ lệ giải mã thành công: Khoảng 70 – 80%

4.2 Giải mã Rail fence

4.2.1 Hiện thực

Cấu trúc mã nguồn

- **Hàm decryptRailFence(file_input, file_output)**

- Đọc file_input để lấy ciphertext
- Set start_time và end_time để tính toán thời gian thực thi

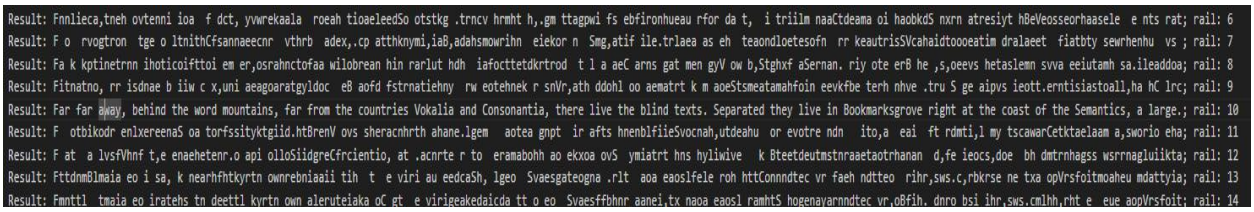
- **Hàm decryptRailFence_help(ciphertext, file_output, keyofCaesar)**

- keyofCaesar chỉ sử dụng khi áp dụng mã nhân
- Khởi tạo ma trận rail: Một ma trận key x len(ciphertext) được tạo ra để điền ký tự vào.
- Điền ký tự vào ma trận theo cấu trúc zigzag: Duyệt qua các ký tự của văn bản mã hóa và điền chúng vào ma trận theo hình zigzag.
- Đọc ký tự từ ma trận: Đọc các ký tự từ ma trận theo đường chéo để giải mã.
- Ghi kết quả vào file_output

Mã nguồn triển khai

```python
def decryptRailFence_help(ciphertext, file_output, keyofCaesar):
    with open(file_output, 'w', encoding='utf-8') as f_out:
        for key in range(2, len(ciphertext)):
            rail = [['\n' for i in range(len(ciphertext))] for j in range(key)]

            dir_down = None
            row, col = 0, 0

            for i in range(len(ciphertext)):
                if row == 0:
                    dir_down = True
                if row == key - 1:
                    dir_down = False

                rail[row][col] = '\0'
                col += 1

                if dir_down:
                    row += 1
                else:
                    row -= 1

            index = 0
            for i in range(key):
                for j in range(len(ciphertext)):
                    if ((rail[i][j] == '\0') and
                    (index < len(ciphertext))):
                        rail[i][j] = ciphertext[index]
                        index += 1

            result = []
            row, col = 0, 0
            for i in range(len(ciphertext)):

                if row == 0:
                    dir_down = True
                if row == key-1:
                    dir_down = False

                if (rail[row][col] != '\0'):
                    result.append(rail[row][col])
                    col += 1

                if dir_down:
                    row += 1
                else:
                    row -= 1
            if(keyofCaesar!=-1):
                f_out.write(f'Result: {"".join(result)}; rail: {key}; key: {keyofCaesar}\n')
            else:
                #print(result)
                f_out.write(f'Result: {"".join(result)}; rail: {key}\n')
    print(f'Ghi kết quả vào {file_output} thành công!')
```

4.2.2 Kết quả thử nghiệm



Dữ liệu đầu vào: File input.txt

Ciphertext: Fnicahd mt eaidan o nten dtkaa eSg.rh tseuad ritehor ther etn,hni ,eleteokthtmafbhu ttlCa bxayBshe ala eof raoil tr gg fn r, mamiknties alnricota yw roeosnvh.piioro i aaod rsVoaet ev v atc,wrf n Seess

Kết quả đầu ra: File result_2.txt

Rail: 10, Decrypted: Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large.

```
PS D:\Monhoc\MMANM\BTL> python Decrypt.py
Ghi kết quả vào ./result_2.txt thành công!
Thời gian chạy: 0.159001 giây
Tất cả các phép tính đã hoàn thành!
```

Hiệu suất: Với ciphertext dài khoảng 200 kí tự thì chương trình chạy khoảng 0.159001 giây. Mất khá nhiều thời gian so với caesar một phần là do phải kiểm tra và phải ghi toàn bộ trường hợp rail vào file.

4.2.3 Đánh giá giải pháp

Ưu điểm

- Phương pháp brute force dễ hiểu và dễ triển khai.
- Tỉ lệ thành công cao.

Nhược điểm

- Khó khăn với văn bản dài và phức tạp.
- Có thể có nhiều key phải thử để tìm ra plaintext.

- Mất thời gian.

Giới hạn

- Khả năng xử lý Unicode bị hạn chế.

Tỉ lệ giải mã thành công: Khoảng 70 – 80%

4.3 Mã hóa Nhân (Caesar rồi đến rail fence)

4.3.1 Hiện thực

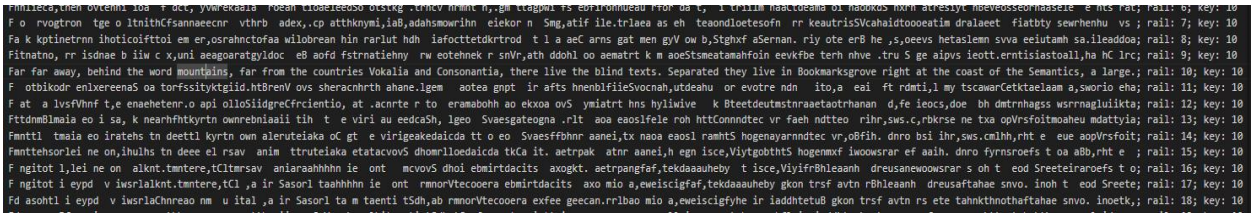
Cấu trúc mã nguồn

- **Hàm decryptMultiple(file_input, file_output)**
- Sử dụng lại 2 hàm giải mã Caesar và rail fence
- Đọc file_input để lấy ciphertext
- Cho chạy giải mã Caesar trước để giới hạn số lượng khóa khả thi, chỉ đưa về các trường hợp có kí tự tiếng Anh.
- Cho chạy giải mã Rail fence tương ứng với từng trường hợp Caesar đã giải mã.
- Vì Rail fence không làm ảnh hưởng biến đổi kí tự mà chỉ thay đổi vị trí kí tự nên ưu tiên giải mã Caesar trước để rút ngắn trường hợp chạy brute force.
- Set start_time và end_time để tính toán thời gian thực thi
- Ghi kết quả vào file_output

Mã nguồn triển khai

```
def decryptMultiple(file_input, file_output):
    start_time = time.time()
    with open(file_input, 'r', encoding='utf-8') as f_in:
        ciphertext = f_in.read()
    results=decrypt_caesar_help(ciphertext)
    for key, decrypted in results:
        decryptRailFence_help(decrypted, file_output, key)
    end_time = time.time()  # Ghi lại thời gian kết thúc
    elapsed_time = end_time - start_time  # Tính thời gian chạy
    print(f"Thời gian chạy: {elapsed_time:.6f} giây")
```

4.3.2 Kết quả thử nghiệm



Dữ liệu đầu vào: File input.txt

Kết quả đầu ra: File result_3.txt

Result: Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large.; **rail:** 10; **key:** 10

```
PS D:\Monhoc\MMANM\BTL> python Decrypt.py
Ghi kết quả vào ./result_3.txt thành công!
Thời gian chạy: 0.156828 giây
Tất cả các phép tính đã hoàn thành!
```

Hiệu suất: Với ciphertext dài khoảng 200 kí tự thì chương trình chạy khoảng 0. 156828 giây. Thời gian tương đương với chạy rail fence từ đó cho thấy rail fence quyết định rất nhiều đến quá trình thực thi, muốn giảm thời gian thực thi cần tối ưu thuật toán giải mã rail fence.

4.3.3 Đánh giá giải pháp

Ưu điểm

- Phương pháp brute force dễ hiểu và dễ triển khai.
- Tỉ lệ thành công cao.

Nhược điểm

- Khó khăn với văn bản dài và phức tạp.
- Mất thời gian để xác định khóa đã sử dụng ở cả 2 thuật toán
- Có thể có nhiều key phải thử để tìm ra plaintext.
- Mất thời gian thực thi.

Tỉ lệ giải mã thành công: Khoảng 70 – 80%

Chương 5: Kết luận

5.1 Những điểm đạt được

- Xây dựng thành công phương pháp giải mã không cần khóa cho cả ba phương thức Caesar, Rail fence và mã hóa nhân.

- Tỉ lệ giải mã thành công khá cao đạt khoảng **70–80%**

5.2 Những điểm chưa làm được

- Chỉ chấp nhận một số kí tự đặc biệt trong tiếng Anh, nếu bản rõ xuất hiện kí tự lạ ngay từ đầu thì gần như không giải mã được

- Chưa tối ưu hóa thời gian chạy cho các trường hợp văn bản rất dài hoặc số lượng khóa Rail Fence lớn.

5.3 Ưu điểm

- Hiệu quả cao trên các văn bản không quá dài và có cấu trúc rõ ràng.

- Các thuật toán sử dụng dễ dàng triển khai

5.4 Nhược điểm

- Hiệu quả chưa ổn định đối với các bản rõ chứa các kí tự quá đặc biệt

- Thời gian xử lý tăng đáng kể khi văn bản hoặc số khóa cần kiểm tra trở nên lớn.

- Chỉ xử lý được các phương pháp mã hóa cụ thể như Caesar, Rail fence và không thể áp dụng cho các dạng mã hóa phức tạp hơn.

5.5 Hướng phát triển

- Tăng tỉ lệ giải mã thành công với cả các trường hợp đặc biệt trong phương pháp giải mã Caesar

- Xây dựng thuật toán giải mã cho rail fence giúp giảm bớt rail cần thử

- Tối ưu hóa hiệu suất, áp dụng các thuật toán song song hoặc cải thiện việc xử lý các trường hợp văn bản dài.

Tài liệu tham khảo

[1].“Cryptography and Network Security”, William Stallings, 6th Edition, Prentice Hall, 2013

[2]. Frequency Analysis. Truy cập tại: nguồn

[3]. Kasiski examination - Wikipedia. Truy cập tại: nguồn

[4]. Rail Fence Cipher – Encryption and Decryption. Truy cập tại: nguồn

[5]. What is a Trigraph? - Trigraph Examples and Definition. Truy cập tại: nguồn

PHỤ LỤC

Phụ lục 1: Thông tin tài liệu tham khảo

Tài liệu lý thuyết:

- "Cryptography and Network Security" của William Stallings, cung cấp cơ sở lý thuyết về các thuật toán mã hóa và giải mã.

Nguồn tham khảo trực tuyến:

- Hướng dẫn giải mã Caesar bằng Frequency Analysis
- Hướng dẫn giải mã Rail Fence Cipher từ GeeksforGeeks.

Phụ lục 2: Môi trường phát triển ứng dụng

Hệ điều hành

- Windows 11

Ngôn ngữ lập trình

- Python 3.13

Công cụ phát triển:

- Visual Studio Code: Cài đặt các tiện ích mở rộng như Python Extension.

Quản lý thư viện:

- pip: Dùng để cài đặt các thư viện Python cần thiết

Phụ lục 3: Thư viện và mã nguồn mở

Thư viện Python sử dụng

- collections.Counter: Thống kê tần suất xuất hiện của ký tự trong chuỗi.
- time: Đo thời gian chạy của thuật toán.
- threading: hỗ trợ lập trình đa luồng (multithreading)