BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH JAVA

CÀI ĐẶT THUẬT TOÁN MÃ HÓA AES, HÀM BĂM SHA-256 VÀ ỨNG DỤNG

Sinh viên thực hiện: **Trần Mai Ngọc Duy**

Mã số sinh viên: **65130650**

Giảng viên hướng dẫn: **TS.Phạm Văn Nam**

Học kỳ 2, năm học 2024-2025

Khánh Hoà - 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH JAVA

CÀI ĐẶT THUẬT TOÁN MÃ HÓA AES, HÀM BĂM SHA-256 VÀ ỨNG DỤNG

Sinh viên thực hiện: **Trần Mai Ngọc Duy**

Mã số sinh viên: **65130650**

Giảng viên hướng dẫn: **TS.Phạm Văn Nam**

Học kỳ 2, năm học 2024-2025

Khánh Hoà - 2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG I. GIỚI THIỆU ĐỀ TÀI

1.1. LÝ DO CHỌN ĐỀ TÀI

Là một sinh viên năm hai ngành công nghệ thông tin, em nhận thức rõ tầm quan trọng của bảo mật dữ liệu trong thời đại số hóa hiện nay. Những sự cố rò rỉ thông tin, tấn công mạng, và đánh cắp dữ liệu ngày càng phổ biến, gây ra những hậu quả nghiêm trọng không chỉ đối với cá nhân mà còn cho các tổ chức, doanh nghiệp lớn. Chính vì vậy, em thấy việc nghiên cứu về các thuật toán bảo mật như AES và SHA-256 không chỉ cần thiết mà còn rất thú vị. Đây là cơ hội để em khám phá sâu hơn lĩnh vực bảo mật thông tin, một trong những mảng công nghệ thông tin đang phát triển mạnh mẽ.

Thuật toán AES đóng vai trò quan trọng trong việc mã hóa dữ liệu, đảm bảo rằng thông tin được bảo vệ an toàn trước khi truyền tải hoặc lưu trữ. Trong khi đó, hàm băm SHA-256 giúp bảo toàn tính toàn vẹn của dữ liệu, đảm bảo rằng dữ liệu không bị chỉnh sửa hoặc giả mạo. Việc kết hợp hai thuật toán này tạo ra một giải pháp mạnh mẽ cho nhiều ứng dụng bảo mật thực tế như bảo mật giao tiếp mạng, lưu trữ thông tin nhạy cảm, và kiểm tra tính toàn vẹn của file.

Em chọn đề tài này vì nó không chỉ cho phép em áp dụng lý thuyết đã học vào thực tế mà còn giúp em phát triển kỹ năng lập trình và tư duy giải quyết vấn đề. Qua việc triển khai các thuật toán và xây dựng ứng dụng, em có cơ hội rèn luyện khả năng làm việc với các công nghệ bảo mật hiện đại. Hơn nữa, em tin rằng đây là một bước đệm quan trọng để em tiến xa hơn trong việc theo đuổi sự nghiệp liên quan đến lĩnh vực an ninh mạng.

Ngoài ra, em cũng rất hào hứng với việc khám phá các ứng dụng thực tế của AES và SHA-256, chẳng hạn như bảo mật giao dịch tài chính, xác thực dữ liệu trong blockchain, và bảo vệ dữ liệu cá nhân. Em tin rằng những kiến thức và kỹ năng tích lũy được trong quá trình thực hiện đề tài này sẽ không chỉ hỗ trợ tốt cho việc học tập hiện tại mà còn là nền tảng vững chắc cho tương lai.

Với những lý do này, em mong muốn thông qua đề tài “Cài đặt thuật toán mã hóa AES, hàm băm SHA-256 và ứng dụng”, em có thể nâng cao kiến thức, thử thách bản thân, và chuẩn bị hành trang để tiến xa hơn trong ngành công nghệ thông tin.

1.2. MỤC TIÊU NGHIÊN CỨU

1.2.1. Mục tiêu đề tài

Trong quá trình thực hiện đề tài, nhóm đã tiến hành cài đặt và triển khai thành công thuật toán mã hóa AES và hàm băm SHA-256 bằng ngôn ngữ lập trình Java, đảm bảo quá trình mã hóa và băm dữ liệu diễn ra chính xác và hiệu quả. Đồng thời, nhóm cũng tập trung nghiên cứu chuyên sâu về nguyên lý hoạt động, cấu trúc và cách thức xử lý dữ liệu của từng thuật toán nhằm hiểu rõ cơ chế bảo mật mà chúng cung cấp. Trên cơ sở đó, một ứng dụng minh họa đơn giản với giao diện trực quan đã được xây dựng, giúp người dùng dễ dàng quan sát cách các thuật toán được áp dụng trong việc mã hóa và kiểm tra tính toàn vẹn của dữ liệu. Cuối cùng, hiệu suất của các thuật toán cũng được đánh giá thông qua việc đo lường tốc độ xử lý và phân tích mức độ an toàn, góp phần khẳng định tính ứng dụng thực tiễn của AES và SHA-256 trong các hệ thống bảo mật thông tin.

1.2.2. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Thuật toán mã hóa AES, hàm băm SHA-256, và ứng dụng của chúng trong việc bảo mật dữ liệu và đảm bảo tính toàn vẹn thông tin.

Phạm vi nghiên cứu: Việc tìm hiểu lý thuyết, cài đặt và minh họa thuật toán mã hóa AES, hàm băm SHA-256. Đề tài sẽ triển khai trong môi trường lập trình Java và xây dựng ứng dụng minh họa nhỏ, với phạm vi dữ liệu đầu vào và đầu ra cơ bản, không đi sâu vào các hệ thống lớn hoặc ứng dụng phức tạp.

1.2.3. Công nghệ sử dụng

Ngôn ngữ lập trình: Java

Thư viện hỗ trợ GUI: java.swing

Công cụ lập trình: IntelliJ IDEA

CHƯƠNG II. CƠ SỞ LÝ THUYẾT

2.1. MÃ HÓA ĐỐI XỨNG AES

2.1.1. Giới thiệu

Mã hóa AES (Advanced Encryption Standard) là một thuật toán mã hóa đối xứng được tiêu chuẩn hóa bởi Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) vào năm 2001. AES được phát triển dựa trên thuật toán Rijndael do hai nhà mật mã học người Bỉ là Vincent Rijmen và Joan Daemen đề xuất.

Với đặc điểm mã hóa đối xứng, AES sử dụng cùng một khóa cho cả quá trình mã hóa và giải mã. Điều này giúp cho AES có tốc độ xử lý nhanh, hiệu quả cao và phù hợp với nhiều ứng dụng thực tế như bảo mật dữ liệu, truyền thông an toàn, và các hệ thống nhúng.

AES hiện là một trong những chuẩn mã hóa phổ biến và an toàn nhất, được sử dụng rộng rãi trong các hệ thống chính phủ, doanh nghiệp và cá nhân trên toàn cầu. Thuật toán này hỗ trợ các độ dài khóa 128 bit, 192 bit và 256 bit, tương ứng với số vòng lặp xử lý là 10, 12 và 14 vòng.

Nhờ vào cấu trúc linh hoạt, khả năng bảo mật cao, cùng với khả năng chống lại hầu hết các phương pháp tấn công mật mã hiện đại, AES đã thay thế thuật toán DES (Data Encryption Standard) trước đó và trở thành tiêu chuẩn mã hóa dữ liệu chính thức tại nhiều tổ chức.

2.1.2. Nguyên lý hoạt động

Thuật toán AES hoạt động dựa trên cơ chế mã hóa khối (block cipher), trong đó dữ liệu đầu vào được chia thành các khối có độ dài cố định là 128 bit (tương đương 16 byte). Quá trình mã hóa diễn ra thông qua nhiều vòng lặp (round), mỗi vòng bao gồm một tập hợp các phép biến đổi toán học nhằm đảm bảo độ an toàn và tính không thể đảo ngược của dữ liệu.

Độ dài khóa và số vòng lặp:

AES-128: Khóa dài 128 bit, có 10 vòng lặp.

AES-192: Khóa dài 192 bit, có 12 vòng lặp.

AES-256: Khóa dài 256 bit, có 14 vòng lặp.

Các bước xử lý trong một vòng lặp AES

Mỗi vòng lặp (trừ vòng đầu và vòng cuối) gồm bốn bước chính:

Bước 1: SubBytes

Mỗi byte trong khối dữ liệu được thay thế bằng một byte khác dựa vào bảng S-box phi tuyến. Bước này tạo ra tính phi tuyến cho thuật toán.

Bước 2: ShiftRows

Các hàng trong ma trận dữ liệu được dịch vòng sang trái:

Hàng thứ 1: giữ nguyên.

Hàng thứ 2: dịch trái 1 byte.

Hàng thứ 3: dịch trái 2 byte.

Hàng thứ 4: dịch trái 3 byte.

Bước 3: MixColumns

Mỗi cột được biến đổi thông qua phép nhân ma trận trong trường hữu hạn GF(28). Bước này giúp trộn dữ liệu giữa các byte trong một cột.

Bước 4: AddRoundKey

Kết quả sau ba bước trên sẽ được XOR với khóa con tương ứng. Các khóa con này được sinh ra từ khóa chính nhờ vào quá trình mở rộng khóa (Key Expansion).

Quy trình tổng thể mã hóa thuật toán AES:

Vòng khởi tạo: AddRoundKey

Các vòng chính: SubBytes → ShiftRows → MixColumns → AddRoundKey

Vòng cuối cùng: SubBytes → ShiftRows → AddRoundKey (Không có bước Mixcolumns trong vòng cuối)

Giải mã trong thuật toán AES:

Quá trình giải mã của AES bao gồm các bước ngược lại:

InvShiftRows

InvSubBytes

InvMixColumns

AddRoundKey

Các khóa con được sử dụng theo thứ tự ngược lại so với quá trình mã hóa.

2.1.3. Cấu trúc thuật toán AES

Thuật toán AES hoạt động trên một cấu trúc ma trận gọi là State, kích thước 4x4 byte (tương đương 128 bit). Dữ liệu đầu vào (plaintext) được chia thành các khối 128 bit, và mỗi khối sẽ được biểu diễn dưới dạng ma trận State để thực hiện các phép biến đổi qua các vòng lặp.

Cấu trúc khối dữ liệu

Mỗi khối dữ liệu đầu vào được tổ chức như sau:

$$\text{State} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

Hình 2.1. Ma trận State trong thuật toán AES

Trong đó Si,j là một byte (8 bit) tại vị trí hàng i, cột j trong ma trận.

Quá trình mở rộng khóa (Key Expansion)

Từ khóa ban đầu, thuật toán AES sinh ra nhiều khóa con (round key) để sử dụng cho từng vòng mã hóa. Tổng số khóa con cần tạo ra tùy vào độ dài khóa:

AES-128: 10 + 1 = 11 round keys

AES-192: 12 + 1 = 13 round keys

AES-256: 14 + 1 = 15 round keys

Quá trình mở rộng khóa bao gồm các bước:

Lấy khóa gốc chia thành các từ (word) 32 bit.

Dựa vào các từ trước đó và hàm SubWord, RotWord, kết hợp với hằng số Rcon để tạo ra từ mới.

Mỗi khóa con là tập hợp 4 từ (tổng 128 bit).

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Hình 2.2. Bảng tra cứu 256 giá trị trong bước SubBytes

S-Box (Substitution Box) là một bảng tra cứu gồm 256 giá trị, tương ứng với tất cả các giá trị có thể của 1 byte (từ 0x00 đến 0xFF).

Bảng này được sử dụng trong bước SubBytes() – một trong các bước biến đổi chính của thuật toán AES.

Mỗi byte trong ma trận trạng thái (State) sẽ được thay thế bởi một giá trị tương ứng trong bảng S-Box. Quá trình này tạo ra tính phi tuyến cho thuật toán, giúp chống lại các cuộc tấn công như phân tích tuyến tính và vi sai.

S-Box được xây dựng từ phép nghịch đảo trong trường hữu hạn GF(28), kết hợp với phép biến đổi tuyến tính (Affine Transformation).

- Inverse S-box used in the **InvSubBytes()** transformation

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Hình 2.3. Bảng ngược của S-Box

Inverse S-Box là bảng đảo ngược của S-Box, chứa các giá trị được sử dụng để hoàn tác quá trình SubBytes trong giai đoạn giải mã.

Được sử dụng trong bước InvSubBytes(), khi mỗi byte của bản mã được thay thế bằng giá trị tương ứng trong bảng Inverse S-Box.

Bảng này giúp khôi phục lại dữ liệu gốc từ bản mã trong quá trình giải mã AES.

2.1.4. Độ an toàn của thuật toán AES

Thuật toán AES (Advanced Encryption Standard) được đánh giá là một trong những thuật toán mã hóa đối xứng an toàn và hiệu quả nhất hiện nay. Một trong những yếu tố cốt lõi tạo nên độ an toàn của AES là kích thước khóa lớn, với ba mức độ tương ứng là 128 bit, 192 bit và 256 bit. Việc thử tất cả các khóa khả thi (brute-force) với AES-128 đã là bất khả thi với công nghệ hiện tại, bởi số lượng khóa có thể lên đến 2^{128}, tức khoảng 3.4×10^{38} khóa. Với AES-256, độ an toàn còn tăng lên gấp nhiều lần với không gian khóa là 2^{256}.

Bên cạnh đó, thiết kế của AES theo mô hình mạng hoán vị-thay thế (Substitution–Permutation Network) giúp tăng cường độ phức tạp và khả năng khuếch tán dữ liệu. Các bước xử lý như SubBytes, ShiftRows, MixColumns và AddRoundKey phối hợp chặt chẽ để đảm bảo rằng chỉ một thay đổi nhỏ ở đầu vào cũng làm thay đổi toàn bộ kết quả mã hóa (hiệu ứng avalanche). Đặc biệt, bảng S-box trong AES được thiết kế dựa trên các phép biến đổi phi tuyến học nhằm tăng khả năng kháng cự trước các dạng tấn công vi sai (differential cryptanalysis) và tấn công tuyến tính (linear cryptanalysis).

AES cũng đã được phân tích sâu rộng trong cộng đồng mật mã học và chưa phát hiện lỗ hổng nghiêm trọng nào tính đến thời điểm hiện tại. Thuật toán này đã được Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST) chuẩn hóa vào năm 2001 và hiện đang được sử dụng rộng rãi trong các lĩnh vực yêu cầu mức độ bảo mật cao như chính phủ, quốc phòng, ngân hàng và hệ thống thông tin công nghiệp.

Với không gian khóa khổng lồ, thiết kế bảo mật chặt chẽ và sự kiểm định rộng rãi từ các tổ chức uy tín, AES vẫn là một lựa chọn hàng đầu trong việc bảo vệ dữ liệu trước các mối đe dọa hiện đại.

2.2. THUẬT TOÁN BĂM SHA-256

2.2.1. Giới thiệu

SHA-256 (Secure Hash Algorithm 256-bit) là một thuật toán băm mật mã thuộc họ SHA-2, được phát triển bởi Cơ quan An ninh Quốc gia Hoa Kỳ (NSA) và được Viện

Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) công bố như là một phần của tiêu chuẩn FIPS PUB 180-4. Thuật toán SHA-256 được thiết kế để tạo ra một giá trị băm (digest) cố định có độ dài 256 bit từ một thông điệp đầu vào có độ dài tùy ý. Đầu ra này được gọi là "mã băm" hoặc "giá trị băm", đóng vai trò như một "dấu vân tay" số của dữ liệu gốc.

Hàm băm SHA-256 đóng vai trò quan trọng trong nhiều lĩnh vực của an ninh mạng và mật mã học hiện đại như xác thực dữ liệu, lưu trữ mật khẩu an toàn, kiểm tra toàn vẹn tệp tin, chữ ký số, giao thức SSL/TLS và đặc biệt là trong công nghệ blockchain. Điều quan trọng cần lưu ý là SHA-256 là một hàm băm một chiều, nghĩa là rất dễ để tính giá trị băm từ dữ liệu đầu vào, nhưng gần như không thể khôi phục được dữ liệu gốc từ giá trị băm đó. Ngoài ra, xác suất để hai thông điệp khác nhau cho cùng một giá trị băm là cực kỳ thấp (gọi là khả năng chống va chạm).

Một số đặc tính nổi bật của SHA-256 có thể kể đến:

Tính xác định: Cùng một đầu vào luôn cho ra một đầu ra giống nhau.

Khó đảo ngược: Không thể tìm được thông điệp ban đầu từ giá trị băm.

Chống va chạm: Xác suất hai đầu vào khác nhau tạo ra cùng một giá trị băm là cực kỳ nhỏ.

Khả năng khuếch đại: Chỉ cần thay đổi một bit trong đầu vào thì đầu ra thay đổi hoàn toàn, khiến việc dự đoán rất khó khăn.

SHA-256 sử dụng cấu trúc Merkle–Damgård và dựa trên nguyên lý hàm nén (compression function) để xử lý dữ liệu theo từng khối 512 bit. Sau mỗi vòng xử lý, dữ liệu được biến đổi thông qua một loạt các phép toán logic như dịch vòng, phép XOR, AND, cộng modulo 232, cùng với các hằng số được định nghĩa trước.

Hiện nay, SHA-256 vẫn được xem là an toàn trước hầu hết các kỹ thuật tấn công mật mã đã biết, và là lựa chọn tiêu chuẩn trong các hệ thống đòi hỏi bảo mật cao. Tuy nhiên, với sự phát triển nhanh chóng của công nghệ tính toán và các tiến bộ trong lĩnh vực điện toán lượng tử, việc nghiên cứu và chuyển sang các thuật toán băm thế hệ tiếp theo như SHA-3 cũng đang được quan tâm.

2.2.2. Nguyên lý hoạt động của thuật toán SHA-256

SHA-256 hoạt động dựa trên 2 bước là xử lý dữ liệu đầu vào và băm dữ liệu. Một số kí hiệu được sử dụng ở phần này bao gồm:

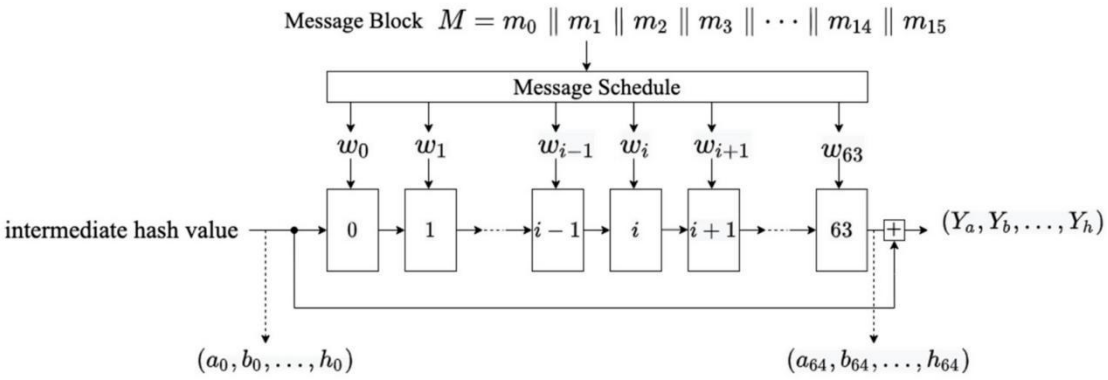
$\oplus$ biểu diễn cho toán tử XOR.

Λ biểu diễn cho toán tử AND.

¬ biểu diễn cho toán tử NOT.

Phép cộng ở đây là cộng nhị phân modulo 232.

Xử lý dữ liệu đầu vào



Hình 2.4. Quá trình xử lý dữ liệu đầu vào của SHA-256

Ban đầu, 16 từ đầu tiên w0,…,w15 được lấy trực tiếp từ các từ đầu vào m0,…, m15. Các từ còn lại w16,…,w63 được tạo ra bằng công thức lặp sử dụng các phép toán logic như xoay vòng (rotate), dịch phải (shift right), và XOR theo định nghĩa của chuẩn SHA-256.

Mục đích của bước này là tạo ra 64 từ w_i được sử dụng ở mỗi vòng của hàm nén, nhằm đảm bảo rằng toàn bộ thông điệp ảnh hưởng sâu sắc đến giá trị băm cuối cùng (đặc tính khuếch đại).

Trước khi bắt đầu xử lý khối dữ liệu đầu vào, SHA-256 sử dụng một tập hợp các giá trị khởi tạo ban đầu (initial hash values), gồm 8 từ 32-bit, ký hiệu là a0,b0,..., h0. Đây là các hằng số được định nghĩa trước trong chuẩn SHA-2, thường được suy ra từ phần thập phân của các căn bậc hai của các số nguyên tố đầu tiên.

Khối chính của sơ đồ là 64 vòng xử lý (được biểu diễn từ w0 đến w63). Trong mỗi vòng, thuật toán sử dụng các từ thông điệp wiw_iwi cùng với các hằng số vòng kj, kết hợp với giá trị băm tạm thời hiện tại (các thanh ghi a, b, ..., h) để tính toán lại 8 giá trị băm mới.

Các phép toán được sử dụng bao gồm:

Phép XOR

Phép AND, OR

Phép quay trái/quay phải (rotate)

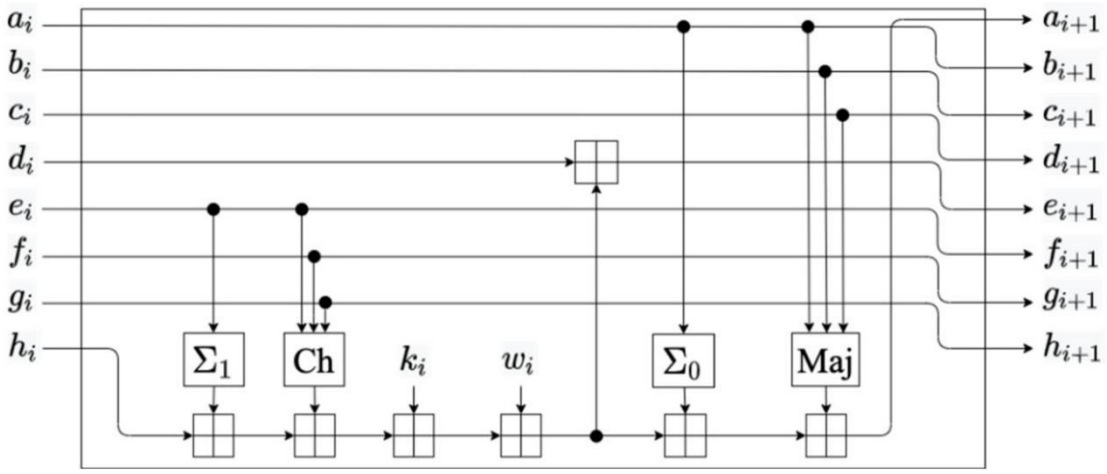
Hàm lựa chọn (Ch)

Hàm đa số (Maj)

Cộng modulo 232

Sau khi kết thúc 64 vòng, giá trị băm trung gian ban đầu (a0, b0,…, h0) sẽ được cộng với kết quả của vòng cuối cùng (a64, b64,…, h64) để tạo ra giá trị băm trung gian mới. Giá trị này sẽ được sử dụng làm đầu vào cho khối dữ liệu tiếp theo (nếu có), hoặc là kết quả cuối cùng nếu đây là khối cuối cùng.

Mã hóa dữ liệu



Hình 2.5. Qúa trình mã hóa dữ liệu của SHA-256

Trong mỗi vòng i của thuật toán SHA-256, 8 thanh ghi tạm thời ai,bi,...,hi, đại diện cho giá trị băm hiện tại sẽ được cập nhật thông qua các phép toán logic, cộng modulo và hàm phi tuyến. Các phép toán này đảm bảo tính hỗn loạn và khuếch đại – hai tính chất cốt lõi trong các hàm băm mật mã.

Các thành phần chính:

Σ0 và Σ1:

Là hai hàm hoán vị phi tuyến được định nghĩa như sau

Σ0(x) = ROTR2(x) ⊕ ROTR13(x) ⊕ ROTR22(x)

Σ1(x) = ROTR6(x) ⊕ ROTR11(x) ⊕ ROTR25(x)

Ch (Choose function):

Hàm chọn bit từ hai thanh ghi dựa trên giá trị điều kiện:

Ch(e,f,g) = (e ∧ f) ⊕ (¬e∧g)

Nếu e là 1 thì chọn bit từ f, ngược lại chọn từ g.

Maj (Majority function):

Trả về giá trị chiếm đa số trong 3 bit:

Maj(a,b,c) = (a∧b) ⊕ (a∧c) ⊕ (b∧c)

Ki: Là hằng số vòng, được định nghĩa sẵn theo chuẩn SHA-256

Wi: Là từ thông điệp tại vòng thứ iii, được tạo ra từ bước "Message Schedule" đã mô tả ở Hình 2.4

Quá trình tính toán

Tại mỗi vòng i, ta thực hiện:

$T1 = hi + \Sigma 1(ei) + Ch(ei,fi,gi) + ki + wi$

$T2 = \Sigma 0(ai) + Maj(ai,bi,ci)$

Sau đó, các thanh ghi được cập nhật theo quy tắc:

$hi+1 = gi, gi+1 = fi, fi+1 = ei, ei+1 = di+T1$

$di+1 = ci, ci+1 = bi, bi+1 = ai, ai+1 = T1 + T2$

Sơ đồ trong hình thể hiện một vòng cập nhật trạng thái nội bộ trong thuật toán SHA-256. Đây là phần trung tâm của hàm nén, nơi mà các giá trị băm được biến đổi qua nhiều vòng để tăng tính hỗn loạn và đảm bảo rằng mỗi bit của thông điệp ảnh hưởng đến kết quả cuối cùng.

Thông qua các phép toán logic như Ch, Maj, và các hàm hoán vị Σ, kết hợp với giá trị từ message schedule wi và hằng số vòng ki, thuật toán SHA-256 đạt được tính không thể đảo ngược, tính chống va chạm và tính phân tán mạnh mẽ, làm cho nó phù hợp để sử dụng trong bảo mật dữ liệu, xác thực và nhiều ứng dụng mật mã khác.

2.2.3 Độ an toàn của thuật toán SHA-256

SHA-256 (Secure Hash Algorithm 256-bit) được thiết kế bởi Cơ quan An ninh Quốc gia Hoa Kỳ (NSA) và được công bố như một phần của chuẩn FIPS PUB 180-4 do Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST) ban hành. Cho đến nay, SHA-256 vẫn được xem là một trong những hàm băm mật mã mạnh mẽ và an toàn nhất được sử dụng rộng rãi trong thực tiễn.

SHA-256 sinh ra một giá trị băm dài 256 bit, tương đương 22562^{256}2256 khả năng giá trị đầu ra. Điều này khiến việc tấn công vét cạn (brute-force) để tìm đầu vào tương ứng gần như không khả thi với khả năng tính toán hiện tại, vì nó đòi hỏi một lượng tài nguyên tính toán vượt quá khả năng của tất cả các hệ thống hiện nay trong thời gian hợp lý.

Một hàm băm được coi là an toàn nếu rất khó để tìm ra hai thông điệp khác nhau $x \neq y$ sao cho $SHA256(x) = SHA256(y)$. Với không gian 256 bit, theo phân tích lý

thuyết, để tìm va chạm bằng tấn công sinh nhật (birthday attack), kẻ tấn công cần thực hiện khoảng 2128 phép tính băm một con số cực kỳ lớn và chưa thực tế.

SHA-256 cũng đảm bảo rằng không thể tìm được đầu vào nếu chỉ biết giá trị băm h = SHA256(m). Việc tìm ra mmm sao cho SHA256(m) = h hiện vẫn chưa có thuật toán khả thi nào đạt hiệu quả tốt hơn brute-force, vốn yêu cầu trung bình 2256 bước thử.

SHA-256 còn ngăn chặn việc tìm một thông điệp khác m′ sao cho SHA256(m) = SHA256(m') dù đã biết trước một thông điệp mmm. Để phá vỡ đặc tính này cũng cần trung bình 2256 phép tính tương đương với mức độ an toàn rất cao.

Dù có một số nghiên cứu lý thuyết cố gắng rút ngắn quá trình phá SHA-2 (bao gồm SHA-256), cho đến nay chưa có bất kỳ cuộc tấn công nào thành công trên SHA-256 toàn bộ. Các biến thể có chiều dài ngắn hơn như SHA-1 đã bị phá bởi va chạm, nhưng SHA-256 vẫn chưa có lỗ hổng thực tiễn.

SHA-256 được dùng trong nhiều ứng dụng bảo mật như xác thực mật khẩu, chữ ký số, blockchain (ví dụ: Bitcoin), và nhiều giao thức truyền thông bảo mật (TLS/SSL). Tính phổ biến và được chấp nhận rộng rãi là minh chứng cho mức độ tin cậy và an toàn cao của thuật toán.

CHƯƠNG III. PHÂN TÍCH YÊU CẦU

3.1. YÊU CẦU CHỨC NĂNG

3.1.1. Chức năng mã hóa và giải mã

Ứng dụng được xây dựng hỗ trợ chức năng mã hóa và giải mã cả văn bản và file sử dụng thuật toán AES, đảm bảo tính bảo mật và toàn vẹn dữ liệu trong quá trình xử lý. Quá trình mã hóa được thực hiện theo chế độ CBC (Cipher Block Chaining), kết hợp với IV (Initialization Vector) để tăng cường độ an toàn, ngăn chặn các tấn công lặp khối dữ liệu. Việc hỗ trợ chế độ CBC cùng IV cho phép mỗi khối dữ liệu được mã hóa phụ thuộc vào khối trước đó và IV ban đầu, giúp kết quả mã hóa khó bị đoán trước kể cả khi dữ liệu đầu vào giống nhau. Nhờ đó, ứng dụng đáp ứng tốt yêu cầu về bảo mật khi xử lý thông tin nhạy cảm.

3.1.2. Quản lý khóa và IV

Ứng dụng cho phép người dùng linh hoạt trong việc thiết lập thông số bảo mật bằng cách hỗ trợ cả hai hình thức nhập thủ công và tự động tạo khóa cũng như IV. Cụ thể, người dùng có thể nhập khóa mã hóa và IV theo định dạng thủ công với 32 ký tự hex, đảm bảo kiểm soát hoàn toàn quá trình mã hóa. Bên cạnh đó, ứng dụng cũng cung cấp chức năng tự động tạo khóa và IV ngẫu nhiên, thuận tiện cho những trường hợp không yêu cầu cố định khóa hoặc nhằm tăng cường tính ngẫu nhiên và bảo mật trong quá trình sử dụng. Việc hỗ trợ song song cả hai phương thức này giúp nâng cao tính linh hoạt và phù hợp với nhiều mục đích triển khai thực tế.

3.1.3. Xử lý file

Ứng dụng hỗ trợ đầy đủ chức năng làm việc với file, cho phép người dùng dễ dàng chọn file cần mã hóa hoặc giải mã thông qua giao diện trực quan. Khi thực hiện mã hóa, file kết quả sẽ được lưu với phần mở rộng .encrypted, giúp người dùng dễ dàng nhận biết và quản lý các tệp đã mã hóa. Trong quá trình giải mã, ứng dụng hỗ trợ ghi đè file gốc nếu cần thiết, đồng thời cũng cho phép lưu kết quả giải mã vào một file mới để đảm bảo an toàn dữ liệu gốc. Nhờ đó, người dùng có thể linh hoạt lựa chọn cách xử lý và lưu trữ dữ liệu theo nhu cầu sử dụng.

3.1.4. Giao diện người dùng

Ứng dụng được thiết kế với giao diện thân thiện, cho phép người dùng dễ dàng chuyển đổi giữa hai chế độ làm việc: mã hóa/giải mã văn bản và mã hóa/giải mã file, tùy theo nhu cầu sử dụng. Trong quá trình thực hiện, hệ thống hiển thị thời gian bắt đầu

thao tác cũng như thời gian thực hiện mã hóa hoặc giải mã, giúp người dùng theo dõi hiệu suất hoạt động. Kết quả của quá trình xử lý được hiển thị rõ ràng trên giao diện, hỗ trợ việc kiểm tra và đánh giá. Ngoài ra, người dùng có thể xóa nhanh nội dung đầu vào và kết quả chỉ với một thao tác, thuận tiện trong trường hợp muốn làm việc với dữ liệu mới. Chức năng thoát chương trình cũng được tích hợp, giúp đóng ứng dụng một cách an toàn và nhanh chóng.

3.2. YÊU CẦU PHI CHỨC NĂNG

3.2.1. Bảo mật

Ứng dụng sử dụng thuật toán AES – một trong những thuật toán mã hóa đối xứng mạnh và phổ biến nhất hiện nay, đảm bảo mức độ bảo mật cao trong việc bảo vệ dữ liệu. Để tăng cường an toàn, mỗi lần mã hóa đều sử dụng một IV (Initialization Vector) ngẫu nhiên, giúp tránh các tấn công dựa trên mẫu dữ liệu lặp lại. Khóa mã hóa cũng được tạo ngẫu nhiên bằng cách sử dụng SecureRandom, một cơ chế tạo số ngẫu nhiên an toàn phù hợp với yêu cầu trong lĩnh vực mật mã. Đồng thời, ứng dụng tích hợp cơ chế kiểm tra tính hợp lệ của khóa và IV do người dùng nhập vào, đảm bảo rằng cả hai đều đúng định dạng và độ dài cần thiết trước khi thực hiện mã hóa hay giải mã, từ đó tránh các lỗi phát sinh và đảm bảo quá trình xử lý dữ liệu diễn ra chính xác.

3.2.2. Hiệu suất

Ứng dụng có khả năng đo lường và hiển thị chính xác thời gian thực hiện các thao tác mã hóa và giải mã, giúp người dùng theo dõi hiệu suất xử lý của hệ thống một cách trực quan. Đồng thời, tất cả các kết quả thời gian này được lưu trữ lại thành lịch sử, tạo điều kiện thuận lợi cho việc phân tích và so sánh trong các lần thực hiện tiếp theo. Dựa trên dữ liệu lịch sử thu thập được, ứng dụng cũng tính toán và hiển thị thời gian trung bình cho các quá trình mã hóa và giải mã, từ đó cung cấp cái nhìn tổng quan về hiệu suất hoạt động của thuật toán trong nhiều tình huống khác nhau.

3.2.3. Giao diện người dùng

Giao diện của ứng dụng được thiết kế hiện đại với bảng màu hài hòa, tạo cảm giác dễ chịu và chuyên nghiệp khi sử dụng. Font chữ Segoe UI được lựa chọn nhằm đảm bảo độ rõ ràng, dễ đọc trong mọi điều kiện hiển thị. Các nút chức năng tích hợp hiệu ứng hover sinh động, giúp người dùng dễ dàng nhận biết vị trí tương tác. Bên cạnh đó, các icon trực quan được sử dụng để minh họa rõ ràng cho từng chức năng, góp phần nâng cao trải nghiệm thao tác. Ứng dụng còn hỗ trợ tooltip hướng dẫn chi tiết từng bước sử

dụng, đồng thời cung cấp các thông báo lỗi và cảnh báo rõ ràng, giúp người dùng nhanh chóng nhận diện và khắc phục sự cố trong quá trình làm việc.

3.2.4. Khả năng sử dụng

Giao diện ứng dụng được thiết kế trực quan và thân thiện, giúp người dùng dễ dàng thao tác và sử dụng ngay cả khi không có nhiều kinh nghiệm. Ứng dụng hỗ trợ linh hoạt cả hai chế độ làm việc với văn bản và file, đáp ứng đa dạng nhu cầu xử lý dữ liệu. Đặc biệt, hệ thống có cơ chế kiểm tra đầu vào kỹ lưỡng nhằm tránh các lỗi phát sinh trong quá trình xử lý. Khi xảy ra lỗi, ứng dụng sẽ hiển thị thông báo rõ ràng, giúp người dùng nhanh chóng nắm bắt và khắc phục. Ngoài ra, để bảo vệ dữ liệu, hệ thống luôn yêu cầu xác nhận từ người dùng trước khi thực hiện ghi đè lên file đã tồn tại, đảm bảo an toàn và tránh mất mát thông tin không mong muốn.

3.2.5. Khả năng bảo trì

Mã nguồn của ứng dụng được tổ chức theo cấu trúc rõ ràng và khoa học, giúp dễ dàng quản lý và phát triển về sau. Các giá trị như màu sắc và cấu hình đều được định nghĩa dưới dạng hằng số, tạo sự nhất quán và thuận tiện khi cần thay đổi. Phần logic xử lý được tách biệt hoàn toàn khỏi phần giao diện, đảm bảo sự rõ ràng và giúp việc bảo trì, mở rộng trở nên đơn giản hơn. Ngoài ra, toàn bộ code đều được chú thích đầy đủ, cung cấp giải thích chi tiết về các chức năng và quy trình thực hiện, hỗ trợ tốt cho việc đọc hiểu và cộng tác trong nhóm phát triển.

3.2.6. Tương thích

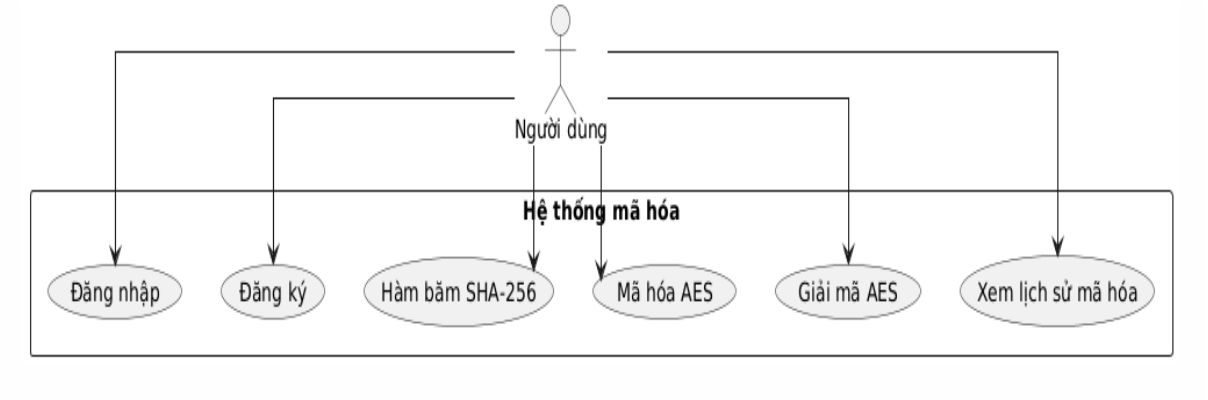
Ứng dụng được thiết kế để hỗ trợ mã hóa và giải mã các file có kích thước lớn một cách hiệu quả, đảm bảo xử lý dữ liệu mượt mà mà không gặp phải các giới hạn về bộ nhớ. Đồng thời, hệ thống cũng tương thích hoàn toàn với các ký tự Unicode trong văn bản, giúp mã hóa và giải mã chính xác nhiều ngôn ngữ và ký tự đặc biệt khác nhau. Nhờ được phát triển trên nền tảng Java, ứng dụng có khả năng tương thích cao với nhiều hệ điều hành khác nhau, mang lại sự linh hoạt và thuận tiện cho người dùng trên các môi trường Windows, macOS hay Linux.

3.2.7. Độ tin cậy

Ứng dụng được trang bị khả năng xử lý các trường hợp lỗi một cách hiệu quả, giúp ngăn ngừa và khắc phục các sự cố phát sinh trong quá trình sử dụng. Trước khi thực hiện bất kỳ thao tác nào, hệ thống luôn kiểm tra tính hợp lệ của dữ liệu đầu vào để đảm bảo các thông tin được nhập đúng định dạng và đầy đủ. Đặc biệt, với những thao tác

quan trọng như ghi đè file hay xóa dữ liệu, ứng dụng luôn yêu cầu người dùng xác nhận lại, tránh những sai sót không mong muốn. Tất cả các biện pháp này đều nhằm mục đích bảo vệ tối đa dữ liệu và quyền lợi của người dùng trong suốt quá trình làm việc với phần mềm.

3.3. USE-CASE DIAGRAM VÀ MÔ TẢ



Hình 3.1. Sơ đồ Use-Case

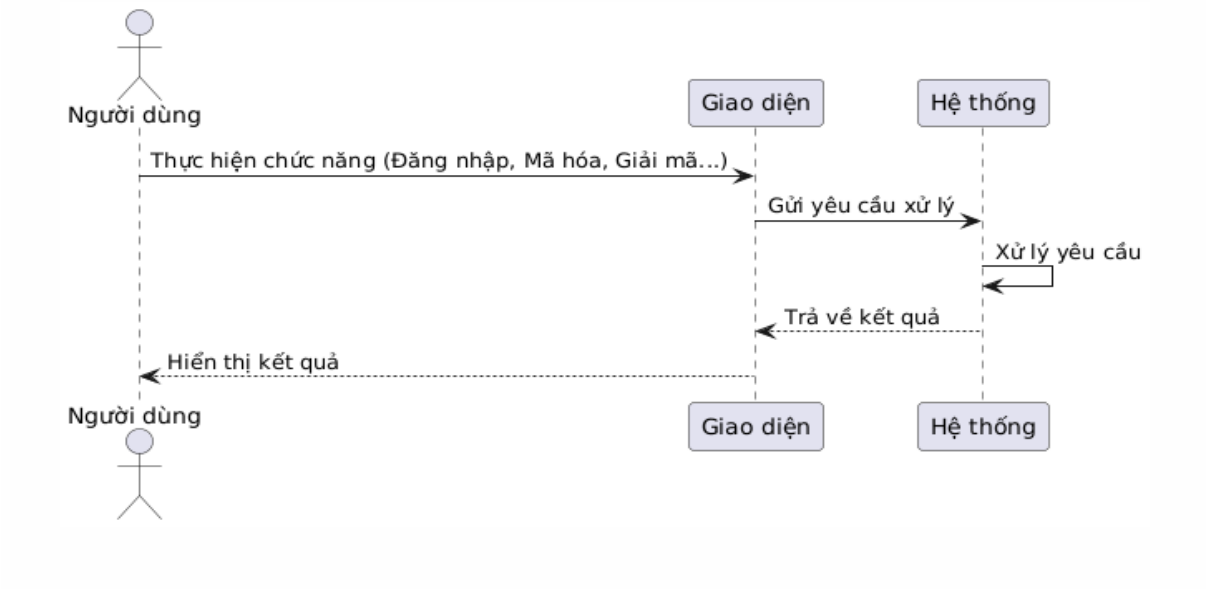
Đăng nhập: Khi truy cập vào hệ thống, người dùng cần thực hiện đăng nhập bằng tài khoản của mình. Chức năng này giúp đảm bảo chỉ những người dùng hợp lệ mới có thể sử dụng các tính năng mã hóa, giải mã và băm dữ liệu của hệ thống.

Nhập dữ liệu: Người dùng có thể nhập dữ liệu đầu vào cho quá trình mã hóa bằng hai cách: nhập trực tiếp văn bản vào ô nhập liệu trên giao diện hoặc lựa chọn một file có sẵn từ máy tính. Điều này giúp người dùng linh hoạt trong việc cung cấp dữ liệu cần mã hóa hoặc giải mã.

Chọn cách mã hóa: Sau khi nhập dữ liệu, người dùng có thể lựa chọn một trong hai phương pháp: mã hóa/giải mã bằng thuật toán AES (mã hóa đối xứng) hoặc sử dụng hàm băm SHA-256 để tạo ra giá trị băm của dữ liệu. Việc lựa chọn này giúp người dùng chủ động kiểm soát cách thức bảo vệ và xử lý dữ liệu.

Hiển thị kết quả: Người dùng sẽ thực hiện các thao tác nhập dữ liệu (văn bản hoặc file), chọn phương pháp xử lý (AES hoặc SHA-256), và xem kết quả (bao gồm cả các bước thực hiện của thuật toán) thông qua giao diện của hệ thống. Các chức năng này đảm bảo tính linh hoạt, minh bạch và dễ sử dụng cho người dùng khi làm việc với các thuật toán mã hóa và băm dữ liệu.

3.4. SEQUENCE DIAGRAM CHO CÁC CHỨC NĂNG CHÍNH



Hình 3.2. Sơ đồ trình tự xử lý yêu cầu

Hình trên mô tả sequence diagram (sơ đồ trình tự) cho quy trình chính của hệ thống mã hóa dữ liệu sử dụng AES và SHA-256, có tích hợp chức năng đăng nhập. Quy trình bắt đầu khi người dùng nhập thông tin đăng nhập trên giao diện. Giao diện sẽ gửi thông tin này đến hệ thống xác thực để kiểm tra. Sau khi xác thực thành công hoặc thất bại, kết quả sẽ được trả về cho người dùng.

Khi đăng nhập thành công, người dùng tiếp tục nhập hoặc chọn dữ liệu cần xử lý, sau đó lựa chọn phương pháp mã hóa (AES hoặc SHA-256). Giao diện sẽ kiểm tra lựa chọn của người dùng. Nếu người dùng chọn AES, giao diện sẽ truyền dữ liệu, khóa và IV đến module AES để thực hiện mã hóa hoặc giải mã, sau đó nhận lại kết quả. Nếu người dùng chọn SHA-256, giao diện sẽ truyền dữ liệu đến module SHA-256 để thực hiện băm và nhận lại kết quả băm.

Cuối cùng, kết quả (và từng bước thực hiện nếu có) sẽ được hiển thị cho người dùng trên giao diện. Sơ đồ này thể hiện rõ luồng tương tác giữa các thành phần chính của hệ thống, đảm bảo tính bảo mật, linh hoạt và minh bạch trong quá trình xử lý dữ liệu.

CHƯƠNG IV. THIẾT KẾ

4.1. KIẾN TRÚC TỔNG THỂ

4.1.1. Lớp giao diện người dùng

Chức năng chính của ứng dụng là hiển thị giao diện thân thiện, tạo điều kiện thuận lợi cho người dùng thực hiện các thao tác như đăng nhập, nhập hoặc chọn dữ liệu, lựa chọn thuật toán và xem kết quả xử lý. Ứng dụng tiếp nhận dữ liệu đầu vào từ người dùng và đồng thời hiển thị kết quả đầu ra một cách rõ ràng, trực quan. Các thành phần tiêu biểu đảm nhận vai trò này bao gồm các lớp như AESPanel, các form đăng nhập, cùng với các nút chức năng, ô nhập liệu và vùng hiển thị kết quả, góp phần tạo nên trải nghiệm người dùng mượt mà và hiệu quả.

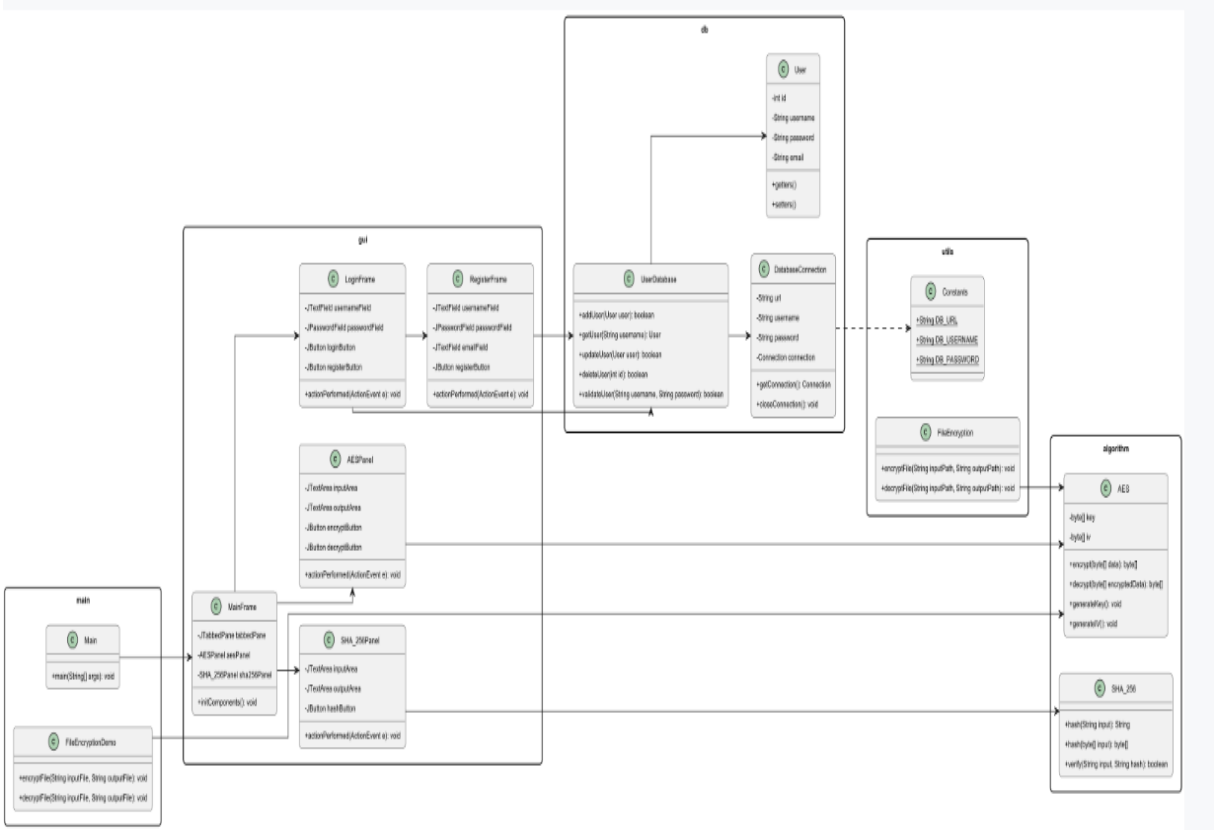
4.1.2. Lớp xử lý nghiệp vụ

Chức năng của ứng dụng bao gồm xử lý các yêu cầu từ giao diện người dùng như xác thực đăng nhập, mã hóa và giải mã dữ liệu bằng thuật toán AES, cũng như băm dữ liệu sử dụng SHA-256. Trong quá trình này, hệ thống kiểm tra tính hợp lệ của dữ liệu đầu vào, khóa mã hóa và IV để đảm bảo quá trình xử lý diễn ra chính xác và an toàn. Ngoài ra, ứng dụng còn điều phối luồng dữ liệu giữa các lớp để duy trì sự đồng bộ và hiệu quả trong hoạt động. Các thành phần tiêu biểu thực hiện các chức năng này gồm các lớp AES, SHA256, FileEncryption, và Authenticator (nếu có), với các phương thức chuyên biệt đảm nhiệm việc mã hóa, giải mã, băm và xác thực thông tin.

4.1.3. Lớp dữ liệu

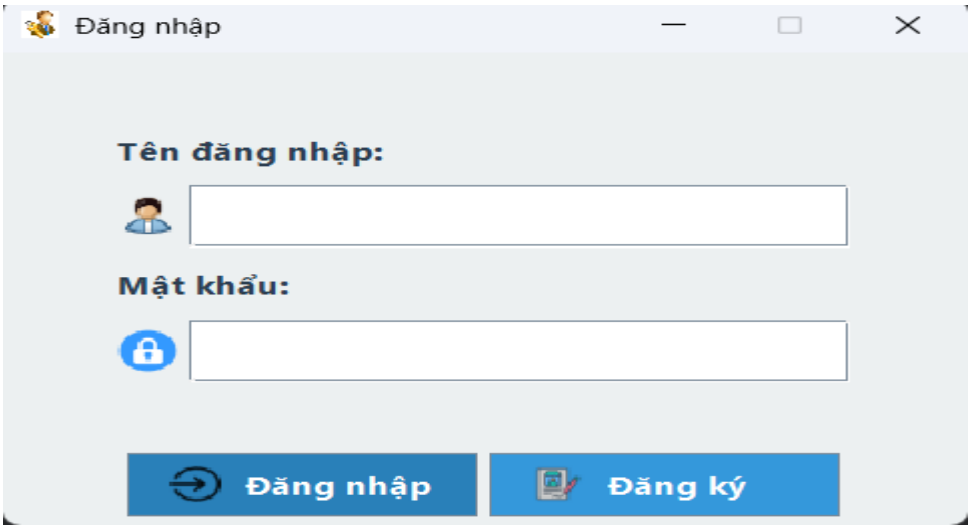
Chức năng của hệ thống bao gồm lưu trữ và truy xuất dữ liệu người dùng, các file đầu vào và đầu ra, cũng như thông tin đăng nhập nếu có. Khi người dùng chọn file để xử lý hoặc lưu kết quả, ứng dụng thực hiện các thao tác đọc và ghi file một cách chính xác và hiệu quả. Các thành phần tiêu biểu đảm nhiệm chức năng này thường là các lớp hoặc phương thức xử lý file hệ thống (File I/O), chịu trách nhiệm quản lý việc đọc, ghi dữ liệu và lưu trữ thông tin liên quan một cách an toàn và tiện lợi.

4.2. CLASS DIAGRAM



Hình 4.1. Sơ đồ các lớp đối tượng

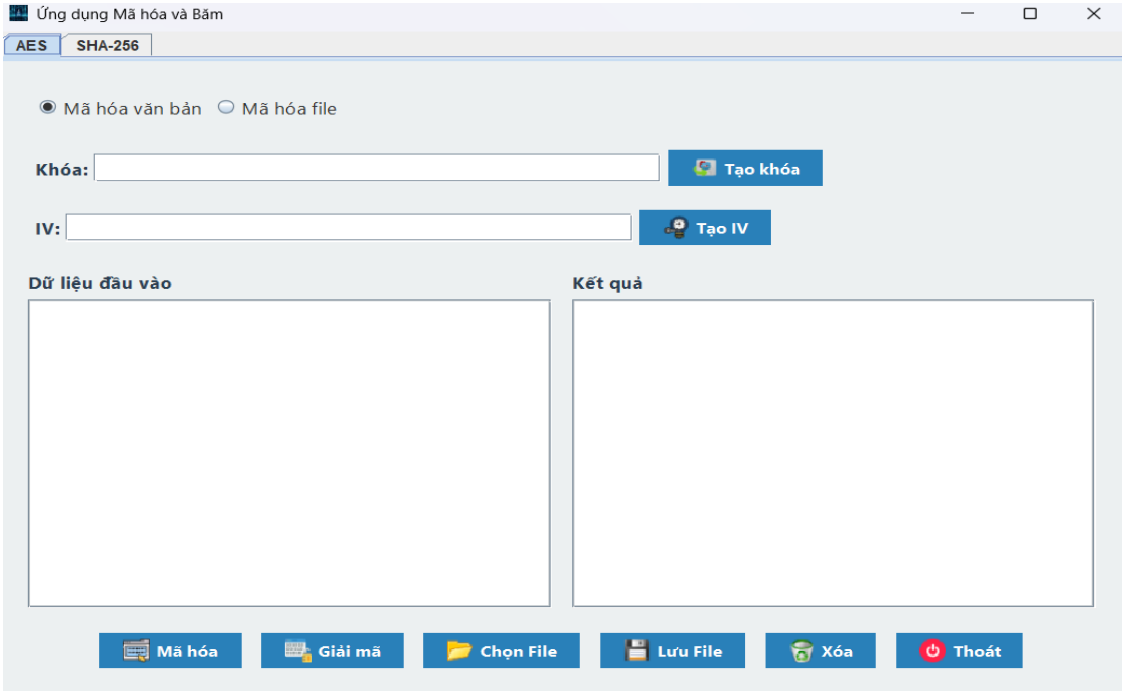
4.3. GIAO DIỆN NGƯỜI DÙNG



Hình 4.2. Giao diện đăng nhập

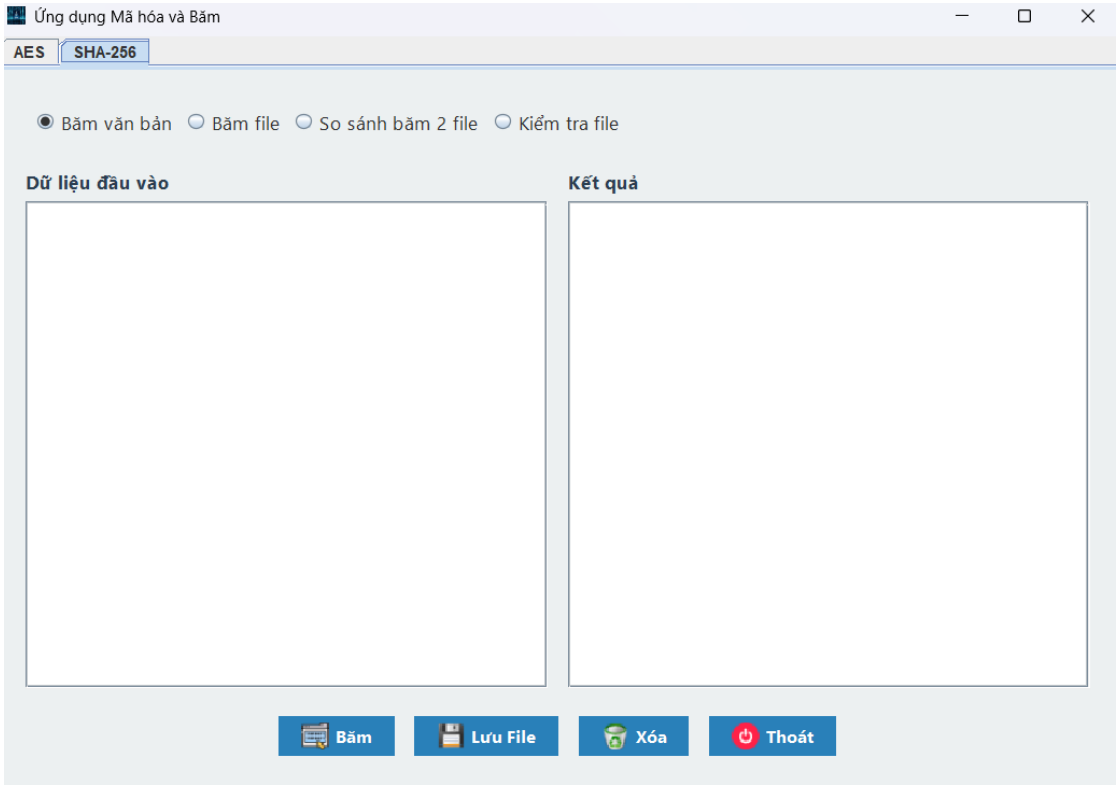
Giao diện trên là màn hình đăng nhập của hệ thống. Người dùng sẽ nhập tên đăng nhập và mật khẩu vào hai ô tương ứng. Giao diện được thiết kế thân thiện, trực quan với các biểu tượng minh họa cho từng trường nhập liệu, giúp người dùng dễ dàng nhận biết chức năng của từng ô. Hai nút chức năng chính ở phía dưới là "Đăng nhập" và "Đăng ký", cho phép người dùng thực hiện đăng nhập vào hệ thống hoặc chuyển sang màn hình

đăng ký tài khoản mới. Màu sắc chủ đạo là xanh dương tạo cảm giác hiện đại, chuyên nghiệp và dễ nhìn. Giao diện này đảm bảo tính bảo mật và thuận tiện cho người dùng khi truy cập vào các chức năng mã hóa, giải mã và băm dữ liệu của hệ thống.



Hình 4.3. Giao diện thuật toán mã hóa AES

Giao diện trên là màn hình chính của chức năng mã hóa/giải mã AES trong ứng dụng. Người dùng có thể lựa chọn giữa hai chế độ: mã hóa văn bản hoặc mã hóa file thông qua các nút radio ở phía trên. Giao diện cung cấp các trường để nhập khóa (Key) và IV (Initialization Vector), đồng thời hỗ trợ tạo khóa và IV ngẫu nhiên bằng các nút "Tạo khóa" và "Tạo IV" tiện lợi. Phần trung tâm của giao diện được chia thành hai vùng lớn: "Dữ liệu đầu vào" để người dùng nhập hoặc chọn dữ liệu cần mã hóa/giải mã, và "Kết quả" để hiển thị kết quả sau khi xử lý. Dưới cùng là các nút chức năng: Mã hóa, Giải mã, Chọn File, Lưu File, Xóa và Thoát, giúp người dùng thao tác dễ dàng với dữ liệu và file. Giao diện được thiết kế trực quan, hiện đại với các biểu tượng minh họa rõ ràng cho từng chức năng, giúp người dùng dễ dàng sử dụng và thao tác nhanh chóng trong quá trình mã hóa hoặc giải mã dữ liệu bằng thuật toán AES.

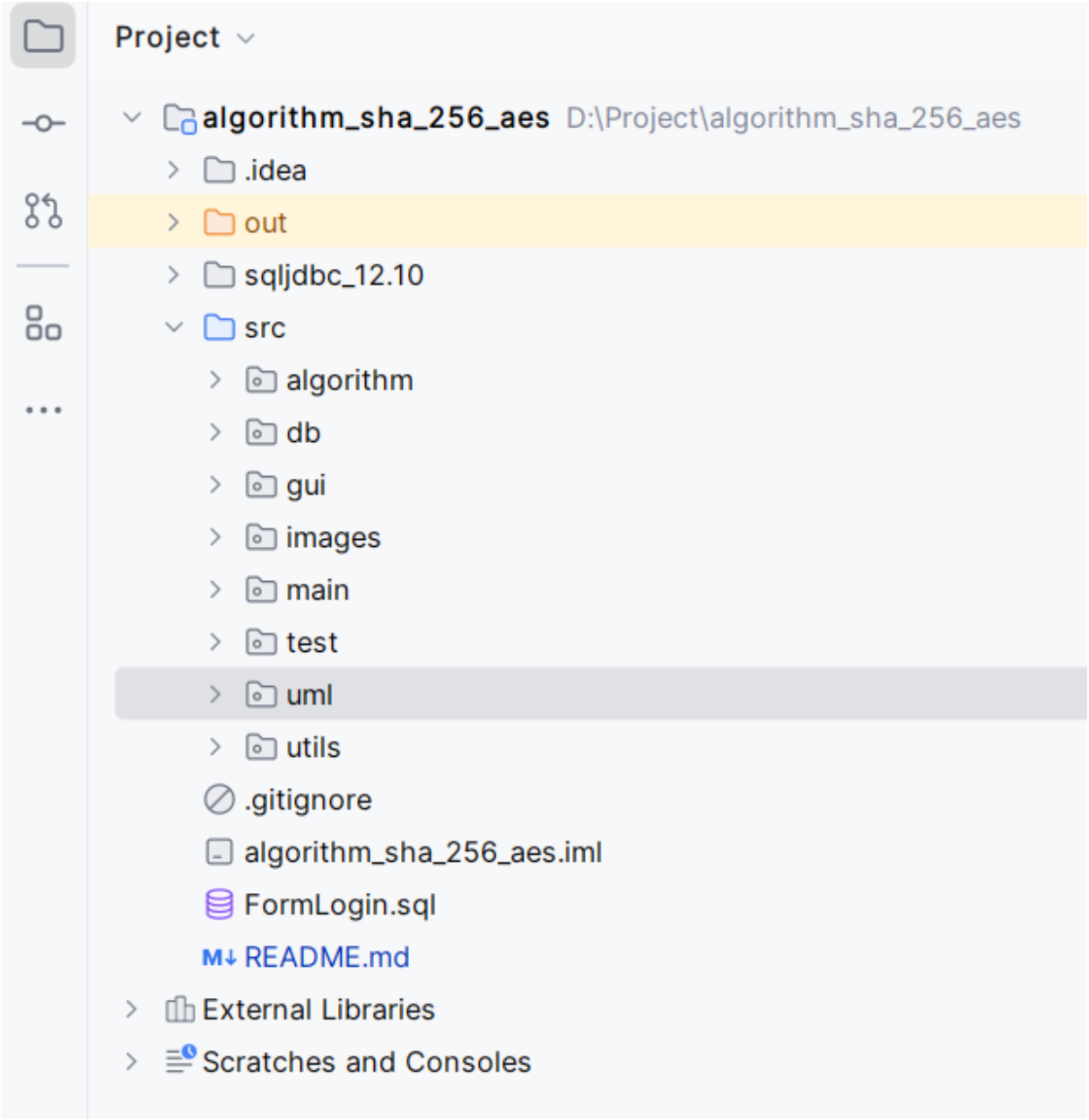


Hình 4.4. Giao diện thuật toán băm SHA-256

Giao diện trên là màn hình chức năng băm SHA-256 trong ứng dụng. Người dùng có thể lựa chọn giữa nhiều chế độ thao tác như: băm văn bản, băm file, so sánh băm 2 file, hoặc kiểm tra file thông qua các nút radio ở phía trên. Giao diện được chia thành hai vùng chính: "Dữ liệu đầu vào" để người dùng nhập văn bản hoặc chọn file cần băm, và "Kết quả" để hiển thị giá trị băm hoặc kết quả so sánh/kiểm tra. Phía dưới là các nút chức năng: Băm, Lưu File, Xóa và Thoát, giúp người dùng dễ dàng thực hiện thao tác băm dữ liệu, lưu lại kết quả, xóa dữ liệu hoặc thoát khỏi chương trình. Giao diện được thiết kế đơn giản, trực quan, giúp người dùng thao tác nhanh chóng và thuận tiện khi làm việc với thuật toán băm SHA-256 trong việc kiểm tra tính toàn vẹn và bảo mật dữ liệu.

CHƯƠNG V. CÀI ĐẶT

5.1. CẤU TRÚC SOURCE CODE



Hình 5.1. Cấu trúc của source code

Project được tổ chức một cách khoa học và rõ ràng theo các package và thư mục, giúp dễ dàng bảo trì, mở rộng và thuận tiện trong quá trình phát triển. Toàn bộ mã nguồn được chứa trong thư mục src/, trong đó có các package chuyên biệt như sau:

Package giao diện người dùng (GUI) gồm các lớp như LoginFrame.java (giao diện đăng nhập), RegisterFrame.java (giao diện đăng ký tài khoản), AESPanel.java (giao diện mã hóa/giải mã AES) và SHA_256Panel.java (giao diện băm SHA-256).

Package src/algorithm/ chứa các lớp hiện thực thuật toán, gồm AES.java cho thuật toán mã hóa và giải mã AES, cùng với SHA_256.java cho thuật toán băm SHA-256.

Package src/db/ đảm nhận quản lý dữ liệu người dùng, bao gồm lớp User.java định nghĩa đối tượng người dùng và UserDatabase.java quản lý danh sách người dùng cũng như thực hiện xác thực đăng nhập và đăng ký.

Package src/utils/ chứa các lớp tiện ích hỗ trợ, nổi bật là FileEncryption.java phục vụ cho việc mã hóa và giải mã file, cùng các lớp hoặc hàm tiện ích khác như chuyển đổi dữ liệu, kiểm tra định dạng.

Package src/images/ dùng để lưu trữ các file hình ảnh, icon sử dụng trong giao diện như nút bấm và logo, đồng thời cũng chứa các file sơ đồ UML quan trọng như class_diagram.puml (sơ đồ lớp), use_case.puml (sơ đồ use case) và sequence.puml (sơ đồ trình tự), hỗ trợ cho việc phân tích và thiết kế hệ thống.

5.2. CÁC THUẬT TOÁN VÀ KỸ THUẬT QUAN TRỌNG

5.2.1. Các thuật toán

```
1   package algorithm;
2
3   import utils.Constants;
4
5   public class AES {  6 usages   Tran Mai Ngoc Duy
6     // Mã hóa AES chế độ CBC
7     public static byte[] encryptAES_CBC(byte[] data, byte[] key, byte[] iv) {...}
39
40    // Giải mã AES chế độ CBC
41    public static byte[] decryptAES_CBC(byte[] encryptedData, byte[] key, byte[] iv) {...}
74
75    // Thêm padding PKCS#7
76    private static byte[] padData(byte[] data) {...}
85
86    // Loại bỏ padding PKCS#7
87    private static byte[] removePadding(byte[] paddedData) {...}
93
94    // Mã hóa một khối 16 bytes
95    private static byte[] encryptBlock(byte[] block, byte[][] expandedKey) {...}
130
131   // Giải mã một khối 16 bytes
132   private static byte[] decryptBlock(byte[] block, byte[][] expandedKey) {...}
167
168   // Thay thế byte bằng S-box
169   private static void subBytes(byte[][] state) {...}
176
177   // Thay thế byte bằng Inverse S-box
178   private static void invSubBytes(byte[][] state) {...}
185
186   // Dịch chuyển hàng
187   private static void shiftRows(byte[][] state) {...}
```

Hình 5.2. Thuật toán mã hóa AES

Mục đích của thuật toán là bảo vệ dữ liệu bằng cách chuyển đổi dữ liệu gốc thành dạng không thể đọc được nếu không có khóa giải mã tương ứng. Để tăng cường bảo mật, thuật toán sử dụng chế độ hoạt động CBC (Cipher Block Chaining), trong đó mỗi lần mã hóa đều áp dụng một vector khởi tạo (IV) ngẫu nhiên, giúp ngăn chặn các tấn công dựa trên mẫu dữ liệu lặp lại. Quá trình quản lý khóa và IV được thực hiện dưới dạng chuỗi hex, đồng thời có cơ chế kiểm tra kỹ lưỡng về độ dài và tính hợp lệ của chúng trước khi thực hiện mã hóa hoặc giải mã. Thuật toán hỗ trợ cả việc xử lý văn bản và file, đảm bảo tính linh hoạt và ứng dụng rộng rãi trong bảo vệ dữ liệu.

```
package algorithm;

public class SHA_256 {  12 usages  Tran Mai Ngoc Duy
    // Hằng số cho SHA-256
    private static final int[] K = {  1 usage
            0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
            0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
            0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
            0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
            0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
            0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
            0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
            0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
    };

    // Hàm tính toán hash SHA-256
    public byte[] calculateSHA256(byte[] data) {...}

    // Hàm padding dữ liệu cho SHA-256
    private byte[] padSHA256(byte[] data) {...}

    // Hàm xử lý từng khối 512 bits
    private void processBlock(byte[] data, int offset, int[] H) {...}

    // Hàm xoay phải (rotate right)
    private int rotateRight(int value, int shift) { return (value >>> shift) | (value << (32 - shift)); }
}
```

Hình 5.3. Thuật toán băm SHA-256

Mục đích của thuật toán là tạo ra một giá trị băm (hash) duy nhất tương ứng với mỗi dữ liệu đầu vào, giúp kiểm tra tính toàn vẹn của dữ liệu hoặc xác thực thông tin một cách chính xác. Thuật toán thực hiện đầy đủ các bước xử lý của SHA-256, bao gồm padding dữ liệu, chia nhỏ thành các khối, xử lý từng khối và cập nhật liên tục giá trị hash cuối cùng. Ngoài ra, hệ thống còn hỗ trợ băm cả văn bản và file, đồng thời cung cấp chức năng so sánh giá trị băm giữa hai file để kiểm tra tính toàn vẹn và xác định xem dữ liệu có bị thay đổi hay không.

5.2.2. Các kỹ thuật

Kỹ thuật xác thực người dùng

Mục đích của kỹ thuật này là đảm bảo chỉ những người dùng hợp lệ mới có thể truy cập và sử dụng các chức năng của hệ thống, từ đó bảo vệ an toàn cho dữ liệu và các hoạt động bên trong ứng dụng. Để tăng cường bảo mật, mật khẩu người dùng được lưu trữ dưới dạng giá trị băm SHA-256 thay vì lưu trữ trực tiếp mật khẩu gốc, giúp giảm thiểu nguy cơ lộ thông tin nhạy cảm. Quá trình kiểm tra thông tin đăng nhập và đăng ký được thực hiện thông qua lớp quản lý người dùng (UserDatabase), chịu trách nhiệm xác thực và quản lý dữ liệu người dùng một cách an toàn.

Kỹ thuật xử lý file

Mục đích của kỹ thuật này là cho phép người dùng dễ dàng mã hóa, giải mã, băm và lưu trữ dữ liệu dưới dạng file, giúp mở rộng khả năng ứng dụng của phần mềm trong thực tế. Để xây dựng giao diện trực quan và thân thiện, hệ thống sử dụng Java Swing để tạo các cửa sổ, panel, nút bấm, trường nhập liệu cùng nhiều thành phần giao diện khác. Bên cạnh đó, việc sử dụng icon, màu sắc hài hòa và hiệu ứng hover không chỉ làm tăng tính thẩm mỹ mà còn nâng cao trải nghiệm người dùng, giúp thao tác trở nên dễ dàng và trực quan hơn.

5.3. CODE MẪU CHO CÁC CHỨC NĂNG CHÍNH

```
179            // Kiểm tra đăng nhập
180            User user = UserDatabase.findUser(username);
181              if (user != null && user.getHashedPassword().equals(hashedPasswordHex)) {
182              ImageIcon successIcon = new ImageIcon(getClass().getResource( name: "/images/Accept.png"));
183
184              // Tạo custom button cho thông báo
185              UIManager.put("OptionPane.buttonBackground", SECONDARY_COLOR);
186              UIManager.put("OptionPane.buttonForeground", Color.WHITE);
187              UIManager.put("OptionPane.buttonFont", new Font( name: "Segoe UI", Font.BOLD,  size: 14));
188
189              // Hiển thị thông báo đăng nhập thành công
190              int response = JOptionPane.showOptionDialog(
191                      parentComponent: LoginFrame.this,
192                      message: "Đăng nhập thành công!",
193                      title: "Thông báo",
194                      JOptionPane.DEFAULT_OPTION,
195                      JOptionPane.INFORMATION_MESSAGE,
196                      successIcon,
197                      options: null,
198                      initialValue: null
199              );
```

Hình 5.4. Kiểm tra đăng nhập trong LoginFrame

```
287    private void registerEvents() {  1 usage  Tran Mai Ngoc Duy
288        // Sự kiện cho nút Mã hóa
289        encryptButton.addActionListener(new ActionListener() {  Tran Mai Ngoc Duy
290            @Override  Tran Mai Ngoc Duy
291            public void actionPerformed(ActionEvent e) {
292                try {
293                    String keyHex = keyField.getText();
294                    String ivHex = ivField.getText();
295                    if (keyHex.length() != 32 || !keyHex.matches( regex: "[0-9a-fA-F]{32}")) {
296                        JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
297                                message: "Khóa phải là 32 ký tự hex (16 byte)",
298                                title: "Lỗi",
299                                JOptionPane.ERROR_MESSAGE);
300                        return;
301                    }
302                    if (ivHex.length() != 32 || !ivHex.matches( regex: "[0-9a-fA-F]{32}")) {
303                        JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
304                                message: "IV phải là 32 ký tự hex (16 byte)",
305                                title: "Lỗi",
306                                JOptionPane.ERROR_MESSAGE);
307                        return;
308                    }
309
310                    byte[] keyBytes = hexToBytes(keyHex);
311                    byte[] ivBytes = hexToBytes(ivHex);
312
313                    if (isFileMode) {
314                        if (selectedFile == null) {
315                            JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
```

Hình 5.5. Mã hóa văn bản bằng thuật toán AES

```
// Sự kiện cho nút Giải mã
decryptButton.addActionListener(new ActionListener() {  Tran Mai Ngoc Duy
    @Override  Tran Mai Ngoc Duy
    public void actionPerformed(ActionEvent e) {
        try {
            String keyHex = keyField.getText();
            String ivHex = ivField.getText();
            if (keyHex.length() != 32 || !keyHex.matches( regex: "[0-9a-fA-F]{32}")) {
                JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
                        message: "Khóa phải là 32 ký tự hex (16 byte)",
                        title: "Lỗi",
                        JOptionPane.ERROR_MESSAGE);
                return;
            }
            if (ivHex.length() != 32 || !ivHex.matches( regex: "[0-9a-fA-F]{32}")) {
                JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
                        message: "IV phải là 32 ký tự hex (16 byte)",
                        title: "Lỗi",
                        JOptionPane.ERROR_MESSAGE);
                return;
            }

            byte[] keyBytes = hexToBytes(keyHex);
            byte[] ivBytes = hexToBytes(ivHex);

            if (isFileMode) {
                if (selectedFile == null) {
```

Hình 5.6. Mã hóa văn bản bằng thuật toán AES

```
                // Mã hóa file
                File encryptedFile = new File( pathname: selectedFile.getAbsolutePath() + ".encrypted");
                long startTime = System.nanoTime();
                FileEncryption.encryptFile(selectedFile, encryptedFile, keyBytes);
                long endTime = System.nanoTime();

                String time = getCurrentTime();
                String executionTime = formatExecutionTime(startTime, endTime);
                JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
                        message: "Đã mã hóa file thành công!\n" +
                        "File đã mã hóa: " + encryptedFile.getName() + "\n" +
                        "Thời gian: " + time + "\n" +
                        "Thời gian thực hiện: " + executionTime,
                        title: "Thông báo",
                        JOptionPane.INFORMATION_MESSAGE);
            } else {
                // Mã hóa văn bản
                String input = inputTextArea.getText();
                if (input.isEmpty()) {
                    JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
                            message: "Vui lòng nhập văn bản cần mã hóa",
                            title: "Cảnh báo",
                            JOptionPane.WARNING_MESSAGE);
                    return;
```

Hình 5.7. Giải mã văn bản bằng thuật toán AES

```
392             if (isFileMode) {
393                 if (selectedFile == null) {
394                     JOptionPane.showMessageDialog( parentComponent: AESPanel.this,
395                             message: "Vui lòng chọn file cần giải mã",
396                             title: "Cảnh báo",
397                             JOptionPane.WARNING_MESSAGE);
398                     return;
399                 }
400
401                 // Giải mã file
402                 String decryptedPath = selectedFile.getAbsolutePath();
403                 if (decryptedPath.endsWith(".encrypted")) {
404                     decryptedPath = decryptedPath.substring(0, decryptedPath.length() - 10);
405                 }
406                 File decryptedFile = new File(decryptedPath);
407
408                 if (decryptedFile.exists()) {
409                     int response = JOptionPane.showConfirmDialog( parentComponent: AESPanel.this,
410                             message: "File " + decryptedFile.getName() + " đã tồn tại. Bạn có muốn ghi đè không?",
411                             title: "Xác nhận ghi đè",
412                             JOptionPane.YES_NO_OPTION,
413                             JOptionPane.QUESTION_MESSAGE);
```

Hình 5.8. Giải mã file bằng thuật toán AES

```
190     private void addComponentsToPanel() {  1 usage   Tran Mai Ngoc Duy
191         // Panel cho input và output
192         JPanel textPanel = new JPanel(new GridLayout( rows: 1,  cols: 2,  hgap: 15,  vgap: 0));
193         textPanel.setBackground(BACKGROUND_COLOR);
194
195         // Tạo panel cho input với tiêu đề
196         JPanel inputPanel = new JPanel(new BorderLayout( hgap: 5,  vgap: 5));
197         inputPanel.setBackground(BACKGROUND_COLOR);
198         JLabel inputLabel = new JLabel( text: "Dữ liệu đầu vào");
199         inputLabel.setFont(new Font( name: "Segoe UI", Font.BOLD,  size: 14));
200         inputLabel.setForeground(TEXT_COLOR);
201         inputPanel.add(inputLabel, BorderLayout.NORTH);
202         inputPanel.add(new JScrollPane(inputTextArea), BorderLayout.CENTER);
203
204         // Tạo panel cho output với tiêu đề
205         JPanel outputPanel = new JPanel(new BorderLayout( hgap: 5,  vgap: 5));
206         outputPanel.setBackground(BACKGROUND_COLOR);
207         JLabel outputLabel = new JLabel( text: "Kết quả");
208         outputLabel.setFont(new Font( name: "Segoe UI", Font.BOLD,  size: 14));
209         outputLabel.setForeground(TEXT_COLOR);
210         outputPanel.add(outputLabel, BorderLayout.NORTH);
211         outputPanel.add(new JScrollPane(outputTextArea), BorderLayout.CENTER);
212
213         textPanel.add(inputPanel);
214         textPanel.add(outputPanel);
215
```

Hình 5.9. Băm văn bản bằng SHA-256

```
243     private void registerEvents() {  1 usage   Tran Mai Ngoc Duy
244         // Sự kiện cho nút Băm
245         hashButton.addActionListener(new ActionListener() {   Tran Mai Ngoc Duy
246             @Override   Tran Mai Ngoc Duy
247             public void actionPerformed(ActionEvent e) {
248                 try {
249                     if (isVerifyMode) {
250                         if (selectedFile == null) {
251                             JOptionPane.showMessageDialog( parentComponent: SHA_256Panel.this,
252                                     message: "Vui lòng chọn file cần kiểm tra",
253                                     title: "Cảnh báo",
254                                     JOptionPane.WARNING_MESSAGE);
255                             return;
256                         }
257
258                         // Tính hash hiện tại của file
259                         byte[] fileBytes = readFile(selectedFile);
260                         byte[] currentHashBytes = new SHA_256().calculateSHA256(fileBytes);
261                         String currentHash = bytesToHex(currentHashBytes);
262
263                         // Kiểm tra với hash đã lưu
264                         String savedHash = fileHashes.get(selectedFile.getAbsolutePath());
265                         if (savedHash == null) {
266                             // Lưu hash mới
267                             fileHashes.put(selectedFile.getAbsolutePath(), currentHash);
268                             JOptionPane.showMessageDialog( parentComponent: SHA_256Panel.this,
269                                     message: "Đã lưu hash cho file: " + selectedFile.getName() + "\n" +
270                                     "Hash: " + currentHash
```

Hình 5.10. Băm file bằng SHA-256

```
423         // Sự kiện cho nút So sánh
424         compareButton.addActionListener(new ActionListener() {   Tran Mai Ngoc Duy
425             @Override   Tran Mai Ngoc Duy
426             public void actionPerformed(ActionEvent e) {
427                 if (selectedFile == null || compareFile == null) {
428                     JOptionPane.showMessageDialog( parentComponent: SHA_256Panel.this,
429                             message: "Vui lòng chọn hai file cần so sánh",
430                             title: "Cảnh báo",
431                             JOptionPane.WARNING_MESSAGE);
432                     return;
433                 }
434
435                 try {
436                     // Tính hash cho file thứ nhất
437                     byte[] file1Bytes = readFile(selectedFile);
438                     long startTime1 = System.nanoTime();
439                     byte[] hash1Bytes = new SHA_256().calculateSHA256(file1Bytes);
440                     long endTime1 = System.nanoTime();
441                     String hash1 = bytesToHex(hash1Bytes);
442                     double time1 = (endTime1 - startTime1) / 1_000_000_000.0;
443
444                     // Tính hash cho file thứ hai
445                     byte[] file2Bytes = readFile(compareFile);
446                     long startTime2 = System.nanoTime();
447                     byte[] hash2Bytes = new SHA_256().calculateSHA256(file2Bytes);
448                     long endTime2 = System.nanoTime();
449                     String hash2 = bytesToHex(hash2Bytes);
```

Hình 5.11. So sánh file bằng thuật toán SHA-256

```
242         private void registerEvents() {  1 usage   Tran Mai Ngoc Duy
244             hashButton.addActionListener(new ActionListener() {  Tran Mai Ngoc Duy
246                 public void actionPerformed(ActionEvent e) {
247                     try {
248                         if (isVerifyMode) {
249                             if (selectedFile == null) {
250                                 JOptionPane.showMessageDialog( parentComponent: SHA_256Panel.this,
251                                         message: "Vui lòng chọn file cần kiểm tra",
252                                         title: "Cảnh báo",
253                                         JOptionPane.WARNING_MESSAGE);
254                                 return;
255                             }
256
257                             // Tính hash hiện tại của file
258                             byte[] fileBytes = readFile(selectedFile);
259                             byte[] currentHashBytes = new SHA_256().calculateSHA256(fileBytes);
260                             String currentHash = bytesToHex(currentHashBytes);
261
262                             // Kiểm tra với hash đã lưu
263                             String savedHash = fileHashes.get(selectedFile.getAbsolutePath());
264                             if (savedHash == null) {
265                                 // Lưu hash mới
266                                 fileHashes.put(selectedFile.getAbsolutePath(), currentHash);
267                                 JOptionPane.showMessageDialog( parentComponent: SHA_256Panel.this,
268                                         message: "Đã lưu hash cho file: " + selectedFile.getName() + "\n" +
269                                         "Hash: " + currentHash,
270                                         title: "Thông báo",
271                                         JOptionPane.INFORMATION_MESSAGE);
272                             } else {
273                                 // So sánh với hash đã lưu
```

Hình 5.12. Kiểm tra file bằng thuật toán SHA-256

5.4. HƯỚNG DẪN BUILD VÀ CHẠY CHƯƠNG TRÌNH

5.4.1. Yêu cầu hệ thống

Để phát triển ứng dụng, cần cài đặt Java Development Kit (JDK) phiên bản 8 trở lên nhằm đảm bảo tương thích và hỗ trợ đầy đủ các tính năng cần thiết. Về môi trường phát triển tích hợp (IDE), khuyến nghị sử dụng các công cụ phổ biến như IntelliJ IDEA, Eclipse hoặc NetBeans, tuy nhiên người dùng cũng có thể lựa chọn làm việc trực tiếp trên dòng lệnh nếu muốn. Các lựa chọn này giúp tối ưu hóa quá trình viết, kiểm tra và gỡ lỗi mã nguồn một cách hiệu quả.

5.4.2. Build và chạy bằng IDE

Để mở project, trước tiên bạn khởi động IDE và chọn tính năng Open Project hoặc Import Project, sau đó trỏ đến thư mục chứa mã nguồn của project. Tiếp theo, cần thiết lập cấu hình chạy bằng cách đảm bảo JDK phù hợp đã được chọn cho project. Bạn cũng phải chỉ định class chứa hàm main — thường là class khởi động giao diện như Main.java hoặc LoginFrame.java — làm điểm bắt đầu (entry point) cho chương trình. Cuối cùng, tiến hành build và chạy project bằng cách nhấn nút Run hoặc sử dụng phím tắt tương ứng như Shift+F10 trong IntelliJ, F11 trong NetBeans hoặc Ctrl+F11 trong Eclipse. Khi

đó, giao diện chương trình sẽ hiện ra, cho phép bạn đăng nhập và sử dụng đầy đủ các chức năng của ứng dụng.

CHƯƠNG VI. KIỂM THỬ

6.1. KẾ HOẠCH KIỂM THỬ

Kế hoạch kiểm thử được xây dựng nhằm đảm bảo tất cả các chức năng chính của hệ thống hoạt động đúng như yêu cầu, phát hiện và sửa các lỗi phát sinh trong quá trình sử dụng, đồng thời đảm bảo giao diện người dùng thân thiện, dễ sử dụng và không xảy ra lỗi trong quá trình thao tác. Phạm vi kiểm thử bao gồm các chức năng chính như đăng nhập, đăng ký; mã hóa và giải mã bằng thuật toán AES (cho cả văn bản và file); băm dữ liệu bằng SHA-256 (văn bản, file, so sánh và kiểm tra băm); các chức năng phụ như lưu file, xóa dữ liệu, thoát chương trình; và cuối cùng là kiểm thử giao diện người dùng.

Các loại kiểm thử được chia thành ba nhóm chính. Thứ nhất là kiểm thử chức năng, bao gồm việc kiểm tra khả năng đăng nhập với tài khoản đúng/sai, đăng ký tài khoản mới (kiểm tra tên trùng, xác nhận mật khẩu), mã hóa và giải mã văn bản/file với khóa hoặc tệp hợp lệ và không hợp lệ, chức năng băm SHA-256, lưu/xóa dữ liệu và thoát chương trình đúng cách. Thứ hai là kiểm thử giao diện, nhằm đánh giá việc hiển thị đầy đủ các thành phần, kiểm tra hoạt động của các nút bấm, trường nhập liệu, thông báo lỗi/thành công, cũng như tính thẩm mỹ như icon, màu sắc, font chữ và hiệu ứng khi tương tác. Cuối cùng là kiểm thử phi chức năng, tập trung vào hiệu suất xử lý với dữ liệu lớn, tính bảo mật (ví dụ như lưu mật khẩu dưới dạng băm thay vì plain text), và độ ổn định của hệ thống khi người dùng thao tác liên tục, nhập dữ liệu lớn hoặc thao tác sai.

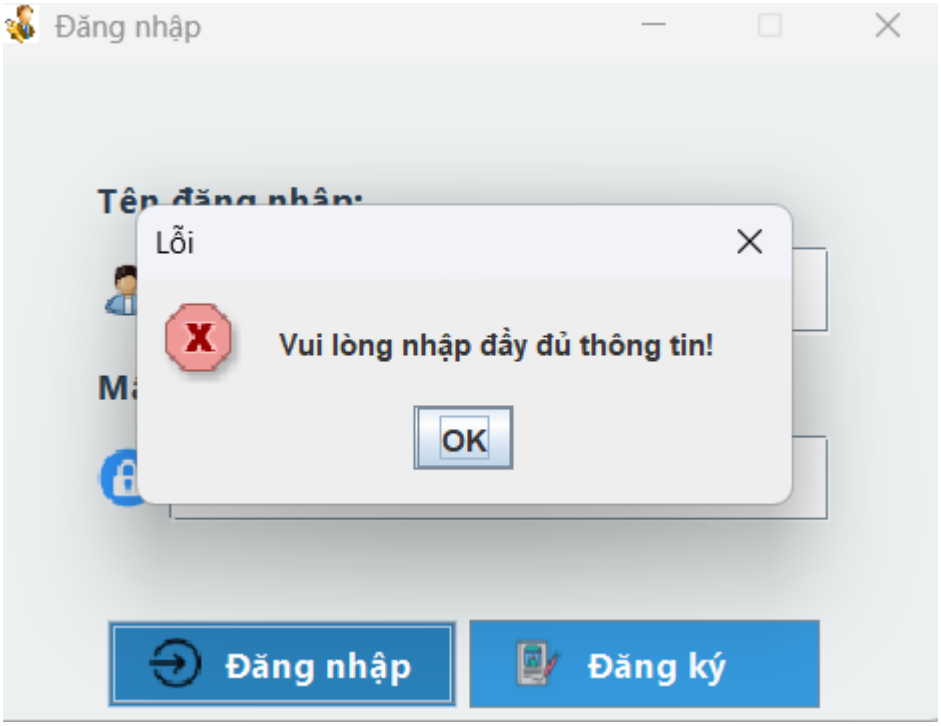
6.2. KẾT QUẢ KIỂM THỬ

Sau quá trình kiểm thử, tất cả các chức năng chính của hệ thống đều hoạt động đúng như mong đợi. Cụ thể, chức năng đăng nhập và đăng ký đều cho kết quả chính xác: hệ thống cho phép đăng nhập với tài khoản hợp lệ và thông báo lỗi khi nhập sai thông tin hoặc trùng tên tài khoản khi đăng ký.

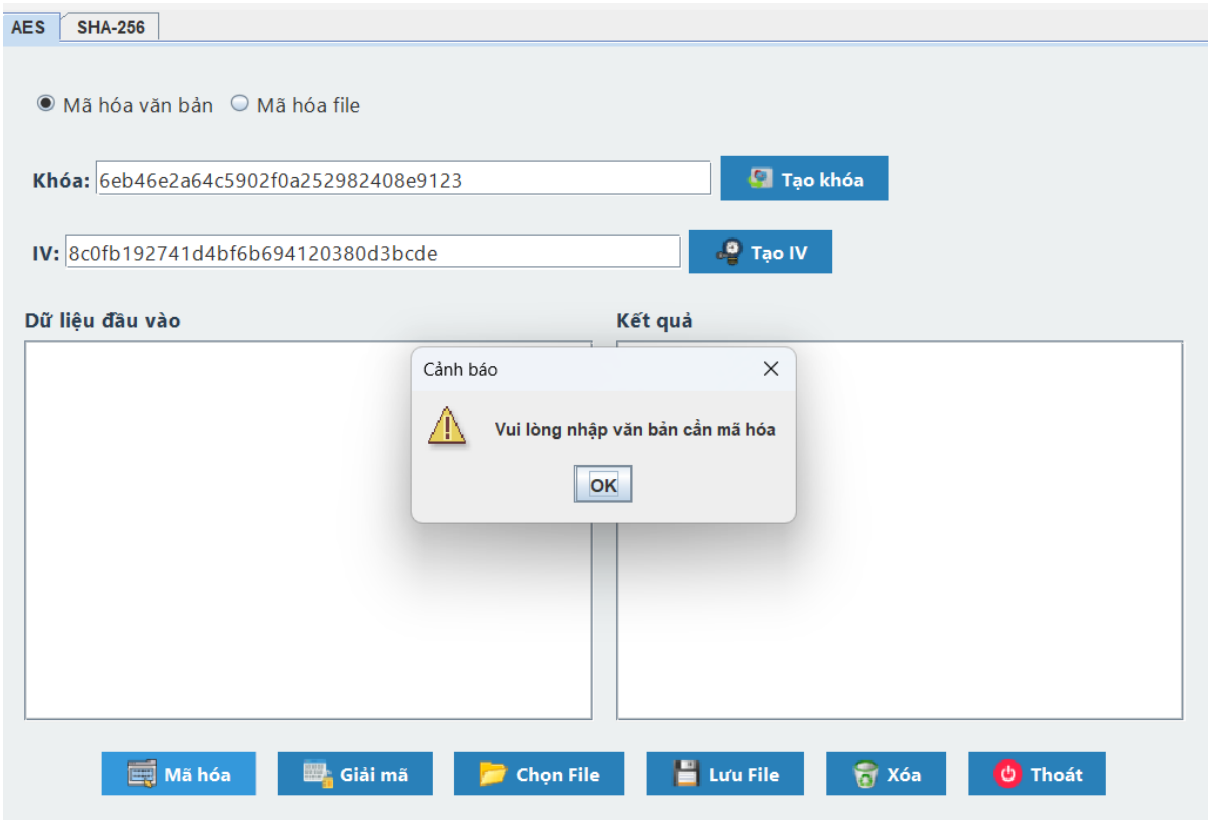
Chức năng mã hóa và giải mã AES hoạt động ổn định, đảm bảo dữ liệu sau khi mã hóa có thể giải mã lại chính xác về văn bản gốc, đồng thời hệ thống cũng kiểm tra và thông báo lỗi khi người dùng nhập khóa hoặc IV không hợp lệ.Chức năng băm SHA-256 cho kết quả băm đúng với cả văn bản và file, đồng thời hỗ trợ so sánh băm giữa hai file và kiểm tra tính toàn vẹn file một cách chính xác. Các chức năng phụ như lưu file, xóa dữ liệu và thoát chương trình đều thực hiện đúng vai trò, không phát sinh lỗi trong

quá trình sử dụng. Giao diện người dùng hiển thị đầy đủ các thành phần, các nút chức năng hoạt động tốt, thông báo rõ ràng, dễ hiểu.

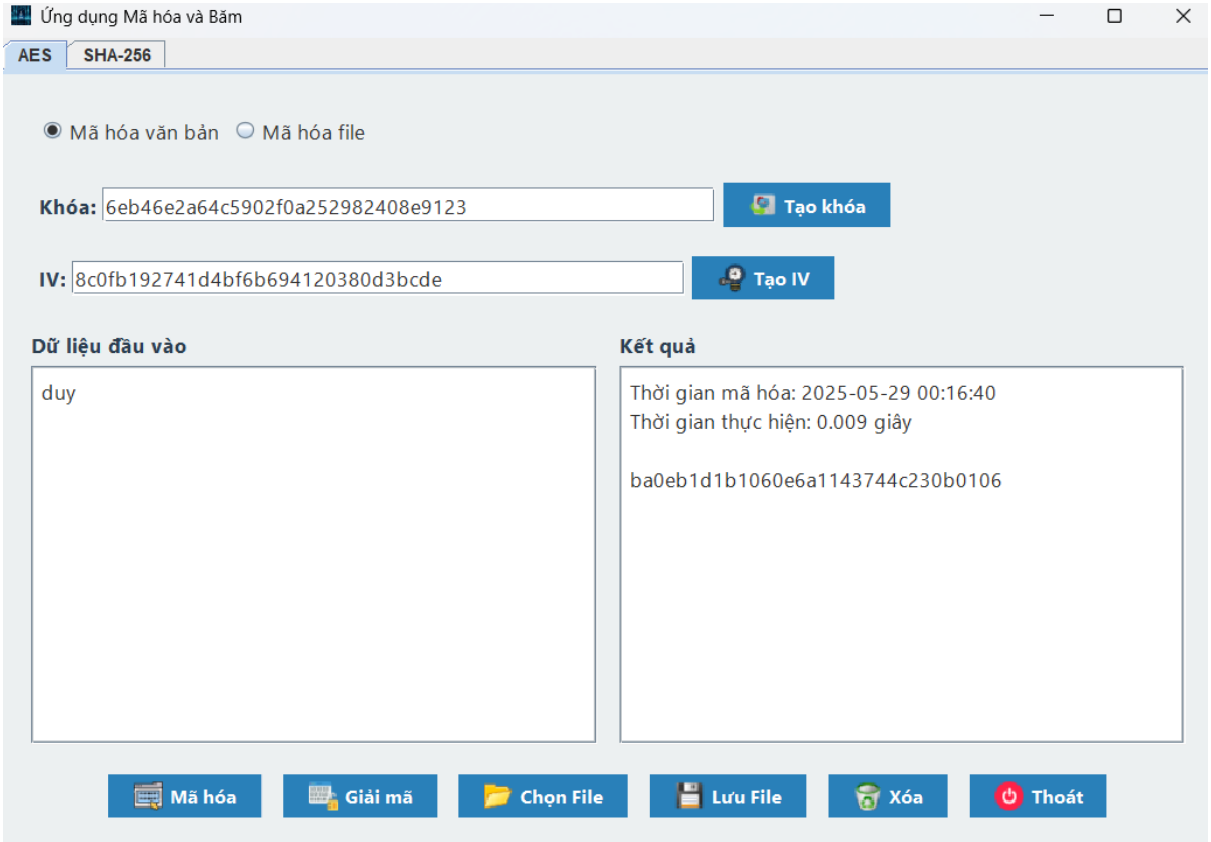
6.3. DEMO CÁC CHỨC NĂNG



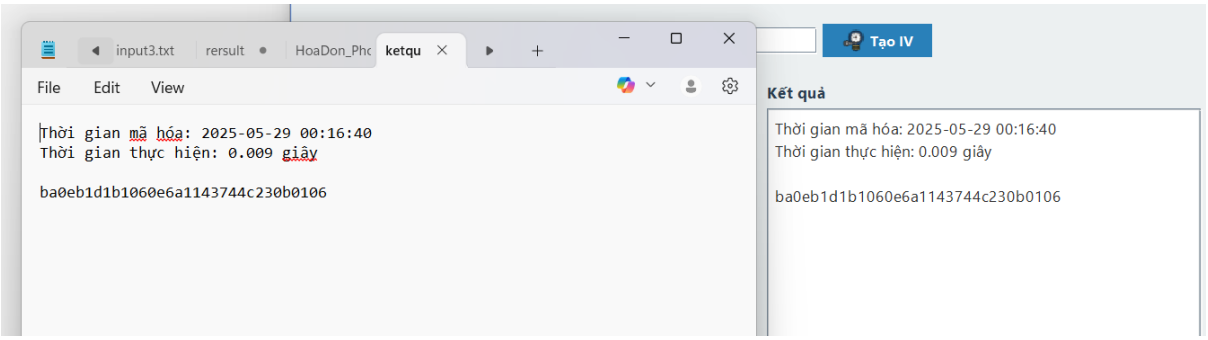
Hình 6.1. Thông báo lỗi đăng nhập thiếu thông tin



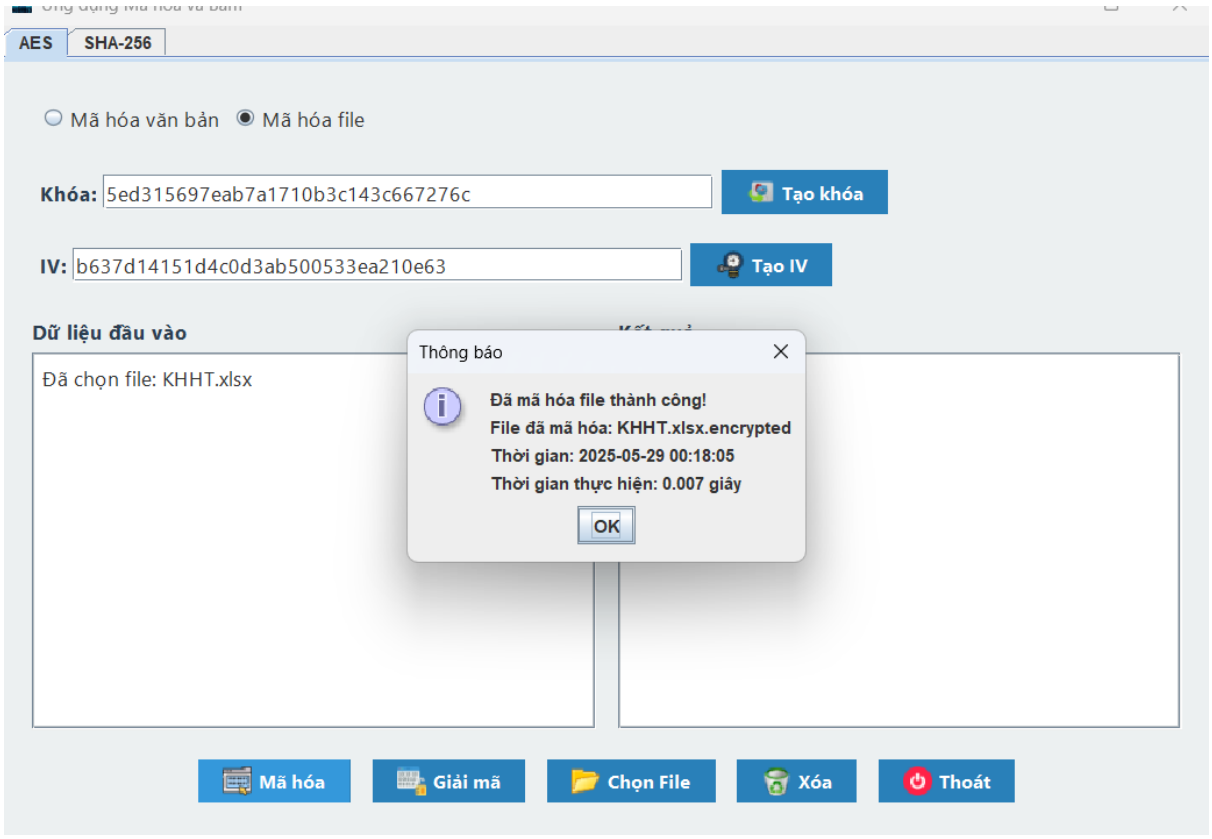
Hình 6.2. Cảnh báo thiếu dữ liệu đầu vào mã hóa



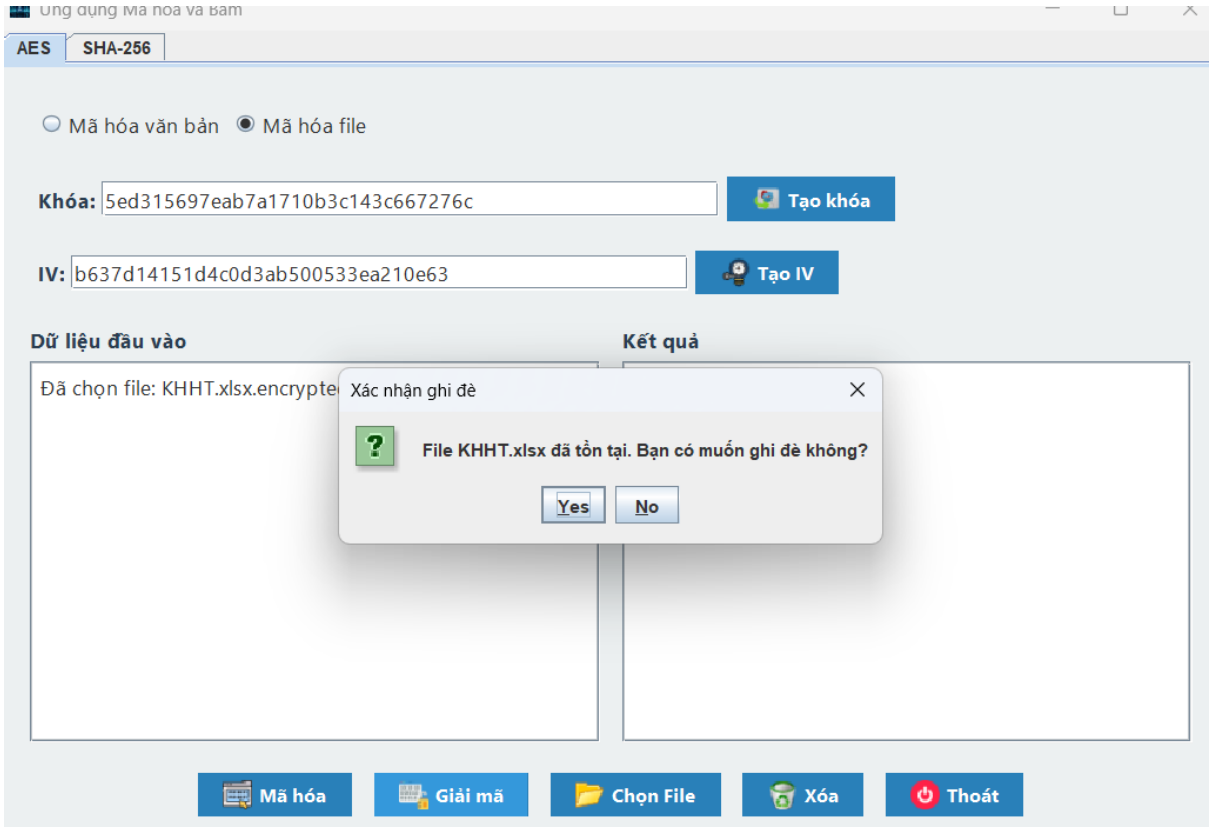
Hình 6.3. Kết quả mã hóa văn bản



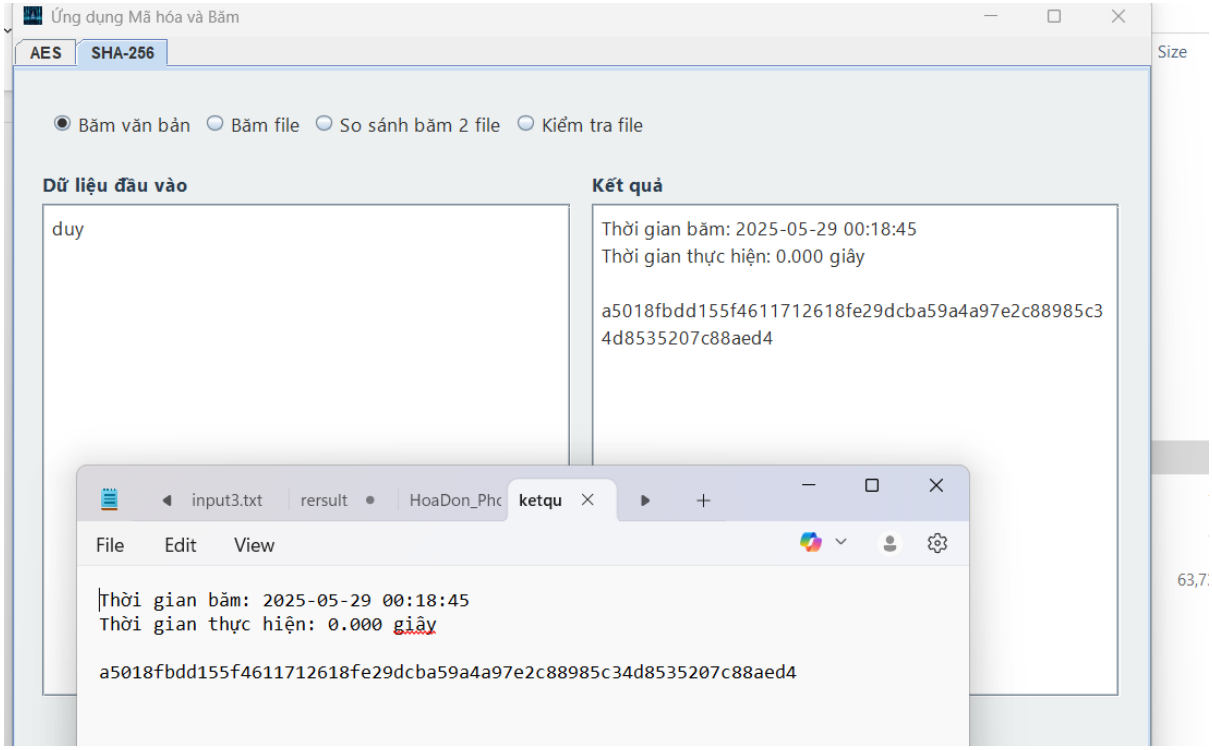
Hình 6.4. Kết quả mã hóa được lưu file txt



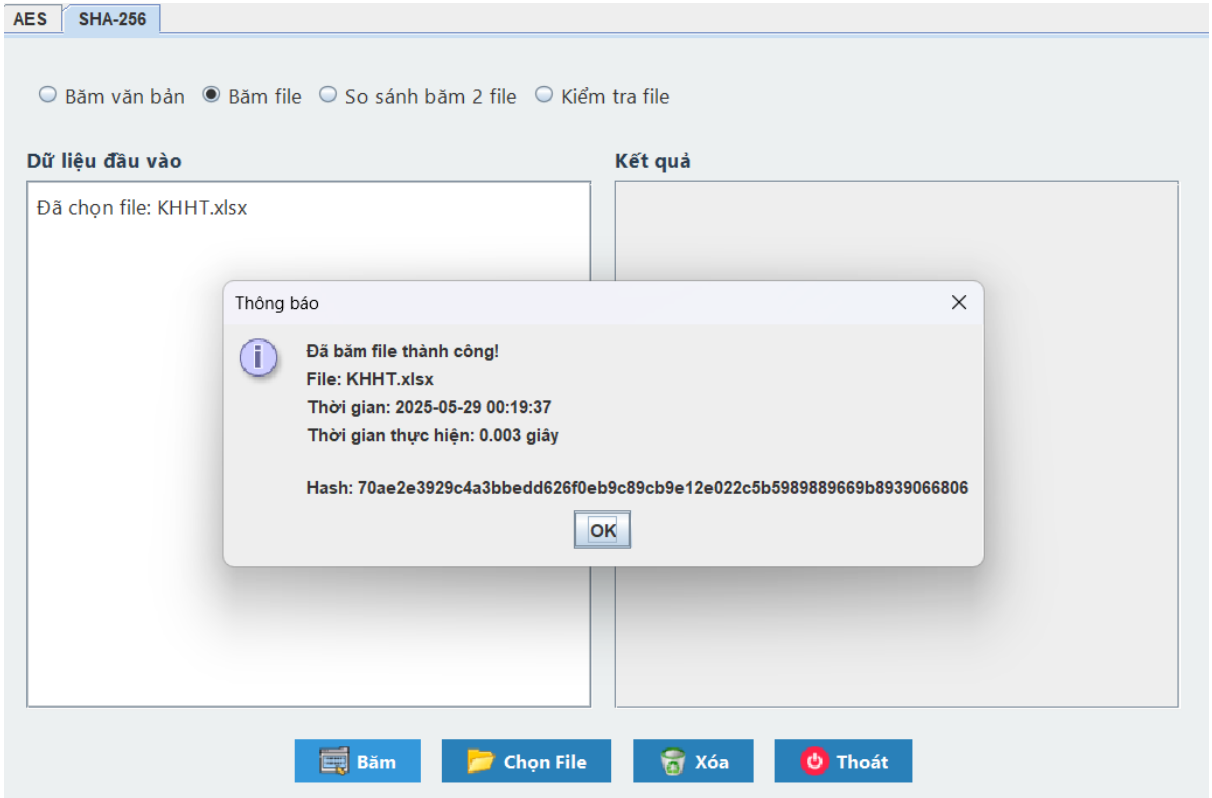
Hình 6.5. Kết quả mã hóa file thành công



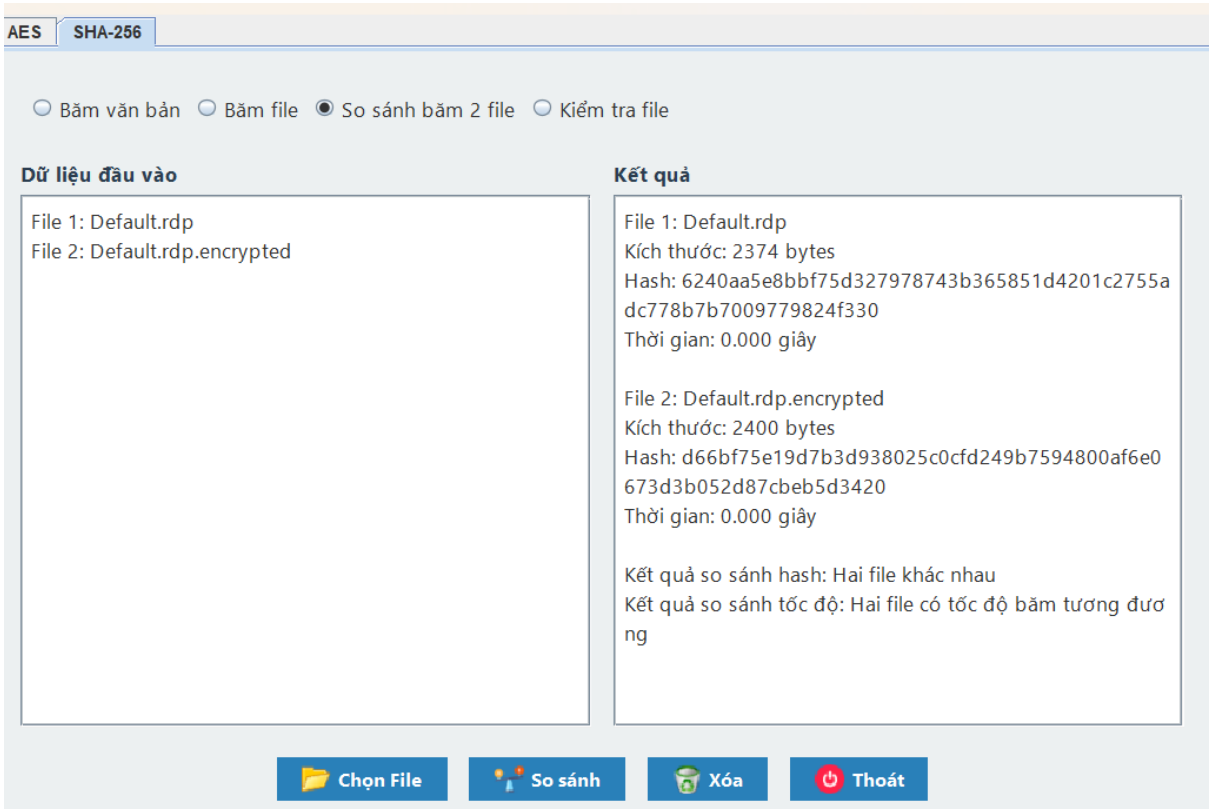
Hình 6.6. Giải mã file thành công



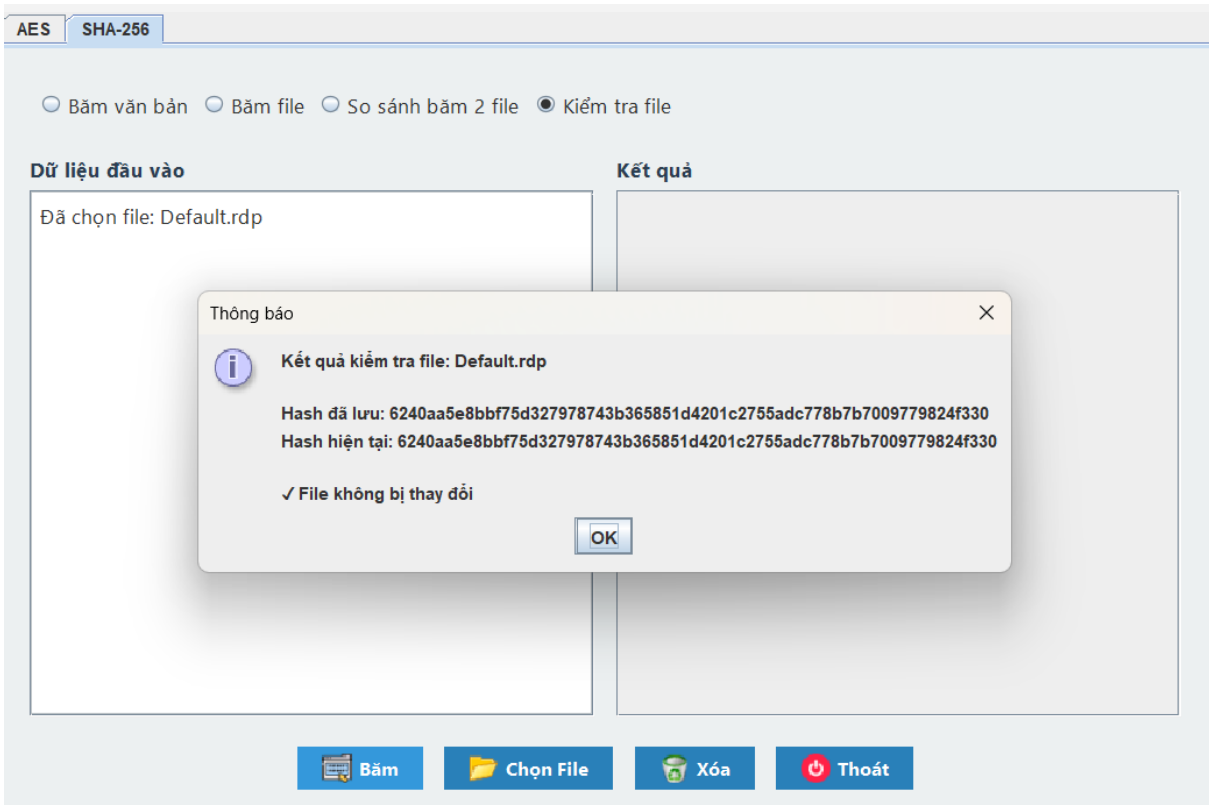
Hình 6.7. Kết quả băm SHA-256 lưu vào file txt



Hình 6.8. Kết quả băm file thành công



Hình 6.9. So sánh băm hai file khác nhau



Hình 6.10. Kết quả kiểm tra file không bị thay đổi

CHƯƠNG VII. KẾT LUẬN

7.1. KẾT QUẢ ĐẠT ĐƯỢC

Trong quá trình thực hiện đề tài, người học đã nắm vững lý thuyết về hai thuật toán quan trọng trong lĩnh vực mật mã hiện đại là AES và SHA-256, bao gồm nguyên lý hoạt động, cấu trúc thuật toán, các bước xử lý dữ liệu và vai trò của chúng trong bảo mật thông tin. Bên cạnh đó, việc cài đặt và mô phỏng thành công hai thuật toán bằng ngôn ngữ lập trình phù hợp như Java hoặc Python đã giúp người học củng cố kiến thức lý thuyết và hiểu rõ từng giai đoạn trong quá trình mã hóa, giải mã và băm dữ liệu. Đề tài cũng tiến hành phân tích chi tiết mức độ an toàn của hai thuật toán, đánh giá khả năng chống lại các dạng tấn công phổ biến như brute-force, tấn công va chạm, tìm ngược,… qua đó khẳng định tính tin cậy khi áp dụng trong thực tế. Cuối cùng, đề tài trình bày rõ mối quan hệ và ứng dụng thực tiễn của AES và SHA-256 trong các lĩnh vực quan trọng như bảo vệ dữ liệu, xác thực thông tin, blockchain, lưu trữ mật khẩu và chữ ký số.

7.2 HẠN CHẾ VÀ PHÁT TRIỂN

7.2.1. Hạn chế

Mặc dù đề tài đã đạt được những kết quả tích cực về mặt lý thuyết và thực hành, nhưng vẫn còn một số hạn chế nhất định. Cụ thể, hệ thống chưa được triển khai trên quy mô lớn hoặc tích hợp vào các hệ thống thực tế do giới hạn về thời gian và môi trường triển khai. Bên cạnh đó, hiệu năng của các thuật toán chưa được đánh giá một cách toàn diện, đặc biệt là về thời gian xử lý và mức độ tối ưu khi so sánh với các phương pháp khác hoặc khi xử lý khối lượng dữ liệu lớn. Ngoài ra, đề tài cũng chưa tiến hành thử nghiệm với dữ liệu thực tế trong các môi trường mạng có yếu tố an ninh, như trong giao tiếp giữa máy chủ và máy khách, hoặc trong các hệ thống lưu trữ phân tán, vốn là những tình huống ứng dụng phổ biến của AES và SHA-256.

7.2.2. Hạn chế phát triển

Trong tương lai, hướng phát triển của đề tài có thể tập trung vào việc tích hợp thuật toán AES và SHA-256 vào các hệ thống bảo mật hoàn chỉnh, chẳng hạn như ứng dụng đăng nhập an toàn hoặc hệ thống quản lý dữ liệu có tích hợp mã hóa và xác thực. Bên cạnh đó, việc mở rộng nghiên cứu sang các thuật toán nâng cao như AES-GCM (kết hợp mã hóa và xác thực), SHA-3 (phiên bản mới của SHA-2), hoặc các phương pháp mật mã hậu lượng tử cũng là một hướng đi tiềm năng để nâng cao mức độ bảo mật. Ngoài ra, đề tài có thể tiến hành đánh giá hiệu suất của các thuật toán trên phần cứng chuyên

dụng như FPGA hoặc GPU, từ đó hiểu rõ hơn về khả năng triển khai thực tế và hiệu quả xử lý của từng giải pháp. Cuối cùng, việc mở rộng nghiên cứu sang các vấn đề liên quan đến bảo mật toàn vẹn dữ liệu, bảo vệ quyền riêng tư người dùng và các ứng dụng thực tiễn như chữ ký số, xác thực trong blockchain hay các hệ thống phi tập trung cũng là những định hướng quan trọng và có tính ứng dụng cao.

TÀI LIỆU THAM KHẢO

1. Phạm Văn Nam (2011, 2013), *Lập trình hướng đối tượng Java*. Thư viện Trường Đại học Nha Trang, Nha Trang.

2. Trần Minh Văn (2011), *Giáo trình An toàn và bảo mật thông tin*. Thư viện Trường Đại học Nha Trang, Nha Trang.

3. Trần Thị Kim Thủy (2012), *Nghiên Cứu Về Ứng Dụng Chuẩn Mật Mã Nâng Cao (AES) Trong Xây Dựng Hàm Băm*, Luận văn thạc sĩ, Học Viện Công Nghệ Bưu Chính Viễn Thông, Hà Nội.

4. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm (simpliearn, 2024, Hướng dẫn dứt khoát để tìm hiểu SHA-256 (Thuật toán băm an toàn)).