**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP. HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙠🕮🙢-----

**BÁO CÁO**

**ĐỒ ÁN CHUYÊN NGÀNH**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG MÃ HÓA DỰA TRÊN THUẬT TOÁN ONE - TIME PAD**

**GVHD: Phạm Tuấn Khiêm**

**Sinh viên thực hiện:**

**Nguyễn Lộc Phát – 2033202020**

**Ngô Tấn Nhật - 2033204014**

**Nguyễn Quốc Thắng - 2033202026**

*TP. HỒ CHÍ MINH, tháng 8 năm 2025*

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP. HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙠🕮🙢-----

****

**BÁO CÁO**

**ĐỒ ÁN CHUYÊN NGÀNH**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG MÃ HÓA DỰA TRÊN THUẬT TOÁN ONE - TIME PAD**

**GVHD: Phạm Tuấn Khiêm**

**Sinh viên thực hiện:**

**Nguyễn Lộc Phát – 2033202020**

**Ngô Tấn Nhật - 2033204014**

**Nguyễn Quốc Thắng – 2033202026**

*TP. HỒ CHÍ MINH, tháng 8 năm 2025*

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

Chữ ký của giảng viên

**NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN**

Chữ ký của giảng viên

**LỜI CAM ĐOAN**

Chúng em xin cam đoan đây là công trình nghiên cứu của riêng chúng em. Các số liệu, kết quả nêu trong Đồ án là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác. Chúng em xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Đồ án này đã được cảm ơn và các thông tin trích dẫn trong Đồ án đã được chỉ rõ nguồn gốc.

**Sinh viên thực hiện tiểu luận**

*Nguyễn Lộc Phát*

*Nguyễn Quốc Thắng*

*Ngô Tấn Nhật*

**LỜI CẢM ƠN**

Chúng em xin chân thành cảm ơn thầy Phạm Tuấn Khiêm đã nhiệt tình hướng dẫn và hỗ trợ chúng em trong quá trình thực hiện đồ án và báo cáo đồ án tốt nghiệp với đề tài **“XÂY DỰNG ỨNG DỤNG MÃ HÓA DỰA TRÊN THUẬT TOÁN ONE-TIME PAD”.**

Sự tận tâm của thầy đã giúp chúng em hiểu rõ hơn về kiến thức và hoàn thiện bài báo cáo này. Chúng em rất trân trọng những lời khuyên và sự động viên của thầy trong suốt quá trình làm việc. Cảm ơn thầy đã đồng hành cùng chúng em!

**Sinh viên thực hiện tiểu luận**

*Nguyễn Lộc Phát*

*Nguyễn Quốc Thắng*

*Ngô Tấn Nhật*

**MỤC LỤC**

[**A. MỞ ĐẦU 8**](#_Toc207580408)

[**1. Lý do chọn đề tài 8**](#_Toc207580409)

[**2. Mục tiêu nghiên cứu 8**](#_Toc207580410)

[**3. Phạm vi nghiên cứu 9**](#_Toc207580411)

[**4. Phương pháp nghiên cứu 9**](#_Toc207580412)

[**B. NỘI DUNG 10**](#_Toc207580413)

[**CHƯƠNG 1: TỔNG QUAN VỀ THUẬT TOÁN ONE-TIME PAD 10**](#_Toc207580414)

[**1.1. Lý thuyết mật mã 10**](#_Toc207580415)

[**1.2. Thuật toán One-Time Pad (OTP) 23**](#_Toc207580416)

[**1.3. AES – Thuật toán mã hóa bổ sung 26**](#_Toc207580417)

[**1.4. Nén dữ liệu với Huffman Coding 30**](#_Toc207580418)

[**1.5. Hệ thống lai OTP – AES – Huffman 32**](#_Toc207580419)

[**1.6. Cách thuật toán hoạt động. 35**](#_Toc207580420)

[**CHƯƠNG 2: CÔNG NGHỆ JAMSTACK 39**](#_Toc207580421)

[**2.1. Khái niệm JAMstack 39**](#_Toc207580422)

[**2.2. Nguyên lý hoạt động 39**](#_Toc207580423)

[**2.3. Ưu điểm của JAMstack 40**](#_Toc207580424)

[**2.4. Hạn chế và thách thức 41**](#_Toc207580425)

[**2.5. Các công cụ và framework hỗ trợ JAMstack 41**](#_Toc207580426)

[**2.6. Ứng dụng thực tế của JAMstack 43**](#_Toc207580427)

[**2.7. So sánh JAMstack với mô hình truyền thống 43**](#_Toc207580428)

[**2.8. Xu hướng phát triển của JAMstack 44**](#_Toc207580429)

[**CHƯƠNG 3: XÂY DỰNG ỨNG DỤNG MINH HỌA 45**](#_Toc207580430)

[**3.1 Thiết kế giao diện ứng dụng 45**](#_Toc207580431)

[**3.2 Chức năng của ứng dụng 47**](#_Toc207580432)

[**3.3. Triển khai ứng dụng lên môi trường web 68**](#_Toc207580433)

[**C. KẾT LUẬN 75**](#_Toc207580434)

[**TÀI LIỆU THAM KHẢO 78**](#_Toc207580435)

# MỞ ĐẦU

## Lý do chọn đề tài

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, việc bảo mật dữ liệu ngày càng trở nên quan trọng. Hầu hết các giao dịch trực tuyến, từ ngân hàng, thương mại điện tử cho đến trao đổi thông tin cá nhân, đều cần đến những cơ chế mã hóa an toàn. Tuy nhiên, đi cùng với sự phát triển đó là những nguy cơ mất an toàn thông tin, hacker ngày càng tinh vi hơn trong việc tấn công và khai thác dữ liệu.

Thuật toán One-Time Pad (OTP) được biết đến như một phương pháp mã hóa đơn giản nhưng lại có tính bảo mật tuyệt đối về mặt lý thuyết. Tuy nhiên, việc triển khai OTP trong thực tế còn nhiều hạn chế, đặc biệt ở khâu quản lý và phân phối khóa. Chính vì vậy, việc nghiên cứu và xây dựng một ứng dụng minh họa cho OTP, đồng thời kết hợp thêm các cơ chế bổ sung như AES để bảo vệ mật khẩu và Huffman để nén khóa, sẽ giúp chúng em hiểu rõ hơn về lý thuyết mật mã và có thêm trải nghiệm thực tế trong việc xây dựng hệ thống bảo mật. Đây cũng là cơ hội để vận dụng kiến thức đã học vào một sản phẩm cụ thể, vừa mang tính học thuật, vừa có tính ứng dụng.

## Mục tiêu nghiên cứu

Nghiên cứu cơ sở lý thuyết về mật mã học, đặc biệt là thuật toán OTP và những vấn đề xoay quanh việc ứng dụng OTP trong thực tiễn. Xây dựng một ứng dụng web cho phép mã hóa và giải mã dữ liệu dựa trên OTP, có bổ sung các kỹ thuật như AES để bảo vệ mật khẩu và Huffman để giảm kích thước khóa.

Triển khai ứng dụng trên nền tảng web (Jamstack, Vercel) để có thể dễ dàng truy cập, thử nghiệm và đánh giá. Kiểm chứng tính đúng đắn và hiệu quả của ứng dụng thông qua các thí nghiệm với nhiều loại dữ liệu khác nhau (file văn bản, hình ảnh, tài liệu).

Từ đó rút ra nhận xét về điểm mạnh, điểm yếu, cũng như tiềm năng ứng dụng thực tiễn của OTP khi kết hợp cùng các kỹ thuật hiện đại.

## Phạm vi nghiên cứu

Đề tài tập trung vào thuật toán OTP kết hợp với AES và Huffman trong phạm vi một ứng dụng minh họa. Ứng dụng chủ yếu thực hiện mã hóa/giải mã file với dung lượng vừa phải, chưa đi sâu vào các hệ thống phân phối khóa phức tạp hay ứng dụng trong quy mô doanh nghiệp lớn.

Việc đánh giá chủ yếu dựa trên thực nghiệm với dữ liệu thử trên web do nhóm xây dựng, không so sánh hiệu năng với các hệ thống thương mại. Nghiên cứu tập trung vào khía cạnh kỹ thuật, chưa đề cập nhiều đến các khía cạnh pháp lý, thương mại hay triển khai thực tế ở môi trường sản xuất.

## Phương pháp nghiên cứu

**Nghiên cứu lý thuyết**: Tìm hiểu tài liệu, giáo trình, các công trình nghiên cứu liên quan đến mật mã học, OTP, AES và Huffman.

**Phân tích và thiết kế**: Xây dựng sơ đồ hệ thống lai (OTP – AES – Huffman), xác định quy trình mã hóa, giải mã, các thành phần chức năng của ứng dụng.

**Thực nghiệm**: Lập trình ứng dụng bằng Python/JavaScript, triển khai giao diện web, kiểm tra với nhiều loại file khác nhau.

**Đánh giá và so sánh**: Ghi nhận kết quả thực nghiệm (tính chính xác, tốc độ xử lý, mức độ nén), so sánh với lý thuyết để rút ra nhận xét.

**Triển khai ứng dụng**: Đưa sản phẩm lên môi trường web (Vercel) để có thể sử dụng trực tuyến, dễ dàng kiểm thử và chia sẻ.

# NỘI DUNG

# CHƯƠNG 1: TỔNG QUAN VỀ THUẬT TOÁN ONE-TIME PAD

* 1. **Lý thuyết mật mã** 
     1. **Khái niệm về lý thuyết mật mã.**

Mật mã là một lĩnh vực khoa học chuyên nghiên cứu về các phương pháp và kỹ thuật đảm bảo an toàn và bảo mật trong truyền tin liên lạc với giả thiết sự tồn tại của các thế lực thù địch, những kẻ muốn ăn cắp thông tin để lợi dụng và phá hoại. Tên gọi trong tiếng Anh, Cryptology được dẫn giải nguồn gốc từ tiếng Hy lạp, trong đó kryptos nghĩa là “che dấu”, logos nghĩa là “từ ngữ”.

Cụ thể hơn, các nhà nghiên cứu lĩnh vực này quan tâm xây dựng hoặc phân tích (để chỉ ra điểm yếu) các giao thức mật mã (cryptographic protocols), tức là các phương thức giao dịch có đảm bảo mục tiêu an toàn cho các bên tham gia (với giả thiết môi trường có kẻ đối địch, phá hoại).

Ngành mật mã (cryptology) thường được quan niệm như sự kết hợp của 2 lĩnh vực con :

* Sinh, chế mã mật (cryptography): nghiên cứu các kỹ thuật toán học nhằm cung cấp các công cụ hay dịch vụ đảm bảo an toàn thông tin
* Phá giải mã (cryptanalysis): nghiên cứu các kỹ thuật toán học phục vụ phân tích phá mật mã và/hoặc tạo ra các đoạn mã giản nhằm đánh lừa bên nhận tin.

Hai lĩnh vực con này tồn tại như hai mặt đối lập, “đấu tranh để cùng phát triển” của một thể thống nhất là ngành khoa học mật mã (cryptology). Tuy nhiên, do lĩnh vực thứ hai (cryptanalysis) ít được phổ biến quảng đại nên dần dần, cách hiểu chung hiện 2 nay là đánh đồng hai thuật ngữ cryptography và cryptology. Theo thói quen chung này, hai thuật ngữ này có thể dùng thay thế nhau. Thậm chí cryptography là thuật ngữ ưa dùng, phổ biến trong mọi sách vở phổ biến khoa học, còn cryptology thì xuất hiện trong một phạm vi hẹp của các nhà nghiên cứu học thuật thuần túy.

Mặc dù trước đây hầu như mật mã và ứng dụng của nó chỉ phổ biến trong giới hẹp, nhưng với sự phát triển vũ bão của công nghệ thông tin và đặc biệt là sự phổ biến của mạng Internet, các giao dịch có sử dụng mật mã đã trở nên rất phổ biến. Chẳng hạn, ví dụ điển hình là các giao dịch ngân hàng trực tuyến hầu hết đều được thực hiện qua mật mã. Ngày nay, kiến thức ngành mật mã là cần thiết cho các cơ quan chính phủ, các khối doanh nghiệp và cả cho cá nhân. Một cách khái quát, ta có thể thấy mật mã có các ứng dụng như sau:

* + Với các chính phủ: bảo vệ truyền tin mật trong quân sự và ngoại giao, bảo vệ thông tin các lĩnh vực tầm cỡ lợi ích quốc gia.
  + Trong các hoạt động kinh tế: bảo vệ các thông tin nhạy cảm trong giao dịch như hồ sơ pháp lý hay y tế, các giao dịch tài chính hay các đánh giá tín dụng.
  + Với các cá nhân: bảo vệ các thông tin nhạy cảm, riêng tư trong liên lạc với thế giới qua các giao dịch sử dụng máy tính và/hoặc kết nối mạng.
    1. **Mật mã học - một số thuật ngữ cơ bản trong cryptography**

Một số thuật ngữ cơ bản của Cryptography bao gồm:

Plaintext (Bản rõ): là thông tin ban đầu chưa được che giấu. Nó có thể ở dạng văn bản, hình ảnh, video hoặc bất kỳ định dạng khác. Plaintext có thể dễ dàng đọc và hiểu bởi bất kỳ ai có quyền truy cập vào nó nên nó cần được che dấu và bảo vệ.

Ciphertext (Bản mã): Là kết quả của quá trình mã hoá plaintext, là dạng dữ liệu đã được che giấu. Ciphertext là vô nghĩa với các bên khác, ngoại trừ các bên trao đổi thông tin với nhau.

Key (Khoá): Còn gọi là cryptographic key (khoá mật mã), là một mẩu thông tin dưới dạng chuỗi số hoặc chữ cái, được sử dụng trong quá trình mã hoá và giải mã. Key là thành phần tối quan trọng, cần được giữ kín. Nếu key bị lộ, bất kỳ ai có được key cũng có thể biết được nội dung của plaintext.

Cipher: Là thuật toán dùng để má hoá và giải mã dữ liệu.

Decipher (Phá mã): Là thuật toán dùng để tấn công vào ciphertext với mục tiêu tiếp cận đến plaintext mà không cần tới key.

Encryption: Là quá trình mã hoá thông tin từ plaintext thành ciphertext sử dụng cipher và key phù hợp.

Decryption: Là quá trình đưa ciphertext về plaintext sử dụng cipher và key phù hợp. Encrypt (Mã hoá): Là hành động biến plaintext thành dạng ciphertext.

Decrypt (Giải mã): Là hành động đưa ciphertext về plaintext sử dụng cipher và key phù hợp.

* + 1. **Mục tiêu của mật mã học**

Về cơ bản, mục tiêu chính của cryptography bao gồm:

Confidentiality (Tính bảo mật): Chỉ có các bên liên quan trong quá trình trao đổi, chủ sở hữu hoặc người có thẩm quyền mới có quyền truy cập vào các thông tin liên quan.

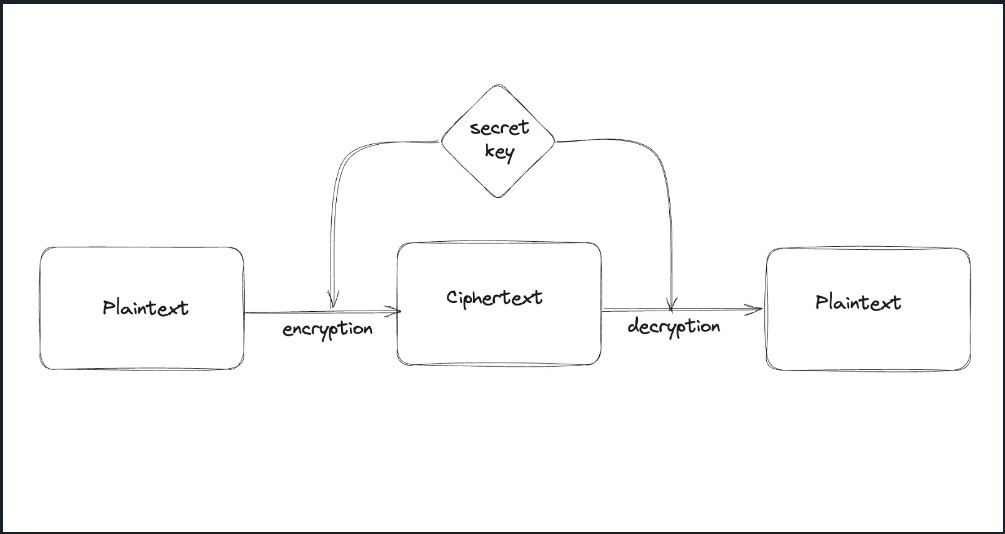
Integrity (Tính toàn vẹn): Dữ liệu trong quá trình lưu trữ và trao đổi không được thay đổi hoặc bị mất.

Authentication (Tính xác thực): Người nhận phải biết được thông điệp mà họ nhận được đến từ ai, và xác thực được danh tính người gửi.

Non-repudiation (Tính chống chối bỏ): Người gửi không thể chối bỏ được thông điệp mà họ đã gửi. Người gửi sẽ phải chịu toàn bộ trách nhiệm với thông tin mà họ đã gửi.

Chúng ta có nhiều loại mã hoá, mỗi loại có mục tiêu khác nhau nên không phải bất cứ loại mã hoá nào cũng có thể đảm bảo được toàn bộ cả bốn tính chất trên. Cho nên, chúng ta thường sử dụng kết hợp các loại mã hoá khác nhau nhằm đưa hệ thống đạt ngưỡng an toàn tốt nhất. Ví dụ, mã hoá bất đối xứng được sử dụng trong quá trình trao đổi secret key của mã hoá đối xứng.

* + 1. **Các loại encryption (mã hoá)**
       1. **Symmetric Encryption (mã hoá đối xứng)**



hay còn có tên khác là secret key encryption, là một loại kỹ thuật mã hoá sử dụng cùng một khoá cho cả quá trình mã hoá và giả mã. Do đó, người gửi và người nhận cần có một cách để chia sẻ và thống nhất khoá an toàn.

Một số ứng dụng của mã hoá đối xứng bao gồm:

* Che giấu dữ liệu: mã hoá đối xứng được dùng để che giấu các thông tin nhạy cảm, ví dụ như email, file.
* Chia sẻ tập tin: mã hoá đối xứng sử dụng như một cách che giấu thông tin an toàn khi chỉa sẻ trên mạng máy tính.

Ưu điểm lớn nhất của mã hoá đối xứng là tốc độ. Mã hoá đối xứng có tốc độ mã hoá và giải mã tốt hơn rất nhiều so với mã hoá bất đối xứng, nên nó thường được sử dụng để mã hoá file.

Nhược điểm lớn nhất của mã hoá đối xứng là cách chia sẻ key. Hai bên cần thống nhất sử dụng chung một key duy nhất một cách an toàn, điều này có thể gây ra một số rủi ro và có một số khó khăn nhất định.

**Một số loại mã hoá đối xứng phổ biến hiện nay bao gồm :**

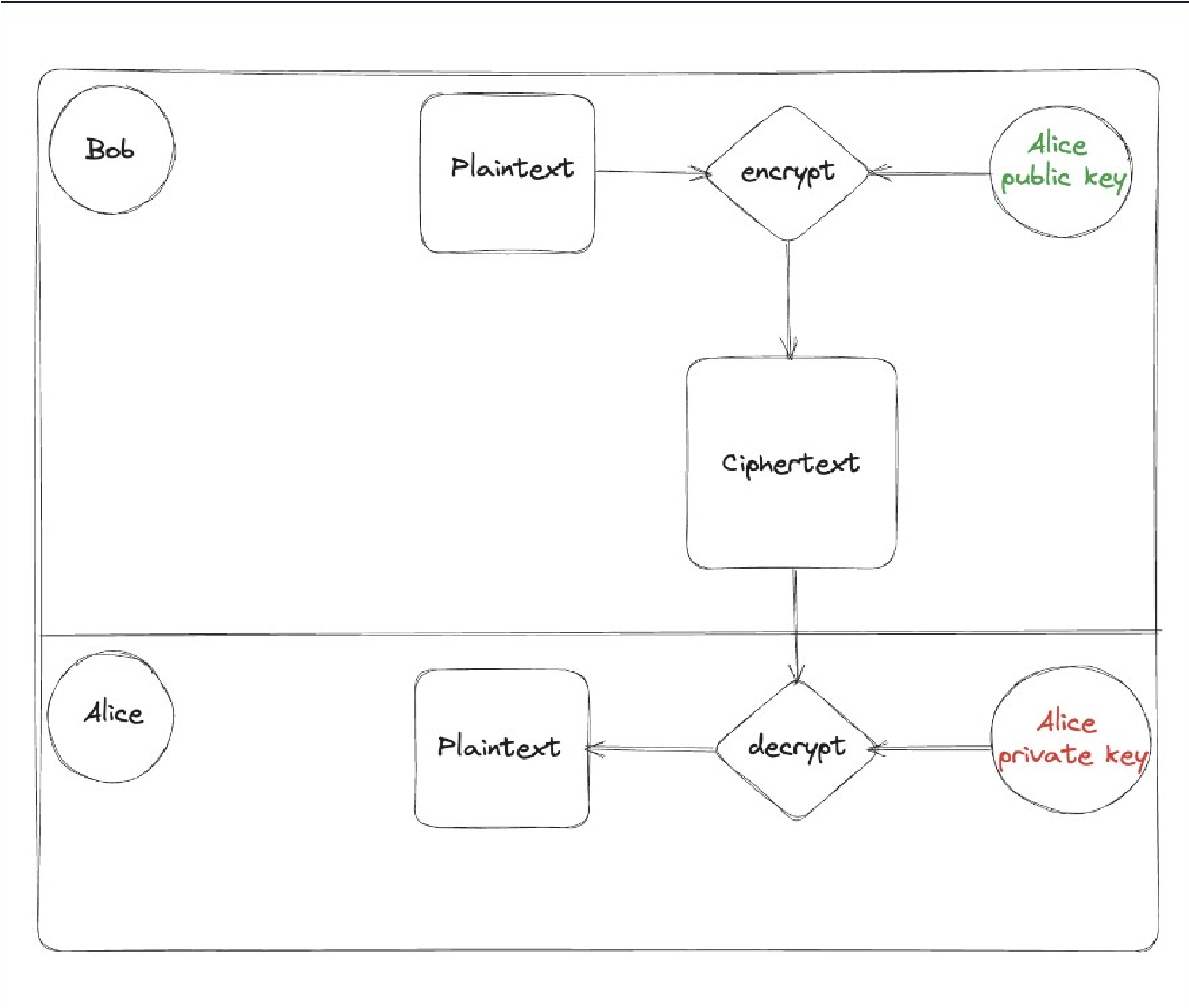
* Data Encryption Standard (DES): DES là một mật mã khối sử dụng khóa 56 bit. Nó được phát hành lần đầu tiên vào năm 1977 và hiện không còn được coi là an toàn vì bị phá vỡ bởi thuật toán vét cạn (exhautive search), mà lý do khoá tương đối ngắn (56 bit). Hiện nay đã bị thay thế bằng AES.
* Triple DES (3DES): 3DES là một biến thể của DES sử dụng ba khóa. 3DES có độ dài khóa 112 bit hoặc 168 bit, do đó, nó an toàn hơn DES. Nó an toàn hơn DES nhưng cũng chậm hơn. Nhìn chung, 3DES là một thuật toán mã hóa an toàn và hiệu quả, nhưng nó không còn được coi là lựa chọn tốt nhất cho các ứng dụng bảo mật hiện đại.
* Advanced Encryption Standard (AES): AES là một mật mã khối sử dụng khóa 128 bit, 192 bit hoặc 256 bit, được coi là thuật toán mã hoá đối xứng an toàn nhất hiện có. Nó được phát triển bởi National Institute of Standards and Technology (NIST), và được sử dụng phổ biến trên toàn thế giới hiện nay.
  + - 1. **Asymmetric Encryption (mã hoá bất đối xứng)**

Asymmetric Encryption, hay có tên khác là public-key encryption (mã hoá công khai) là một loại kỹ thuật mã hoá sử dụng 2 key cho cả quá trình mã hoá và giải mã.

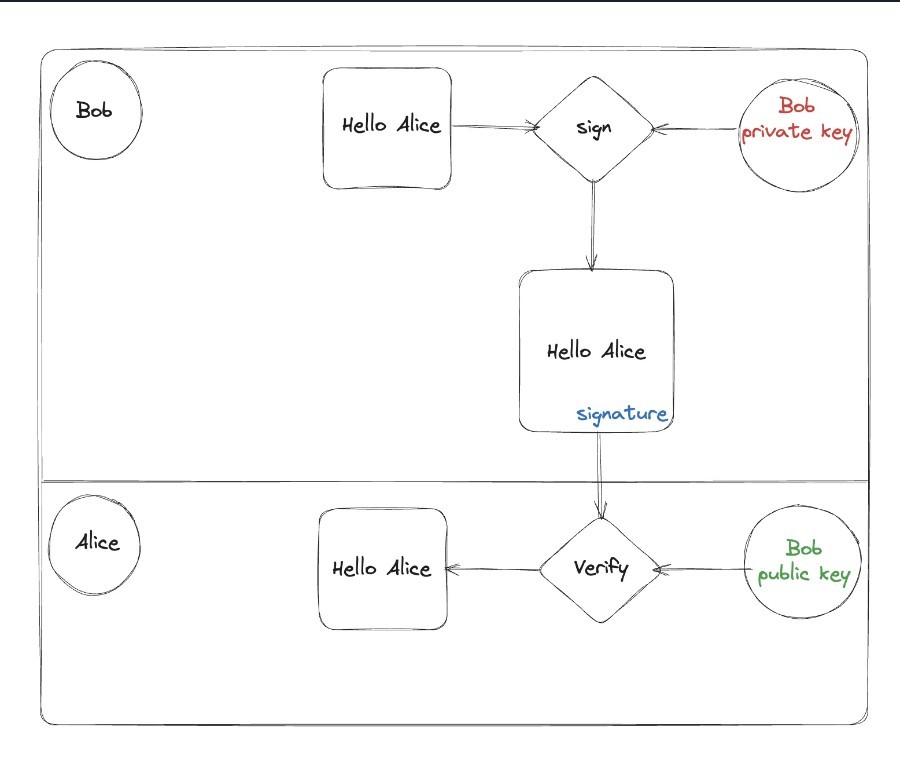
Hai key này đi thành cặp, bao gồm private key và public key. Private key sẽ được giữ bí mật, dùng để giải mã dữ liệu. Public key sẽ được chia sẻ với tất cả mọi người, dùng để dùng để mã hoá dữ liệu

**Tuỳ thuộc vào cách sử dụng key, ta có hai chức năng chính như sau:**

* Khi muốn trao đổi thông tin an toàn, sender sẽ sử dụng public key của receiver để encrypt thông tin và gửi đi. Sau đó, receiver sẽ lấy private key của mình để decrypt thông tin, khi đó, thông tin đã được trao đổi thành công và an toàn.



Trường hợp ngược lại sử dụng secret key để encrypt và public key để decrypt, ta gọi nó là ký số (sign), và thông điệp sau khi bị sign gọi là chữ ký số (digital signature) được đính kèm vào văn bản công khai mà bên gửi đã gửi cho các bên liên quan.

Ý nghĩa chính của digital signature là chỉ có người sở hữu secret key mới có thể sign được signature trên văn bản công khai, do đó, danh tính của sender được đảm bảo.

Ưu điểm lớn nhất của mã hoá bất đối xứng là cách thức trao đổi khoá rất an toàn. Hai bên chỉ cần gửi public key cho nhau để trao đổi mà không cần lo ngại về cách thức trao đổi khoá. Public key dù có bị các bên khác lấy được cũng không thể sử dụng.

Nhược điểm chính của mã hoá bất đối xứng là tốc độ. Mã hoá bất đối xứng chạy chậm nên chỉ phù hợp mã hoá các thông tin có kích thước nhỏ và không phù hợp để sử dụng trên các văn bản lớn.

**Một số mã hoá bất đối xứng phổ biến hiện nay:**

RSA: RSA là từ viết tắt của Rivest–Shamir–Adleman, tên của ba nhà khoa học máy tính đã phát triển nó vào năm 1978. RSA dựa trên độ khó của việc phân tích các số nguyên tố lớn. Mặc dù có tuổi đời rất lâu, RSA vẫn được coi là một trong những thuật toán mã hoá tốt nhất hiện nay.

các số nguyên tố lớn. Mặc dù có tuổi đời rất lâu, RSA vẫn được coi là một trong những thuật toán mã hoá tốt nhất hiện nay.

các số nguyên tố lớn. Mặc dù có tuổi đời rất lâu, RSA vẫn được coi là một trong những thuật toán mã hoá tốt nhất hiện nay.

* ElGamal: ElGamal là một thuật toán mã hóa bất đối xứng được phát triển bởi Taher Elgamal vào năm 1985. Nó dựa trên bài toán logarit rời rạc, một bài toán toán học phức tạp vẫn chưa có giải pháp hiệu quả. ElGamal được coi là an toàn như RSA, nhưng có hiệu suất tốt hơn.
* Diffie-Hellman: Diffie-Hellman là một thuật toán chia sẻ bí mật được phát triển bởi Whitfield Diffie và Martin Hellman vào năm 1976. Nó được sử dụng để tạo ra một khóa chung giữa hai bên mà không cần chia sẻ khóa đó trước đây. Giống với ElGamal, Diffie-Hellman là một thuật toán an toàn dựa trên bài toán logarit rời rạc.
  + - 1. **Cryptographic hash function (Hàm băm mật mã học)**

**Hash function** (hàm băm) là một hàm toán học cho phép chuyển một chuỗi ký tự có độ dài bất kỳ thành một chuỗi có độ dài cố định.

**Cryptographic hash function** (hàm băm băm mật mã học) là một hàm băm nhưng có thể một số tính chất phù hợp hơn với mật mã học.

Trong nhiều tài liệu, các tác giả cũng rút gọn crytographic hash function thành hash function. Bài viết này cũng sẽ rút gọn khái niệm trên thành hash function để tránh xung đột.

Chuỗi sau khi biến đổi gọi là hashed value (giá trị băm). Vì tính chất đặc thù này, chúng ta sẽ chỉ có một số lượng hữu hạn các hashed value và có thể xảy ra hiện tượng hai input trùng một hashed value.

**Các đặc tính của hash function bao gồm :**

* Uniformity (Tính đồng nhất): Một hashing function tốt cho phép mọi giá trị băm được tạo ra có xác suất giống nhau. Ngoài ra, uniformity còn có thể tính toán và đo lường được thông qua Chi-squared test.
* Efficiency (Tính hiệu quả): Hashing function nên dễ tính toán, tốc độ xử lý một chuỗi đầu vào cần có thời gian phù hợp.
* Determinism (Tính xác định): Luôn trả về một kết quả nếu input không đổi.
* Non-reversibility (Tính một chiều): Sẽ rất khó có thể đảo ngược từ hashed value sang input.
* Weak collision resistance (Tính kháng va chạm yếu): Giả sử có giá trị input x, khó có thể tìm ra một y != x với H(y) = H(x)
* Strong collision resistance (Tính kháng va chạm mạnh): Khó có thể tìm được một cặp x, y bất kỳ sao cho H(x) = H(y)

**Một số ứng dụng của hash function bao gồm:**

* Verifying the integrity (Đảm bảo tính toàn vẹn dữ liệu): Người ta thường sử dụng một mã hashed khi gửi kèm với dữ liệu gốc để đảm bảo rằng thông tin của sender toàn vẹn trong quá trình trao đổi.
* Signature generation and verification (Tạo, xác minh chữ ký): Khi sử dụng signature, người ta không sử dụng toàn bộ thông tin để ký mà chỉ sử dụng hashed value của thông tin đó.
* Password verification (Chứng thực mật khẩu): Mật khẩu trong cơ sở dữ liệu sẽ không bao giờ được lưu trữ ở plaintext, mà buộc phải thông qua các hashing function để lưu trữ.

**Một số hash function phổ biến bao gồm:**

* MD5: MD5 được Ronald Rivest thiết kế vào năm 1991 để thay thế hàm băm MD4 trước đó. Nó tạo ra hashed value có 128 bit. Hiện đã bị phá nên không còn được ưu tiên sử dụng.
* SHA0: Là phiên bản đầu tiên của nhóm thuật toán SHA, được công khai năm 1991. Tuy nhiên sau đó bị rút lại do một lỗi bảo mật nghiêm trọng không được công bố, và được thay thế bằng SHA1.
* SHA1: Được thiết kế bởi United States National Security Agency, SHA1 có thể tạo ra hashed value có 160 bit. Điểm yếu của SHA1 đã được công bố, vì thế, sau năm 2010 đã ít được sử dụng, các tiêu chuẩn cũng loại bỏ SHA1.
* SHA2: Là phiên bản nâng cấp của SHA1. SHA2 có thể tạo ra nhiều giá trị đầu ra, bao gồm: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2512/256. SHA2 vẫn còn sử dụng tốt cho đến ngày hôm nay.
* SHA3: Là phiên bản tốt nhất của nhóm thuật toán SHA. Mặc dù cùng nhóm với SHA, tuy nhiên cấu trúc bên trông SHA3 rất khác so với các thuật toán mã hoá trước đó, nó có phần giống với MD5.

Giống SHA2, SHA3 cũng cung cấp nhiều giá trị đầu vào, bao gồm: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHA3-512/224, và SHA3-512/256

* SHAKE: Một biến thể của SHA-3, SHAKE cho phép tạo ra các hàm băm có độ dài đầu ra tùy ý. SHAKE có hai biến thể: SHAKE-128 và SHAKE-256, cho phép tạo ra các hàm băm có độ dài đầu ra là 128 bit và 256 bit tương ứng.
* cSHAKE: Một biến thể của SHAKE cho phép thêm một số tùy chọn tùy chỉnh, chẳng hạn như độ dài của đầu ra, giá trị nonce và số vòng lặp. cSHAKE cũng là một biến thể của SHA3
* TupleHash: Một hàm băm mật mã được thiết kế để băm các mảng của các chuỗi.

TupleHash là một phiên bản khác của SHA3.

Các hash function SHAKE, cSHAKE và TupleHash đều được phát triển bởi NIST (National Institute of Standards and Technology) và đều được mô tả trong FIPS 202:

SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.

* + 1. **Một số lưu ý khi ứng dụng cryptography**
       1. **Cryptography**

Cryptography gần như không có điểm yếu nào nếu sử dụng đúng cách. Nhưng trong quá trình sử dụng, chúng ta có thể mắc một số lỗi cơ bản sau:

Sử dụng key quá yếu: Việc sử dụng một key quá yếu sẽ tạo điều kiện cho attacker có thể đoán, thậm trí dịch ngược được thông tin đã được encrypt.

Sử dụng cryptography đã lỗi thời: Tránh việc sử dụng các cryptography đã cũ vì chúng không còn đủ khả năng bảo vệ.

Bảo quản secret key kém: Việc để lộ, làm mất secret key là việc tối kỵ. Cần cócác chính sách bảo quản các secret key thật tốt.

Không cập nhật key: Một secret key chỉ đủ để bảo vệ thông tin trong một khoảng thời gian nhất định, do đó chúng ta cần thay thế các key cũ trong quá trình trao đổi thông tin.

* + - 1. **Hash function**

Cũng giống như cryptography, hash function gần như không có điểm yếu nếu sử dụng đúng cách. Tuy nhiên, ta cần lưu ý một số trường hợp cụ thể sau:

Sử dụng hashed value để lưu trữ mật khẩu: Dù không thể bị dịch ngược, nhưng attacker vẫn có thể tấn công bằng kỹ thuật dictionary attack, thậm trí brute-force attack nhằm tiếp cận password.

Sử dụng hashed value làm ID: Việc định danh dữ liệu bằng hashed value tiềm ẩn rủi ro xung đột (collision) vì hai dữ liệu trùng một hashed value là hoàn toàn có thể xảy ra. Chúng ta có thể cải thiện hiệu suất tìm kiếm bằng hashed value, nhưng không nên sử dụng nó như một định danh.

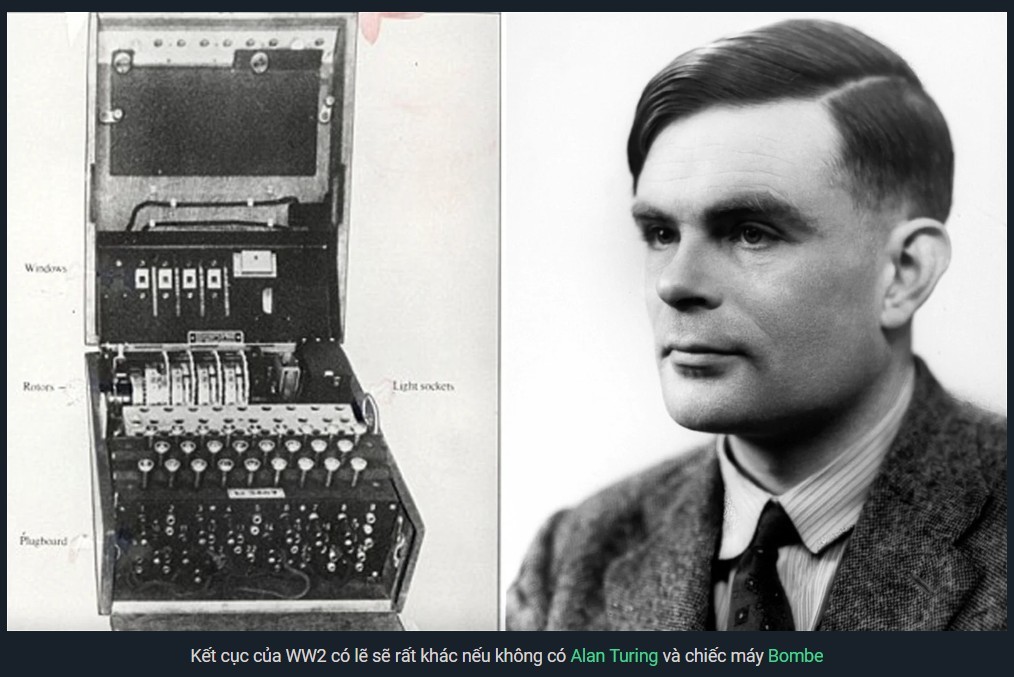
* + 1. **Kết luận về mật mã học**

Các hàm mã hoá rất mạnh mẽ, dễ sử dụng và đáng tin cậy. Nếu sử dụng đúng cách, hệ thống của chúng ta sẽ có một lớp phòng thủ vững chắc, hoàn toàn có thể chống lại sự xâm phạm của kẻ tấn công.

Tuy nhiên, nếu sử dụng không đúng chuẩn, sai cách sử dụng sẽ để lại lỗ hổng bảo mật cực lớn đến hệ thống, dữ liệu, thậm trí máy chủ của chúng ta đều có thể bị đánh cắp, chiếm đoạt hoặc phá hoại.

Ngoài ra, chúng ta cần thường xuyên cập nhật các phần mềm, tránh trường hợp các giao thức, thậm trí các hàm mã hoá mà chúng ta sử dụng không còn hiệu quả.

Trong chiến tranh thế giới thứ hai, cỗ máy Enigma được coi là bất khả chiến bại thời bấy giờ, đã bị cỗ máy Bombe do Alan Turing phát minh giải mã thành công, giúp một phần không nhỏ cho quân Đồng Minh chiến thắng Phát xít Đức.



Theo khuyến nghị của 200lab, chúng ta nên sử dụng các hàm mã hoá theo các tiêu chuẩn khuyến nghị có sẵn, ví dụ như RFCs, để giảm thiểu tối đa nguy cơ và rủi ro bảo mật. Rất nhiều các tiêu chuẩn của RFCs, trong đó có JWT, HTTPs, bcrypt đã phổ biến và được tin dùng.

### 1.2. Thuật toán One-Time Pad (OTP)

**1.2.1. Giới thiệu về One Time Pad**

**1.2.1.1. Giới thiệu chung**

One-Time Pad (OTP) là một trong những thuật toán mã hóa đối xứng cổ điển nhưng có tầm ảnh hưởng lớn trong lịch sử mật mã học. Thuật toán này được phát minh vào năm 1917 bởi Gilbert Vernam, sau đó được hoàn thiện bởi Joseph Mauborgne, và thường được biết đến dưới tên gọi **mã Vernam**. OTP đặc biệt quan trọng bởi nó là **thuật toán duy nhất được chứng minh có khả năng bảo mật tuyệt đối (perfect secrecy)** theo định nghĩa của Claude Shannon – cha đẻ của lý thuyết thông tin.

Trong OTP, thông điệp (plaintext) được kết hợp với một khóa (key) có độ dài bằng chính thông điệp đó, được sinh ra hoàn toàn ngẫu nhiên và chỉ sử dụng một lần duy nhất. Nếu các điều kiện này được đảm bảo, kẻ tấn công sẽ không thể phân biệt được đâu là bản rõ đúng, vì mọi chuỗi ký tự đều có xác suất xuất hiện như nhau.

Mặc dù có độ an toàn tuyệt đối, nhưng OTP không được sử dụng phổ biến trong các hệ thống thông tin hiện đại, nguyên nhân chủ yếu xuất phát từ vấn đề **phân phối và quản lý khóa**. Tuy nhiên, OTP vẫn được xem là “chuẩn vàng” để so sánh với các hệ thống mật mã khác về mức độ an toàn.

**1.2.1.2. Nguyên lý hoạt động**

Nguyên lý hoạt động của OTP dựa trên phép toán logic XOR (⊕) hoặc phép cộng modulo. Quy trình cơ bản gồm hai bước: mã hóa và giải mã.  
Mã hóa:  
Với mỗi ký tự hoặc bit trong bản rõ P, ta kết hợp với khóa K bằng phép XOR để thu được bản mã C:

Cᵢ = Pᵢ ⊕ Kᵢ

Giải mã:  
Người nhận thực hiện lại phép XOR giữa bản mã và khóa để khôi phục bản rõ:

Pᵢ = Cᵢ ⊕ Kᵢ  
Do tính chất:

(A ⊕ B) ⊕ B = A

Lưu ý:  
Bản rõ luôn được phục hồi chính xác nếu khóa K trùng khớp.  
Ví dụ: nếu bản mã thu được là EQNVZ, thì khi áp dụng phép giải mã với đúng khóa, ta sẽ khôi phục lại bản rõ ban đầu.  
Trong trường hợp OTP được biểu diễn ở dạng nhị phân, ta chỉ cần áp dụng phép XOR giữa chuỗi bit của bản rõ và khóa.

**1.2.1.3. Các thành phần chính**

Một hệ thống OTP gồm các thành phần cơ bản sau:

* Bản rõ (Plaintext): Thông điệp gốc mà người gửi muốn bảo vệ.
* Khóa (Key): Chuỗi ngẫu nhiên có cùng độ dài với bản rõ. Đây là thành phần quan trọng nhất, quyết định tính an toàn của OTP.
* Thuật toán XOR (⊕): Phép toán đơn giản để kết hợp khóa với bản rõ, đồng thời cũng được dùng trong quá trình giải mã.
* Bản mã (Ciphertext): Kết quả sau khi mã hóa, có dạng giống chuỗi ngẫu nhiên và không tiết lộ thông tin gì về bản rõ.
* Đặc biệt, điều kiện tiên quyết là khóa phải được sinh ra hoàn toàn ngẫu nhiên, dài bằng bản rõ, giữ bí mật tuyệt đối và chỉ được sử dụng một lần.

**1.2.1.4. Ưu điểm và nhược điểm**

**Ưu điểm:**

* Bảo mật tuyệt đối: Đây là điểm mạnh nhất của OTP. Claude Shannon đã chứng minh rằng nếu các điều kiện của OTP được tuân thủ, thì bản mã không chứa bất kỳ thông tin nào giúp suy đoán bản rõ.
* Đơn giản**:** Thuật toán chỉ dựa trên phép XOR hoặc cộng modulo, dễ lập trình và triển khai.
* Không thể phá vỡ bằng thống kê: Vì bản mã có phân bố hoàn toàn ngẫu nhiên, nên không thể khai thác đặc điểm tần suất để phân tích như các thuật toán cổ điển khác (ví dụ Caesar hay Vigenère).

**Nhược điểm:**

* Vấn đề khóa: Phải có một khóa ngẫu nhiên, có độ dài bằng bản rõ, và khóa này phải được chia sẻ an toàn giữa hai bên. Đây là nhược điểm lớn nhất, vì việc phân phối khóa trong thực tế là cực kỳ khó khăn.
* Quản lý khóa: Nếu cần truyền nhiều thông điệp, ta phải có số lượng khóa khổng lồ tương ứng.
* Nguy cơ tái sử dụng khóa: Nếu một khóa được sử dụng cho hai thông điệp khác nhau, kẻ tấn công có thể khai thác mối liên hệ giữa các bản mã để tìm ra bản rõ.
* Khó áp dụng trong môi trường hiện đại: Với lượng dữ liệu khổng lồ như ngày nay, OTP không còn thực tế vì yêu cầu khóa phải có độ dài tương ứng.

**1.2.1.5. Ứng dụng của giải thuật**

Mặc dù ít được dùng trong hệ thống mật mã hiện đại, OTP vẫn có ứng dụng trong một số lĩnh vực đặc thù:

* Liên lạc ngoại giao và quân sự: Trước đây, các điệp viên hoặc cơ quan ngoại giao thường được phát các “pad giấy” chứa chuỗi số ngẫu nhiên. Sau khi dùng một khóa, trang giấy sẽ bị hủy để tránh lộ lọt.
* Chiến tranh Thế giới II và Chiến tranh Lạnh: OTP được sử dụng trong các kênh liên lạc mật nhờ tính bảo mật tuyệt đối.
* Ứng dụng trong vệ tinh và hệ thống liên lạc bảo mật tầm xa: OTP được triển khai khi việc phân phối khóa có thể được kiểm soát chặt chẽ.
* Trong lý thuyết mật mã: OTP vẫn là nền tảng quan trọng để minh chứng cho khái niệm bảo mật tuyệt đối, và để đánh giá mức độ an toàn của các thuật toán mã hóa hiện đại như AES hoặc RSA.

### 1.3. AES – Thuật toán mã hóa bổ sung

**1.3.1. Tổng quan**

AES (Advanced Encryption Standard) là một trong những chuẩn mã hóa đối xứng hiện đại quan trọng nhất hiện nay. Thuật toán này được Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) chuẩn hóa vào năm 2001 nhằm thay thế chuẩn DES (Data Encryption Standard) vốn đã trở nên lỗi thời trước sự phát triển của năng lực tính toán.

AES được thiết kế bởi Joan Daemen và Vincent Rijmen, ban đầu có tên gọi Rijndael. Sau quá trình đánh giá nghiêm ngặt, Rijndael được chọn làm chuẩn mã hóa mới và được đặt tên chính thức là AES.

Với khả năng bảo mật mạnh mẽ, hiệu năng cao và tính linh hoạt, AES hiện được sử dụng rộng rãi trong hầu hết các hệ thống bảo mật thông tin, từ mã hóa dữ liệu cá nhân, giao dịch tài chính, truyền thông mạng, cho đến bảo vệ thông tin mật của chính phủ và quân sự.

**1.3.2 Nguyên tắc hoạt động**

AES là một thuật toán mã hóa khối đối xứng (block cipher), hoạt động trên khối dữ liệu có độ dài cố định 128 bit. Thuật toán hỗ trợ ba độ dài khóa khác nhau: **128 bit, 192 bit và 256 bit**.

* Với **AES-128**: thực hiện 10 vòng lặp (rounds).
* Với **AES-192**: thực hiện 12 vòng lặp.
* Với **AES-256**: thực hiện 14 vòng lặp.

Trong mỗi vòng lặp, dữ liệu trải qua nhiều phép biến đổi toán học phi tuyến nhằm tăng cường tính an toàn, khiến kẻ tấn công khó có thể tìm ra mối quan hệ giữa bản rõ và bản mã.

Điểm mạnh của AES nằm ở chỗ nó không chỉ đảm bảo tính bảo mật cao mà còn được tối ưu để triển khai trên cả phần mềm và phần cứng, cho phép hoạt động nhanh và hiệu quả.

**1.3.3 Các bước hoạt động chính**

Trong mỗi vòng lặp (ngoại trừ vòng cuối), AES áp dụng bốn bước cơ bản:

1. **SubBytes (Thay thế byte):** Mỗi byte trong khối dữ liệu được thay thế bằng một giá trị khác dựa trên bảng S-box (hộp thay thế). Đây là bước phi tuyến tính nhằm tăng độ phức tạp và khả năng chống lại các tấn công phân tích.
2. **ShiftRows (Dịch hàng):** Các hàng của ma trận dữ liệu (4x4) được dịch chuyển tuần tự: hàng thứ nhất giữ nguyên, hàng thứ hai dịch 1 byte sang trái, hàng thứ ba dịch 2 byte, và hàng thứ tư dịch 3 byte. Điều này giúp trộn lẫn dữ liệu theo chiều ngang.
3. **MixColumns (Trộn cột):** Mỗi cột của ma trận dữ liệu được biến đổi bằng một phép toán đại số trên trường hữu hạn GF(2^8). Bước này trộn dữ liệu theo chiều dọc, làm gia tăng mức độ khuếch tán.
4. **AddRoundKey (Cộng khóa vòng):** Khối dữ liệu được kết hợp với khóa con (round key) sinh ra từ khóa chính bằng phép XOR. Đây là bước gắn trực tiếp sự an toàn của thuật toán với khóa mật mã.

* Ở vòng đầu tiên: chỉ thực hiện **AddRoundKey**.
* Ở các vòng giữa: thực hiện cả 4 bước trên.
* Ở vòng cuối: bỏ qua bước **MixColumns**.

Quá trình giải mã AES về nguyên tắc là nghịch đảo của các bước mã hóa, với các thao tác **InvSubBytes, InvShiftRows, InvMixColumns** và **AddRoundKey**.

**1.3.4. Vai trò trong hệ thống lai với OTP**

Mặc dù One-Time Pad (OTP) mang lại tính bảo mật tuyệt đối về mặt lý thuyết, nhưng như đã phân tích, OTP gặp hạn chế rất lớn trong việc quản lý và phân phối khóa. Do đó, trong các hệ thống mật mã hiện đại, người ta thường **kết hợp OTP với các thuật toán mã hóa khối mạnh mẽ như AES** để tạo thành một hệ thống lai, vừa tận dụng được tính bảo mật tuyệt đối của OTP trong lý thuyết, vừa khắc phục được hạn chế của nó trong thực tiễn.

Một ứng dụng điển hình của AES trong hệ thống lai với OTP là:

* AES được dùng để bảo mật khóa hoặc mật khẩu người dùng trước khi lưu trữ hoặc phân phối. Khi người dùng nhập mật khẩu, hệ thống sẽ mã hóa mật khẩu bằng AES trước khi lưu lại. Nhờ đó, ngay cả khi cơ sở dữ liệu bị lộ, kẻ tấn công cũng không thể dễ dàng khôi phục mật khẩu gốc.
* OTP sau đó có thể được sử dụng để mã hóa thông tin nhạy cảm (như file dữ liệu hoặc nội dung truyền tin) bằng các khóa ngẫu nhiên được bảo vệ an toàn nhờ AES.

Cách tiếp cận này cho phép OTP giữ được tính an toàn lý thuyết, trong khi AES đóng vai trò cầu nối thực tiễn, giúp quản lý khóa và mật khẩu hiệu quả, đồng thời bảo vệ hệ thống trước các tấn công phổ biến hiện nay.

### 1.4. Nén dữ liệu với Huffman Coding

**1.4.1. Khái niệm**

**Huffman Coding** là một trong những thuật toán nén dữ liệu phổ biến nhất trong lĩnh vực mã hóa độ dài thay đổi (variable-length coding). Thuật toán được phát minh bởi David A. Huffman vào năm 1952, dựa trên nguyên tắc gán cho mỗi ký tự một mã nhị phân có độ dài khác nhau tùy theo tần suất xuất hiện.

Ý tưởng chính là: các ký tự xuất hiện với tần suất cao sẽ được gán mã ngắn hơn, trong khi những ký tự ít xuất hiện sẽ được gán mã dài hơn. Điều này giúp giảm tổng số bit cần thiết để biểu diễn toàn bộ thông tin, từ đó nén dữ liệu hiệu quả mà không làm mất mát thông tin.

Huffman Coding là nền tảng của nhiều chuẩn nén dữ liệu hiện nay, chẳng hạn như **DEFLATE** (dùng trong ZIP, GZIP), nén hình ảnh **JPEG**, hoặc nén văn bản trong nhiều hệ thống.

**1.4.2. Nguyên lý hoạt động**

Quy trình xây dựng mã Huffman thường gồm các bước sau:

1. **Tính tần suất xuất hiện của các ký tự** trong thông điệp cần nén.
2. **Tạo cây Huffman:**

* Mỗi ký tự được biểu diễn dưới dạng một nút lá với trọng số bằng tần suất xuất hiện.
* Hai nút có trọng số nhỏ nhất được ghép lại thành một nút cha với trọng số bằng tổng trọng số hai nút con.
* Lặp lại quá trình cho đến khi chỉ còn một nút gốc.

1. **Gán mã nhị phân cho mỗi ký tự:**
   * Duyệt cây Huffman từ gốc xuống lá, gán 0 cho nhánh trái và 1 cho nhánh phải (hoặc ngược lại).
   * Các ký tự xuất hiện thường xuyên hơn sẽ nằm gần gốc → có mã ngắn.
   * Các ký tự hiếm sẽ nằm sâu trong cây → có mã dài.

**1.4.3. Ứng dụng trong hệ thống lai với OTP**

Trong hệ thống kết hợp **One-Time Pad (OTP)** với các kỹ thuật hiện đại, Huffman Coding có thể được sử dụng để **nén khóa OTP sau khi sinh ra**.

Vì khóa OTP thường rất dài (có độ dài bằng đúng thông điệp), việc lưu trữ và phân phối khóa gây ra khó khăn. Khi áp dụng Huffman Coding:

* Khóa được biểu diễn dưới dạng chuỗi nhị phân nén → giảm dung lượng cần lưu trữ hoặc truyền tải.
* Điều này làm tăng hiệu quả hệ thống, đặc biệt trong môi trường tài nguyên hạn chế hoặc truyền thông mạng tốc độ thấp.

**1.4.4. Ưu điểm và nhược điểm**

**Ưu điểm:**

* Tiết kiệm dung lượng: Giảm kích thước dữ liệu cần lưu trữ hoặc truyền đi.
* Tăng hiệu quả: Giúp quá trình quản lý và phân phối khóa OTP trong hệ thống trở nên nhẹ hơn.
* Không mất mát thông tin: Huffman Coding thuộc nhóm lossless compression, nên khóa hoặc dữ liệu sau giải nén hoàn toàn giống bản gốc.

**Nhược điểm:**

* Tốn thêm thời gian xử lý: Quá trình xây dựng cây Huffman, mã hóa và giải mã phức tạp hơn so với dữ liệu thô.
* Phụ thuộc vào tần suất: Hiệu quả nén cao khi dữ liệu có phân bố tần suất không đều, nhưng với dữ liệu ngẫu nhiên (như OTP), mức độ nén có thể không đáng kể.
* Cần bổ sung thông tin giải mã: Hệ thống phải truyền kèm cấu trúc cây Huffman hoặc bảng mã, làm tăng chi phí xử lý.

### 1.5. Hệ thống lai OTP – AES – Huffman

Trong bối cảnh dữ liệu ngày càng lớn và nhu cầu bảo mật ngày càng cao, việc kết hợp nhiều giải thuật để tận dụng ưu điểm và hạn chế nhược điểm của từng thuật toán là một hướng đi hợp lý. Một trong những mô hình điển hình là **hệ thống lai giữa One-Time Pad (OTP), AES và Huffman Coding**.

#### 1.5.1. Quy trình mã hóa

1. **Sinh khóa ngẫu nhiên**
   * Một khóa OTP được tạo ngẫu nhiên, có thể dưới dạng số nguyên (integer key) hoặc chuỗi ASCII.
   * Khóa này có độ dài bằng dữ liệu gốc, đảm bảo tính toàn vẹn của phương pháp OTP.
2. **Mã hóa dữ liệu bằng OTP**
   * Thực hiện phép XOR giữa dữ liệu gốc (plaintext) và khóa OTP để sinh ra bản mã tạm thời (ciphertext).
   * Bản mã OTP đảm bảo tính bảo mật tuyệt đối, nhưng khó lưu trữ và phân phối khóa.
3. **Bảo vệ mật khẩu người dùng bằng AES**
   * Người dùng cung cấp một mật khẩu đăng nhập hệ thống.
   * Mật khẩu này được mã hóa bằng AES để tránh rò rỉ thông tin khi lưu trữ hoặc truyền tải.
   * AES đảm bảo mật khẩu có thể được quản lý an toàn, không bị tấn công dò tìm hay giải mã dễ dàng.
4. **Nén khóa OTP bằng Huffman**
   * Do OTP có độ dài bằng dữ liệu gốc nên dung lượng lưu trữ khóa rất lớn.
   * Áp dụng Huffman Coding để giảm kích thước khóa, giúp tiết kiệm bộ nhớ và tăng hiệu quả truyền tải.

#### 1.5.2 Quy trình giải mã

1. **Giải nén khóa OTP**
   * Sử dụng thuật toán Huffman để khôi phục lại khóa OTP từ bản nén.
2. **Giải mã mật khẩu bằng AES**
   * Mật khẩu đã mã hóa bằng AES được giải mã để xác thực quyền truy cập.
3. **Khôi phục dữ liệu bằng OTP**
   * Thực hiện lại phép XOR giữa bản mã OTP và khóa OTP đã được giải nén để thu được dữ liệu gốc.

#### 1.5.3 Ưu điểm của hệ thống lai

* Bảo mật tối đa: OTP cung cấp tính bảo mật tuyệt đối về lý thuyết.
* An toàn trong quản lý mật khẩu: AES giúp mã hóa mật khẩu người dùng, ngăn chặn tấn công từ bên ngoài.
* Tiết kiệm dung lượng: Huffman giảm kích thước khóa OTP, giúp hệ thống vận hành hiệu quả hơn.
* Kết hợp hài hòa: Mỗi thuật toán giải quyết một vấn đề cụ thể (OTP cho dữ liệu, AES cho mật khẩu, Huffman cho dung lượng).

#### 1.5.4 Hạn chế

* Hiệu năng thấp với dữ liệu lớn: OTP cần khóa có độ dài bằng dữ liệu, gây khó khăn khi xử lý tệp tin dung lượng cao.
* Nhiều bước xử lý: Việc kết hợp 3 giải thuật làm tăng độ phức tạp, cần nhiều tài nguyên tính toán.
* Khó triển khai thực tế: Dù đảm bảo tính bảo mật, nhưng hệ thống này khó áp dụng ở quy mô lớn do vấn đề phân phối khóa OTP.

### 1.6. Cách thuật toán hoạt động.

#### 1.6.1. Quy trình mã hóa thuật toán OTP

**Các bước thực hiện mã hóa:**

- Người dùng nhập file gốc cần mã hóa.

- Hệ thống sinh khóa ngẫu nhiên từ số nguyên hoặc bảng mã ASCII.

- Thuật toán Vernam (XOR) được áp dụng để mã hóa dữ liệu bằng khóa ngẫu nhiên.

- Tạo ma trận vị trí lưu trữ mật khẩu và tạo khóa key dưới dạng Hex.

- Mật khẩu được mã hóa bằng thuật toán AES và đưa vào ma trận.

- Nếu người dùng chọn chế độ nén, khóa sẽ được nén bằng Huffman để giảm dung lượng.

- Cuối cùng, các ma trận được ghép lại thành một ma trận lớn để hình thành file mã hóa.

#### 1.6.2. Quy trình giải mã thuật toán OTP

**Các bước thực hiện giải mã:**

- Người dùng nhập file đã mã hóa.

- Hệ thống tìm vị trí mật khẩu gốc và mật khấu Key dạng Hex trong ma trận.

- Giải mã mật khẩu bằng thuật toán AES.

- Nếu mật khẩu trùng khớp với mật khẩu người dùng nhập, hệ thống tái tạo ma trận khóa và ma trận bản mã.

- Nếu file được nén trước đó, hệ thống tiến hành giải nén khóa bằng Huffman.

- Áp dụng thuật toán Vernam giải mã để khôi phục dữ liệu gốc.

- Xuất ra file văn bản thuần (Plain Text File).

**Ghi chú an toàn.** Mật khẩu đóng vai trò cửa kiểm soát: không đúng mật khẩu thì không truy cập được cả khóa OTP lẫn dữ liệu. Khóa OTP không lộ nguyên dạng vì được gói chung và (có thể) nén; vị trí password được che giấu nhờ **M\_pos**.

# CHƯƠNG 2: CÔNG NGHỆ JAMSTACK

### 2.1. Khái niệm JAMstack

JAMstack là một kiến trúc hiện đại trong phát triển web, được xây dựng dựa trên ba yếu tố cốt lõi: JavaScript, API và Markup. Thay vì phụ thuộc vào các máy chủ truyền thống xử lý toàn bộ quá trình sinh trang, JAMstack tách biệt giao diện tĩnh (được build sẵn) với các dịch vụ động phía sau (API, serverless functions).

Điểm đặc trưng của JAMstack là toàn bộ nội dung tĩnh (HTML, CSS, JavaScript) được build trước (pre-rendering) và phân phối qua CDN, trong khi các chức năng động như xác thực, xử lý thanh toán, tìm kiếm… được đảm nhiệm bởi API hoặc dịch vụ serverless. Cách tiếp cận này làm cho website nhanh hơn, bảo mật hơn và dễ mở rộng.

Nếu các ứng dụng web truyền thống thường phải tải dữ liệu động từ máy chủ mỗi khi có yêu cầu mới, thì JAMstack đưa toàn bộ nội dung có thể dự đoán trước ra phía CDN, giúp người dùng nhận phản hồi gần như tức thì. Chính nhờ sự kết hợp này mà JAMstack được coi là “làn sóng mới” trong phát triển web hiện nay.

### 2.2. Nguyên lý hoạt động

Nguyên lý hoạt động của JAMstack dựa trên sự phân tách rõ ràng giữa phần hiển thị (frontend) và phần logic nghiệp vụ (backend).

1. Pre-rendering nội dung các trang web được build sẵn thành file HTML tĩnh nhờ các công cụ SSG (Static Site Generators). Việc này giúp rút ngắn thời gian tải trang, đồng thời tăng khả năng cache trên CDN.
2. Sử dụng JavaScript phía client JavaScript được dùng để tăng tính tương tác, ví dụ: xử lý form, điều hướng không tải lại trang (SPA), kết nối tới API để lấy dữ liệu động.
3. Tích hợp qua API các tính năng động như xác thực người dùng, bình luận, thanh toán online hay lưu trữ dữ liệu đều được tách riêng dưới dạng API (REST hoặc GraphQL). Điều này tạo nên sự linh hoạt: frontend có thể kết nối với nhiều dịch vụ backend khác nhau mà không bị phụ thuộc vào một hệ thống duy nhất.
4. Phân phối qua CDN sau khi được build, toàn bộ website sẽ được deploy lên CDN (Content Delivery Network). CDN giúp phân phối nội dung từ server gần người dùng nhất, giảm độ trễ và cải thiện hiệu năng.

### 2.3. Ưu điểm của JAMstack

JAMstack mang lại nhiều ưu điểm vượt trội so với kiến trúc truyền thống:

* Hiệu năng cao: Nội dung tĩnh được phục vụ trực tiếp từ CDN, loại bỏ thời gian xử lý phía server.
* Bảo mật: Do không có máy chủ ứng dụng phức tạp, bề mặt tấn công giảm đáng kể.
* Khả năng mở rộng: Website có thể phục vụ hàng triệu người dùng mà không cần nâng cấp hạ tầng phức tạp.
* Trải nghiệm phát triển hiện đại: Các lập trình viên frontend có thể tập trung vào giao diện, trong khi backend được chia nhỏ thành các dịch vụ API dễ tái sử dụng.
* Chi phí thấp: Hosting nội dung tĩnh rẻ hơn nhiều so với việc duy trì server động.

### 2.4. Hạn chế và thách thức

Mặc dù JAMstack có nhiều ưu điểm, song nó cũng tồn tại một số hạn chế:

* Phức tạp trong tích hợp API: Việc phụ thuộc vào nhiều dịch vụ bên ngoài có thể làm gia tăng độ phức tạp trong quản lý.
* Khó khăn với nội dung thay đổi thường xuyên: Nếu website cần cập nhật dữ liệu liên tục (ví dụ: mạng xã hội, sàn giao dịch chứng khoán), thì việc build lại toàn bộ site có thể tốn thời gian.
* Đòi hỏi kỹ năng đa dạng: Nhà phát triển cần hiểu cả frontend hiện đại, API, và quản lý dịch vụ serverless.
* Chi phí dịch vụ thứ ba: Mặc dù hosting rẻ, nhưng nếu lạm dụng dịch vụ API trả phí, tổng chi phí có thể tăng cao.

### 2.5. Các công cụ và framework hỗ trợ JAMstack

Sự thành công của JAMstack gắn liền với sự phát triển mạnh mẽ của hệ sinh thái công cụ hỗ trợ. Có thể chia thành một số nhóm chính như sau:

(a) Static Site Generators (SSG)

* Gatsby: Dựa trên React, tối ưu cho SEO và blog, có hệ thống plugin phong phú.
* Hugo: Viết bằng Go, nổi tiếng với tốc độ build cực nhanh, phù hợp với website lớn.
* Jekyll: Viết bằng Ruby, đơn giản và phổ biến trong cộng đồng GitHub Pages.

(b) Hybrid Frameworks

* Next.js: Cho phép kết hợp SSR, SSG, ISR, được ưa chuộng trong các dự án React hiện đại.
* Nuxt.js: Tương tự Next.js nhưng dành cho Vue, dễ dàng phát triển ứng dụng đa năng.

(c) Hosting & CDN

* Netlify, Vercel, Cloudflare Pages: Cung cấp dịch vụ deploy nhanh chóng, tích hợp CI/CD, hỗ trợ domain và HTTPS mặc định.

(d) Serverless & API services

* AWS Lambda, Firebase, Supabase: Giúp triển khai backend động mà không cần máy chủ chuyên dụng.

Chính nhờ hệ sinh thái phong phú này, JAMstack mới có thể phát triển và lan tỏa nhanh chóng.

### 2.6. Ứng dụng thực tế của JAMstack

JAMstack ngày càng được áp dụng rộng rãi trong nhiều lĩnh vực:

* Blog cá nhân và trang portfolio: Các SSG như Gatsby hoặc Hugo được sử dụng để tạo blog siêu nhanh, dễ deploy.
* Trang thương mại điện tử: JAMstack kết hợp với các API thanh toán (Stripe, PayPal) và giỏ hàng (Snipcart) để xây dựng shop online nhanh, bảo mật.
* Website doanh nghiệp: Nhờ tốc độ tải nhanh và bảo mật cao, JAMstack phù hợp với website giới thiệu sản phẩm, dịch vụ.
* Ứng dụng SaaS: Giao diện frontend được xây dựng tách biệt, trong khi backend được triển khai bằng API serverless.

### 2.7. So sánh JAMstack với mô hình truyền thống

Để thấy rõ lợi ích của JAMstack, cần đặt nó trong tương quan với mô hình kiến trúc truyền thống (monolithic hoặc CMS-based như WordPress).

* Hiệu năng: JAMstack vượt trội do phục vụ qua CDN. Trong khi đó, WordPress phải query database mỗi khi tải trang.
* Bảo mật: Website WordPress dễ bị tấn công SQL Injection, XSS, trong khi JAMstack giảm bề mặt tấn công.
* Khả năng mở rộng: JAMstack dễ dàng scale nhờ CDN, còn mô hình truyền thống cần mở rộng server.
* Quy trình phát triển: JAMstack hiện đại hơn, phù hợp với CI/CD và DevOps, trong khi CMS truyền thống khó tích hợp pipeline.

### 2.8. Xu hướng phát triển của JAMstack

Trong tương lai, JAMstack được dự báo sẽ tiếp tục mở rộng nhờ những yếu tố sau:

* Gia tăng hỗ trợ từ các nền tảng lớn: Vercel, Netlify, Cloudflare không ngừng cải tiến dịch vụ, mở rộng khả năng deploy và tích hợp.
* Kết hợp với công nghệ mới: ISR (Incremental Static Regeneration), Edge Functions và AI sẽ giúp JAMstack linh hoạt hơn.
* Tối ưu trải nghiệm nhà phát triển: Các công cụ quản lý API, dịch vụ serverless sẽ ngày càng thân thiện, giúp giảm độ phức tạp.
* Ứng dụng rộng rãi trong thương mại điện tử và SaaS: Nhờ khả năng mở rộng nhanh, JAMstack sẽ tiếp tục chiếm lĩnh lĩnh vực này.

**Nhiều chuyên gia dự đoán rằng JAMstack sẽ trở thành kiến trúc chủ đạo cho các ứng dụng web hiện đại, đặc biệt trong bối cảnh nhu cầu về tốc độ, bảo mật và trải nghiệm người dùng ngày càng cao.**

# CHƯƠNG 3: XÂY DỰNG ỨNG DỤNG MINH HỌA

### 3.1 Thiết kế giao diện ứng dụng

Ứng dụng được xây dựng trên nền tảng web, nhằm mục tiêu tạo ra một công cụ mã hóa và giải mã đơn giản, trực quan, dễ sử dụng nhưng vẫn đảm bảo tính an toàn theo mô hình thuật toán One-Time Pad kết hợp AES và Huffman ( nếu chọn).  
Giao diện được thiết kế dựa trên các công nghệ:

**HTML5**: dùng để xây dựng cấu trúc nội dung trang web, tổ chức bố cục các khối giao diện.

**CSS3 kết hợp TailwindCSS**: hỗ trợ định dạng và làm đẹp giao diện. Nhờ TailwindCSS, việc thiết kế trở nên gọn gàng, responsive, tương thích với nhiều loại màn hình (PC, laptop, tablet, điện thoại).

**JavaScript**: tạo tính năng tương tác, xử lý logic phía client, giúp người dùng có thể thao tác trực tiếp trên trình duyệt mà không cần cài đặt thêm phần mềm.

Thiết kế tổng thể của ứng dụng gồm các thành phần chính:

**Thanh tiêu đề (header)**: hiển thị tên ứng dụng và cung cấp cảm giác trực quan cho người dùng khi truy cập.

**Khu vực nhập dữ liệu**: cho phép người dùng lựa chọn giữa việc tải lên tệp tin hoặc nhập văn bản trực tiếp.

**Các tùy chọn mã hóa/giải mã**: bao gồm lựa chọn chế độ hoạt động (Encrypt/Decrypt), lựa chọn loại khóa (ngẫu nhiên từ số nguyên hoặc ASCII), cùng với checkbox tùy chọn nén Huffman.

**Khu vực nhập mật khẩu**: người dùng có thể nhập mật khẩu để tăng cường bảo mật. Mật khẩu này sẽ được mã hóa bằng AES và lưu giữ trong file log.

**Nút thao tác (action buttons)**: gồm “Mã hóa (Encrypt)” và “Giải mã (Decrypt)” để thực hiện tiến trình xử lý.

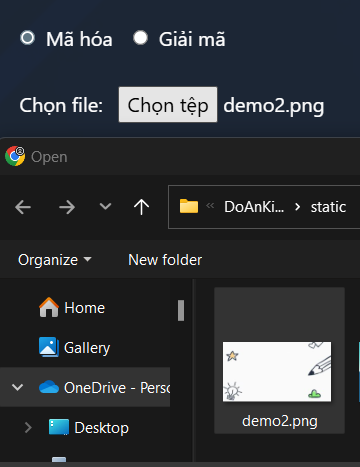
**Khu vực hiển thị kết quả**: hiển thị thông báo thành công/thất bại, và cung cấp liên kết tải file kết quả

📌 *(Hình 3.1: Giao diện tổng thể của ứng dụng)*

Nhờ thiết kế tối giản, gọn gàng, ứng dụng đáp ứng được yêu cầu của một công cụ mã hóa One-Time Pad trực tuyến, đồng thời vẫn tạo trải nghiệm thân thiện cho người dùng.

### 3.2 Chức năng của ứng dụng

**3.2.1. Chức năng mã hóa dữ liệu**

**Bước 1. Chọn tệp gốc cần mã hóa ( ở đây em sẽ tiến hành mã hóa ảnh)**  
Người dùng có thể tải lên một tệp văn bản (.txt) hoặc hình ảnh (.png, .jpg, .jpeg, .bmp, …) từ máy tính để đưa vào ứng dụng. Đây là dữ liệu gốc cần được bảo vệ bằng thuật toán OTP kết hợp AES.

📌 *(Hình 3.2: Giao diện lựa chọn tệp gốc để mã hóa)*

**Bước 2. Đặt tên cho tệp xuất**  
Trước khi thực hiện mã hóa, hệ thống yêu cầu người dùng nhập tên cho file đầu ra. Việc đặt tên giúp dễ dàng phân biệt giữa các file mã hóa, tránh trường hợp nhầm lẫn nếu xử lý nhiều dữ liệu cùng lúc.

Với văn bản: mặc định đặt đuôi file xuất.txt.

Với hình ảnh: xuất file .bin để phân biệt với ảnh gốc ( sẽ giải thích thêm ở 3.2.3)

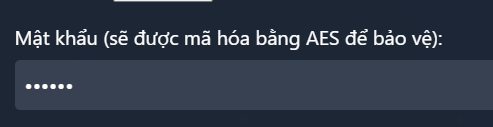
  
📌 *(Hình 3.3: Khung nhập tên tệp xuất)*

**Bước 3. Nhập mật khẩu bảo vệ (có ràng buộc định dạng)**  
Người dùng nhập mật khẩu bảo vệ cho quá trình mã hóa. Để tăng cường độ an toàn, hệ thống áp dụng ràng buộc:

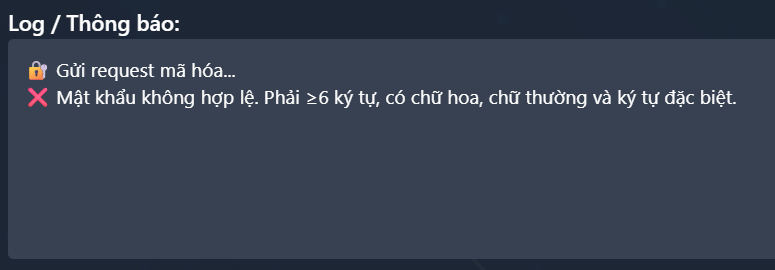
* Mật khẩu phải có **ít nhất 6 ký tự**.
* Mật khẩu phải chứa **ít nhất một chữ cái viết hoa (A–Z)**.
* Mật khẩu phải chứa **ít nhất một chữ cái viết thường (a–z)**.
* Mật khẩu phải chứa **ít nhất một ký tự đặc biệt** (ví dụ: @, #, $, %, !).

Nếu mật khẩu không thỏa mãn các điều kiện trên, hệ thống sẽ hiển thị thông báo lỗi và yêu cầu người dùng nhập lại.

Sau khi hợp lệ, mật khẩu này được **mã hóa bằng AES** và lưu trong file Log để phục vụ quá trình giải mã. Đây là lớp bảo mật bổ sung ngoài OTP, đảm bảo rằng ngay cả khi khóa OTP bị lộ thì dữ liệu vẫn không thể giải mã nếu thiếu mật khẩu AES chính xác.



📌 (Hình 3.4.2: Nhập mật khẩu)

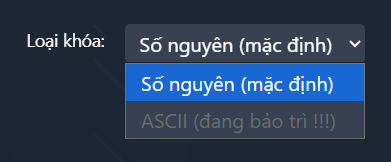


📌 (Hình 3.4.2: Nếu nhập mật khẩu sai định dạng sẽ có log gửi về và người dùng phải đặt lại mật khẩu theo định dạng )

**Bước 4. Chọn loại khóa sinh ngẫu nhiên**  
Người dùng có thể chọn một trong hai loại:

* Khóa sinh từ **số nguyên (Integer)**.
* Khóa sinh từ **bảng mã ASCII**.( Đang bảo trì)

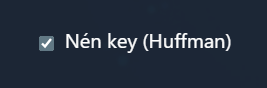
Khóa được sinh ra sẽ được biểu diễn dưới dạng **Hexadecimal (Hex)** để tiện cho lưu trữ, kiểm chứng và chia sẻ.



📌 (Hình 3.5: Chọn loại khóa, chọn khóa số nguyên)

**Bước 5. Tùy chọn nén Huffman**

Người dùng có thể chọn checkbox để nén khóa OTP bằng **thuật toán Huffman**. Việc nén này đặc biệt hữu ích với các file hình ảnh có dung lượng lớn, giúp giảm kích thước file khóa. Đây là tùy chọn, không bắt buộc.

  
📌 (Hình 3.6: Checkbox tùy chọn nén Huffman, có thể chọn hoặc không)

**Bước 6. Nhấn nút “Mã hóa”**  
Khi người dùng nhấn nút **Mã hóa**, hệ thống sẽ tự động thực hiện:

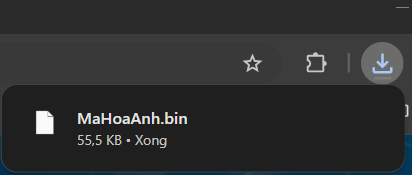
1. Sinh khóa OTP ngẫu nhiên.
2. Áp dụng phép toán XOR (Vernam cipher) giữa dữ liệu gốc và khóa OTP.
3. Mã hóa mật khẩu bằng AES và lưu giữ vào file Log.
4. Nén khóa OTP nếu có chọn Huffman.
5. Kết hợp tất cả các thành phần để tạo ra một file mã hóa duy nhất.



📌 (Hình 3.7: Màn hình hiển thị tiến trình mã hóa)

**Bước 7. Tải tệp mã hóa về máy**  
Khi mã hóa hoàn tất, ứng dụng xuất ra file dữ liệu đã được bảo vệ:

* Nếu là **văn bản**, mặc định xuất file .txt.
* Nếu là **hình ảnh**, xuất file .bin để tránh nhầm lẫn với ảnh gốc.



📌 (Hình 3.8: File mã hóa được tải xuống)

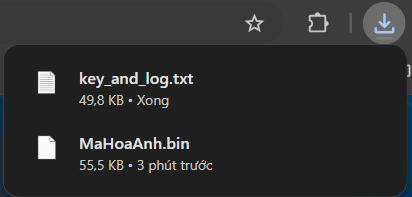
**Bước 8. Tải file Key (Hex) và Log (AES)**  
Ngay sau khi hoàn tất mã hóa, hệ thống sinh ra và yêu cầu người dùng tải về:

* **File Key (Hex):** chứa khóa OTP ngẫu nhiên.
* **File Log (AES):** chứa mật khẩu đã được mã hóa bằng AES.

File Key and Log này là thành phần **bắt buộc** để tiến hành giải mã dữ liệu sau này.



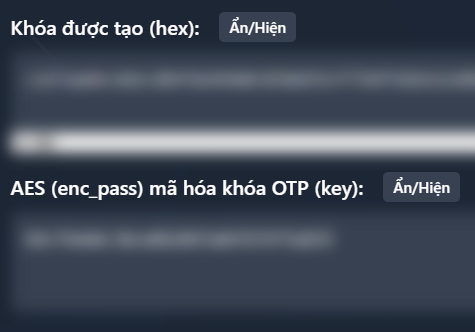
📌 (Hình 3.9.1: Nhấn Key + Log để tạo và tải file về)



📌 (Hình 3.9.2: File Key và Log được tạo sau khi mã hóa)

**Bước 9. Ẩn/Hiện Key Hex và AES trên giao diện**  
Để hỗ trợ việc kiểm chứng mà vẫn đảm bảo an toàn, giao diện cung cấp chức năng **ẩn/hiện (toggle)** khóa Hex và mật khẩu AES:

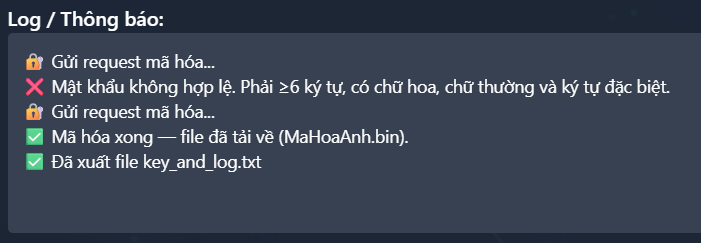
* Khi cần, người dùng có thể bật hiển thị để quan sát trực tiếp.
* Ngược lại, có thể ẩn đi để tránh rò rỉ thông tin trong quá trình thao tác.



📌 (Hình 3.10: Chức năng ẩn/hiện Key Hex và AES)

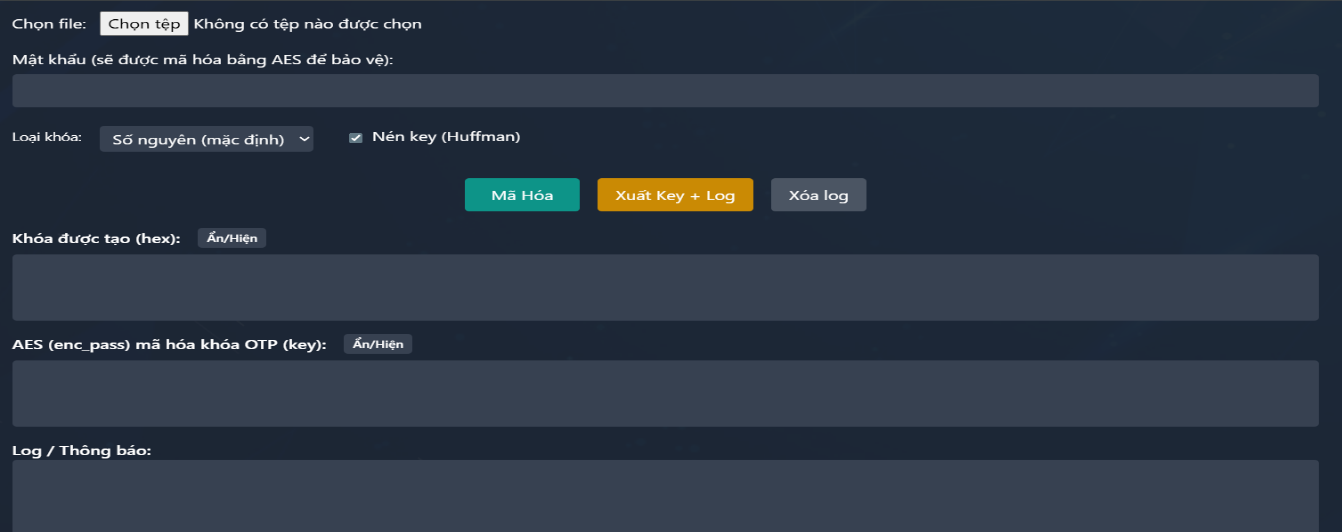
**Bước 10. Xem log/thông báo và Xóa log (Reset dữ liệu nhập)**

Xem log/thông báo để theo dõi tiến trình và sự kiện mà người dùng đã thao tác trước đó, nhập sai hay đúng, các file mã hóa và key log đã tải thành công hay chưa.

📌 (Hình 3.11.1: Xem log/thông báo dữ liệu đã nhập)

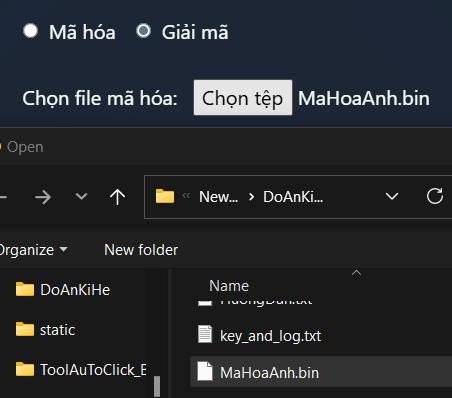
Trong trường hợp nhập sai file, sai mật khẩu hoặc nhầm tùy chọn, người dùng có thể nhấn nút **“Xóa log”** để làm mới toàn bộ dữ liệu đã nhập.

* Tất cả các trường (file gốc, mật khẩu, tùy chọn khóa, Huffman) sẽ được xóa sạch.
* Người dùng có thể bắt đầu lại quy trình mà **không cần tải lại trang**.

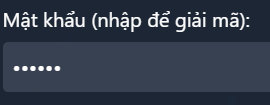


📌 (Hình 3.11.2: Nút “Xóa log” để làm mới dữ liệu đã nhập)

**3.2.2. Chức năng giải mã dữ liệu**

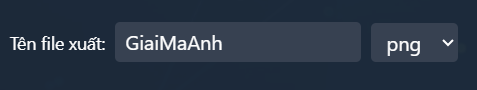
**Bước 1. Chọn tệp mã hóa cần giải mã**Người dùng tải lên tệp đã được mã hóa trước đó. Đây có thể là file .txt (văn bản) mặc định hoặc .bin (hình ảnh).

📌 *(Hình 3.12: Giao diện chọn tệp mã hóa)*

**Bước 2. Nhập mật khẩu bảo vệ**  
Người dùng nhập đúng mật khẩu đã sử dụng khi mã hóa. Nếu mật khẩu sai, hệ thống sẽ báo lỗi và không tiếp tục giải mã.

📌 *(Hình 3.13: Nhập mật khẩu để giải mã – mật khẩu mã hóa đã tạo trước đó)*

**Bước 3: Đặt tên file xuất sau khi giải mã**

**Lưu ý** phải đặt đuôi file có định dạng giống với file gốc, văn bản thì txt còn ảnh thì png hoặc jpg vì quá trình giải mã là biến đổi file mã hóa trở về thư mục gốc nên bắt buộc phải đặt đúng đuôi file

Hình 3.13.1: Nhập tên file và chọn đuôi file giống với file gốc để giải mã

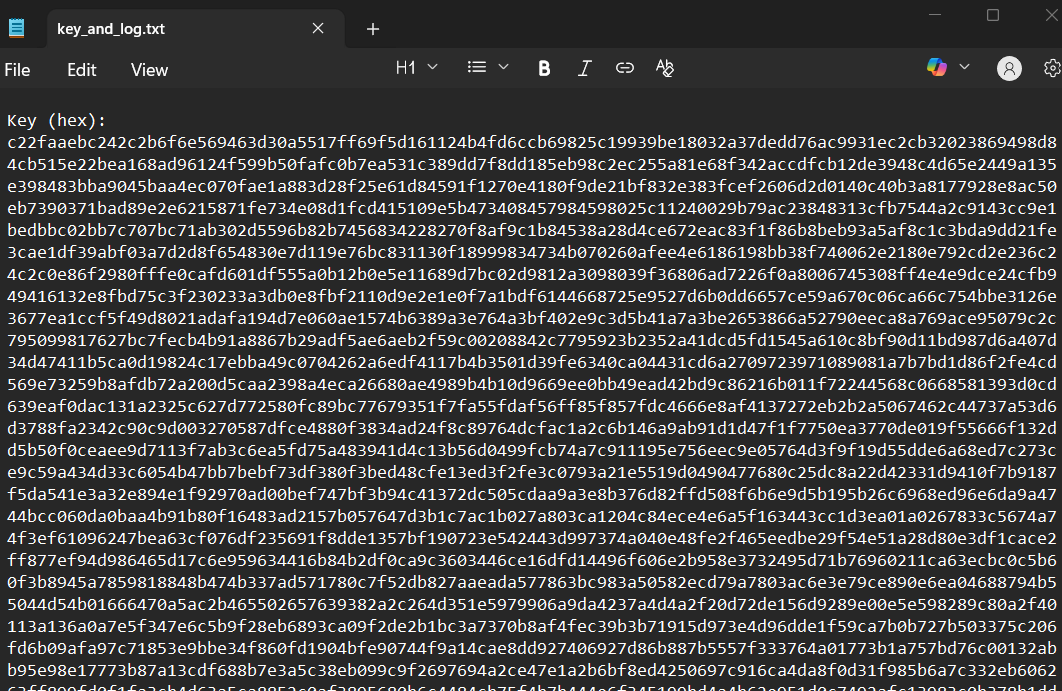
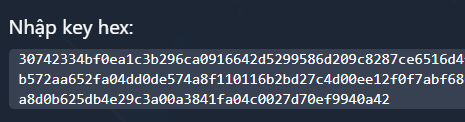
**Bước 4. Tiến hành mở file Key và Log**Người dùng mở file Key (Hex) và file Log (AES) đã lưu trong quá trình mã hóa. Đây là hai thành phần bắt buộc để khôi phục dữ liệu gốc.

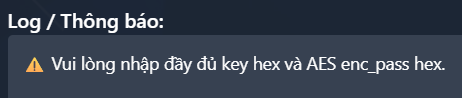


📌 *(Hình 3.14: Mở file Key & Log)*

**Bước 5. Khôi phục khóa OTP**

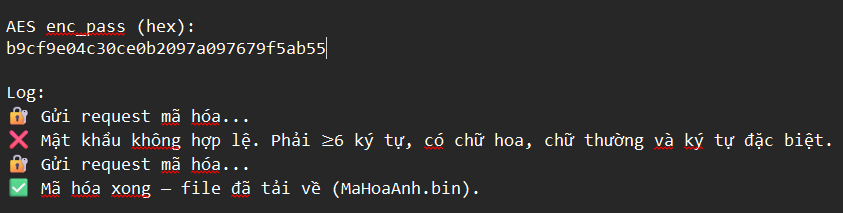
Ứng dụng sử dụng file Key tái tạo lại khóa OTP nếu khóa từng được nén bằng Huffman, tiếp theo đó ta lấy key hex từ flie key log nhập vào hệ thống sẽ tự động giải nén để lấy lại phiên bản ban đầu để tiến hành giải mã.

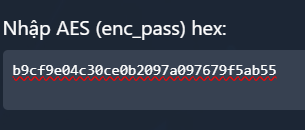
📌 *(Hình 3.15.1: Tiến hành copy key hex trong file key and long để giải mã)*

📌 *(Hình 3.15.2: Nhập key hex trên giao diện*

📌 *(Hình 3.15.3: Mật khẩu và key hex đúng trả về thông báo thiếu key AES*

**Bước 6. Giải mã mật khẩu bằng AES**

Hệ thống giải mã mật khẩu từ file Log và so sánh với mật khẩu người dùng nhập. Nếu trùng khớp, quá trình tiếp tục; nếu sai, thông báo lỗi xuất hiện.

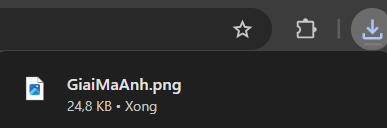
**📌 *(Hình 3.16: Quá trình giả mã mật khẩu AES)*

**Bước 7. Sau khi đã khôi phục khóa OTP từ file Key và xác thực thành công mật khẩu bằng AES, hệ thống sẽ tiến hành giải mã dữ liệu theo nguyên lý của thuật toán Vernam (XOR).** **Cụ thể:**

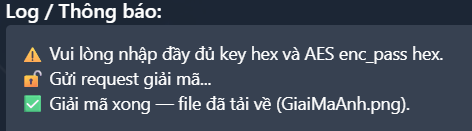
* Bản mã (ciphertext) được tạo ra ở bước mã hóa vốn dĩ là kết quả của phép toán XOR giữa dữ liệu gốc và khóa OTP.
* Do đó, để giải mã, ta chỉ cần thực hiện lại phép toán XOR giữa bản mã và cùng một khóa OTP thì sẽ thu được dữ liệu gốc ban đầu.

Khi giải mã xong, hệ thống sẽ:

1. **Tự động tải xuống file dữ liệu gốc** (có thể là văn bản .txt hoặc hình ảnh .jpg, .png tùy theo đầu vào ban đầu).
2. **Hiển thị log/thông báo trên giao diện** để người dùng biết tiến trình đã hoàn tất hay chưa. Nếu thành công, thông báo hiển thị kèm ký hiệu .

📌 *(Hình 3.17.1: Chọn giải mã để tiến hành giải mã file mã hóa)*

📌 *(Hình 3.17.2: Tải file giải mã ảnh)*



📌 *(Hình 3.17.3: Log thông báo trả về đã giải mã thành công)*

**Bước 8. Đối chiếu file giải mã với file ảnh gốc**

Sau khi tiến hành giải mã thành công bằng thuật toán Vernam (XOR), hệ thống xuất ra dữ liệu gốc dưới dạng file văn bản (.txt) hoặc hình ảnh (.png, .jpg, …) tùy thuộc vào định dạng ban đầu.

Để kiểm chứng tính chính xác và toàn vẹn của quá trình giải mã, người dùng cần thực hiện đối chiếu giữa:

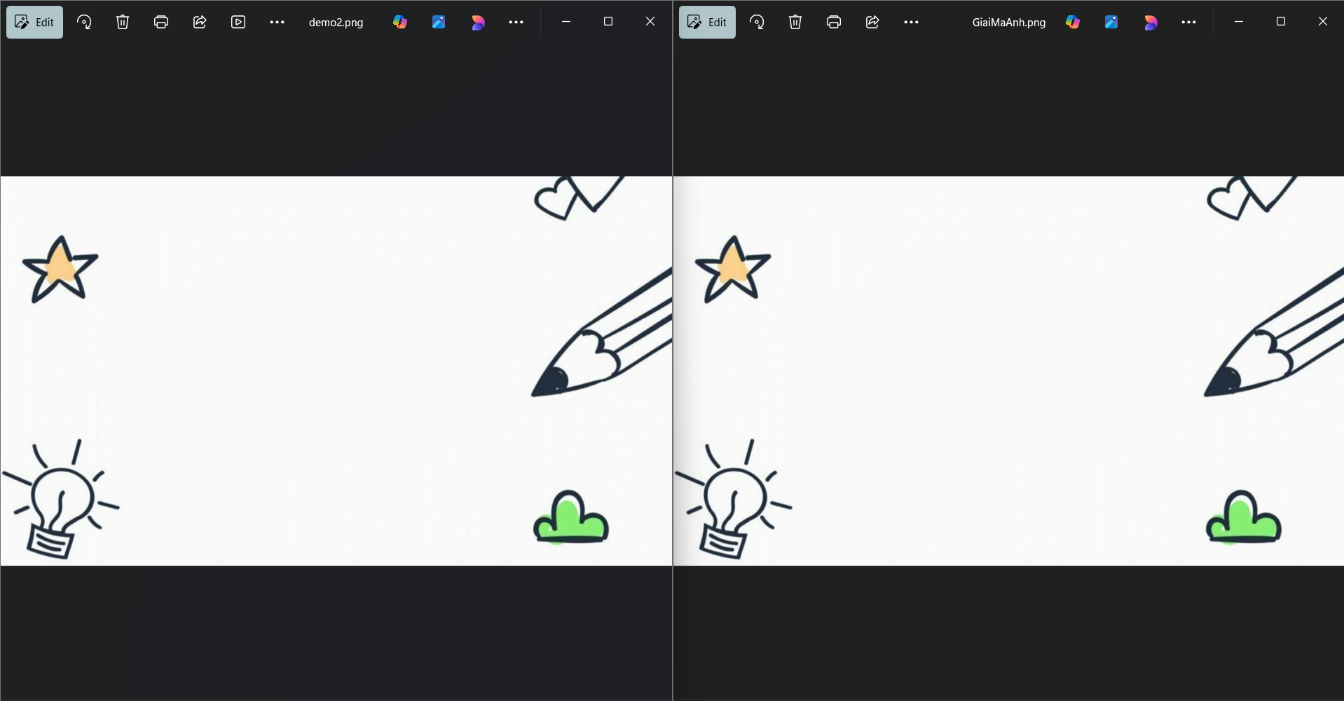
* File ảnh/văn bản gốc trước khi mã hóa.
* File đã được khôi phục sau quá trình giải mã.

Trong trường hợp dữ liệu là hình ảnh, ứng dụng còn hỗ trợ hiển thị preview trực tiếp trên giao diện web. Nhờ vậy, người dùng có thể quan sát ngay trên màn hình mà không cần mở bằng phần mềm ngoài. Việc so sánh trực quan này cho phép nhận biết ngay lập tức xem ảnh đã khôi phục có trùng khớp với ảnh gốc hay không.

Nếu dữ liệu giải mã trùng khớp hoàn toàn với dữ liệu gốc, điều này chứng minh rằng:

1. Quy trình mã hóa – giải mã bằng OTP kết hợp AES hoạt động chính xác.
2. Thuật toán Huffman (nếu có chọn nén) không làm mất mát dữ liệu.
3. Hệ thống có thể đảm bảo tính toàn vẹn, bảo mật và khả năng phục hồi dữ liệu.

Đây là bước cuối cùng nhằm xác nhận tính đúng đắn và độ tin cậy của toàn bộ ứng dụng. Việc đối chiếu không chỉ có ý nghĩa trong kiểm thử, mà còn giúp chứng minh khả năng ứng dụng thực tế của hệ thống vào các bài toán bảo mật dữ liệu.

**📌 *(Hình 3.17.3: Đối chiếu file giải mã và file ảnh gốc, nếu giống nhau tiến trình mã hóa và giải mã thành công)*

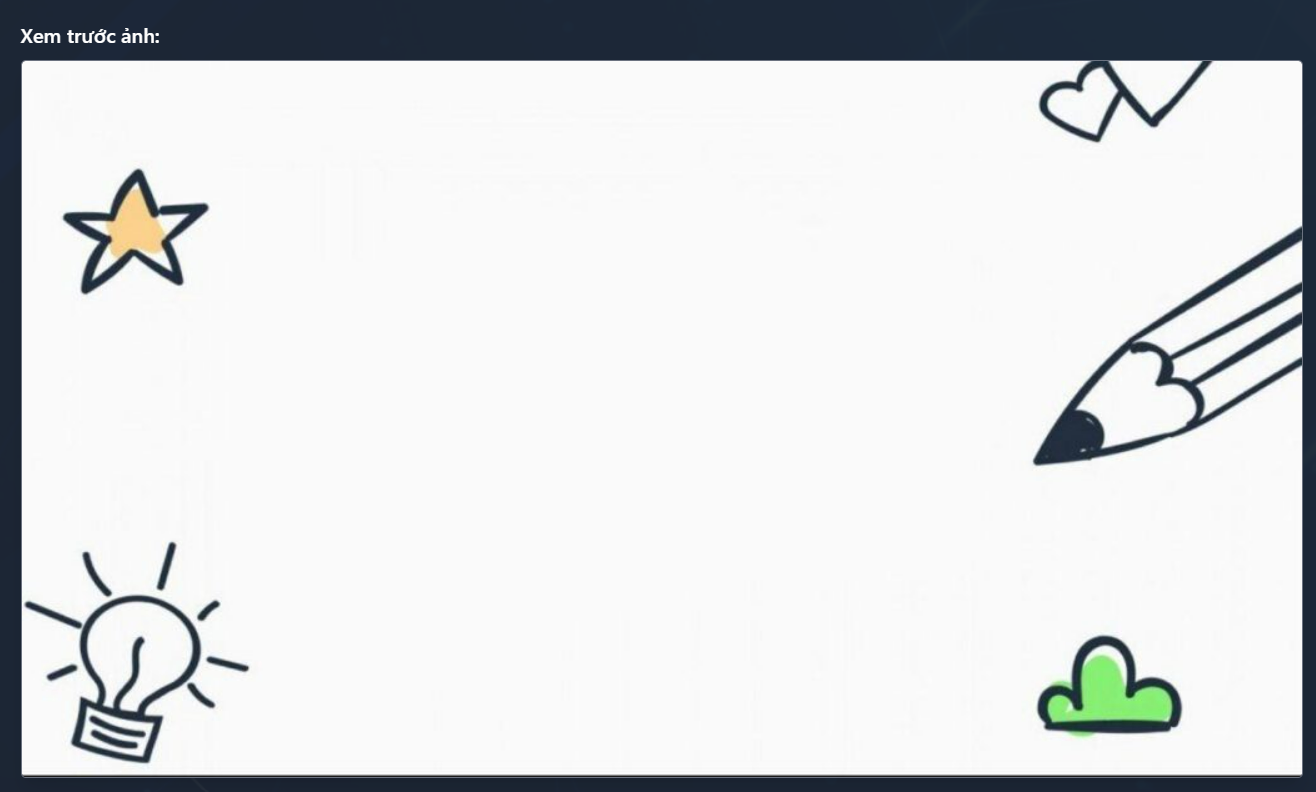
**Bước 9: Giới thiệu thêm về tính năng Preview ảnh sau giải mã (nếu dữ liệu là hình ảnh)**

Với dữ liệu dạng hình ảnh, ngoài việc xuất file ra máy, ứng dụng còn hỗ trợ **preview (xem trước)** ngay trên giao diện web. Điều này giúp người dùng có thể **so sánh trực tiếp ảnh đã giải mã với ảnh gốc** để kiểm chứng tính đúng đắn của quá trình.

Nhờ vậy, hệ thống đảm bảo rằng toàn bộ quy trình giải mã diễn ra **minh bạch, trực quan và dễ kiểm chứng**, đáp ứng tiêu chuẩn của một ứng dụng bảo mật trong môi trường thực nghiệm lẫn triển khai thực tế.

Nếu file cần giải mã là hình ảnh, ngoài việc xuất file gốc, ứng dụng còn hỗ trợ hiển thị preview ảnh ngay trên giao diện web. Điều này giúp người dùng kiểm chứng trực tiếp ảnh đã khôi phục mà không cần tải xuống, đồng thời thuận tiện khi thử nghiệm nhiều lần.

📌 *(Hình 3.18: Preview ảnh sau khi giải mã thành công)*

📌 *(Hình 3.18: Preview ảnh sau khi giải mã thành công)*

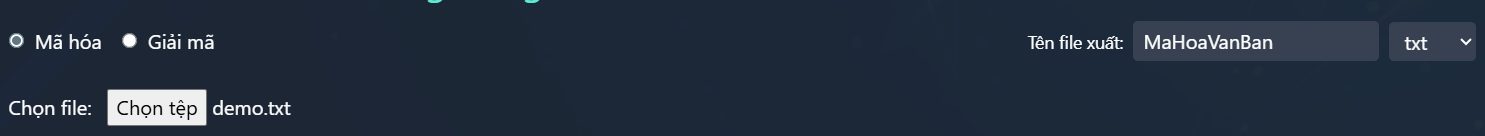
Bước 10. Xóa log (Reset dữ liệu nhập)  
Tương tự chức năng ở phần mã hóa, trong giao diện giải mã cũng có nút “Xóa log” để xóa toàn bộ nội dung đã nhập (file mã hóa, mật khẩu, Key, Log). Người dùng có thể dễ dàng nhập lại dữ liệu mới mà không cần tải lại trang.

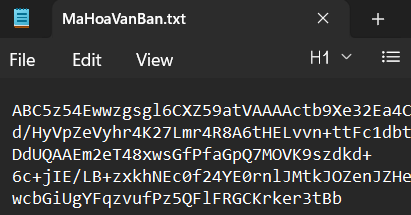
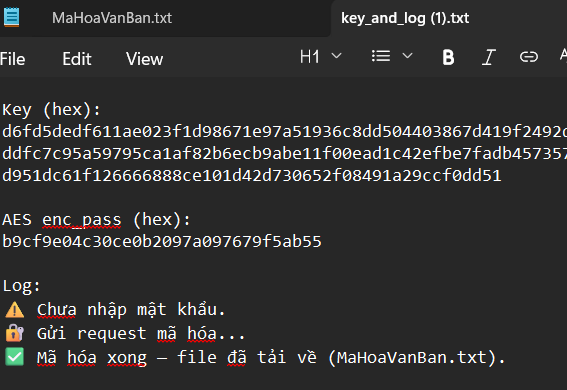
**

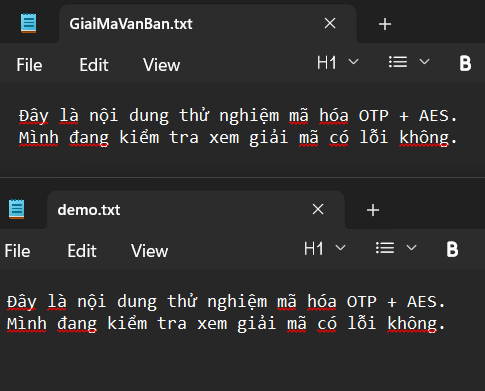
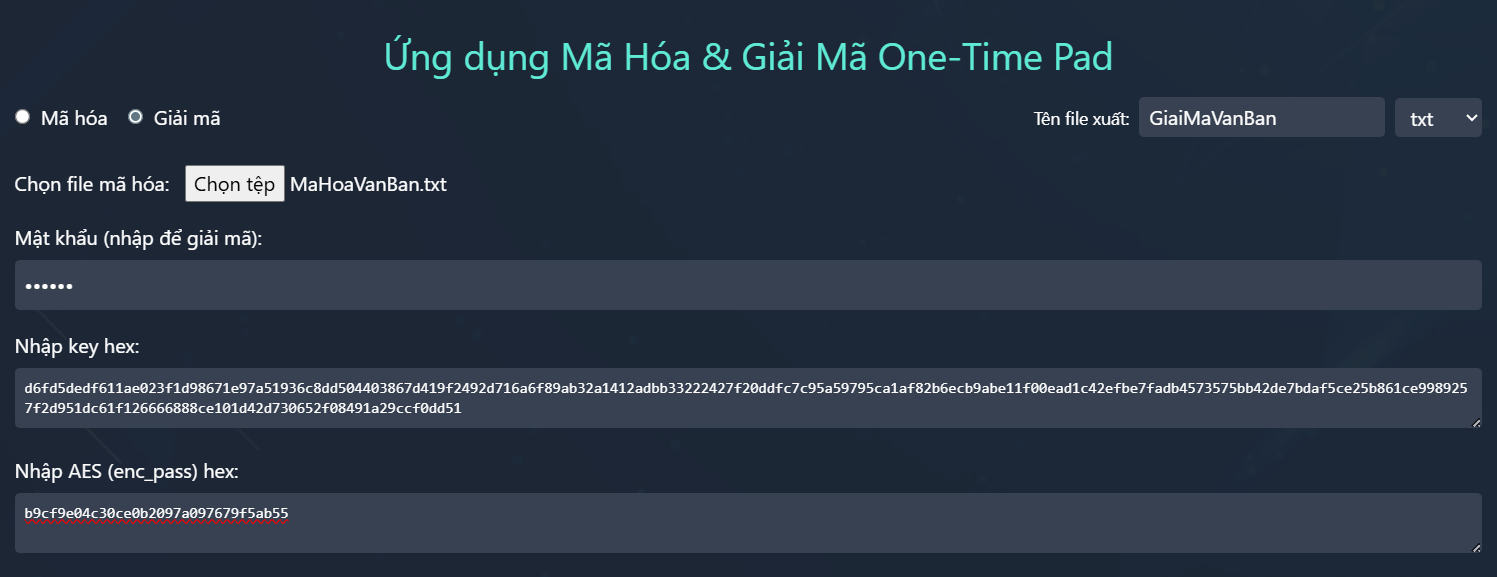
📌 *(Hình 3.19: Nút “Xóa log” trong chức năng giải mã)*

**3.2.3. Mã hóa & giải mã văn bản và hình ảnh có gì khác và sự tối ưu của chúng**

**1. Đối với dữ liệu dạng văn bản (txt), ứng dụng OTP xử lý tương đối đơn giản:**

* **Khi mã hóa:**
  + Người dùng nhập hoặc tải lên một file .txt chứa dữ liệu gốc.
  + Ứng dụng sinh khóa OTP, áp dụng phép toán XOR và xuất ra file mã hóa.
  + Định dạng file mã hóa và file giải mã mặc định đuôi khi tải xuống là **.txt**, để người dùng có thể dễ dàng mở bằng các công cụ đọc văn bản thông thường.
* **Khi giải mã:**
  + Người dùng cung cấp file .txt đã được mã hóa cùng với Key & Log.
  + Ứng dụng tiến hành khôi phục và xuất lại file văn bản gốc dưới dạng .txt.

📌 *(Hình 3.20: Ví dụ mã hóa một đoạn văn bản)*

  
📌 *(Hình 3.21: Kết quả giải mã văn bản trở lại dạng gốc)*

Việc giữ nguyên định dạng .txt giúp cho quá trình thử nghiệm và kiểm tra tính đúng đắn của thuật toán OTP trở nên dễ dàng, đồng thời phù hợp với đặc thù dữ liệu văn bản có cấu trúc đơn giản.

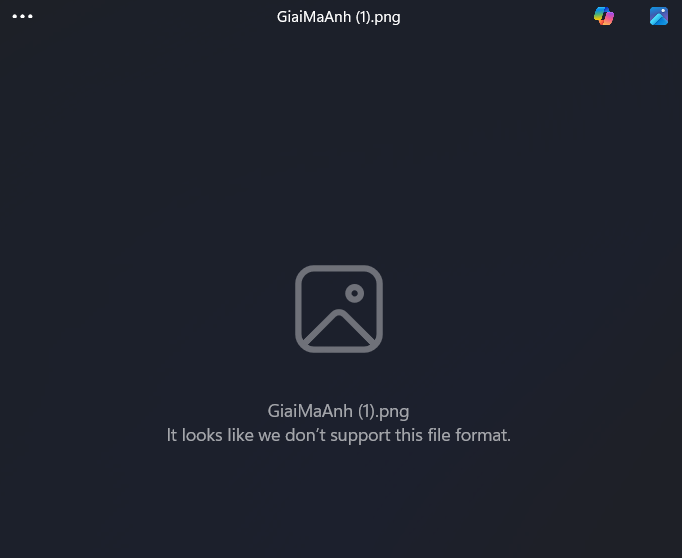
**2. Mã hóa và giải mã đối với dạng hình ảnh**

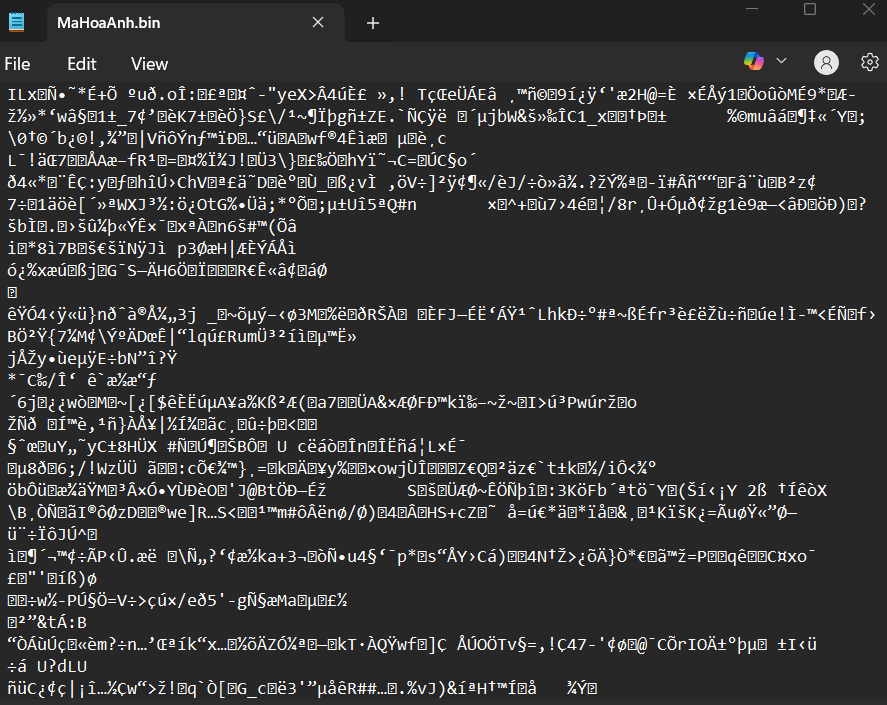
Khác với văn bản, hình ảnh là dữ liệu nhị phân (binary data), có cấu trúc phức tạp và nhạy cảm với thay đổi định dạng. Nếu mã hóa xong mà vẫn giữ đuôi gốc .png hoặc .jpg, người dùng có thể gặp vấn đề:

* File mã hóa không thể mở bằng trình xem ảnh thông thường vì đã thay đổi dữ liệu nhị phân bên trong.
* Khó phân biệt đâu là ảnh gốc và đâu là file đã mã hóa.

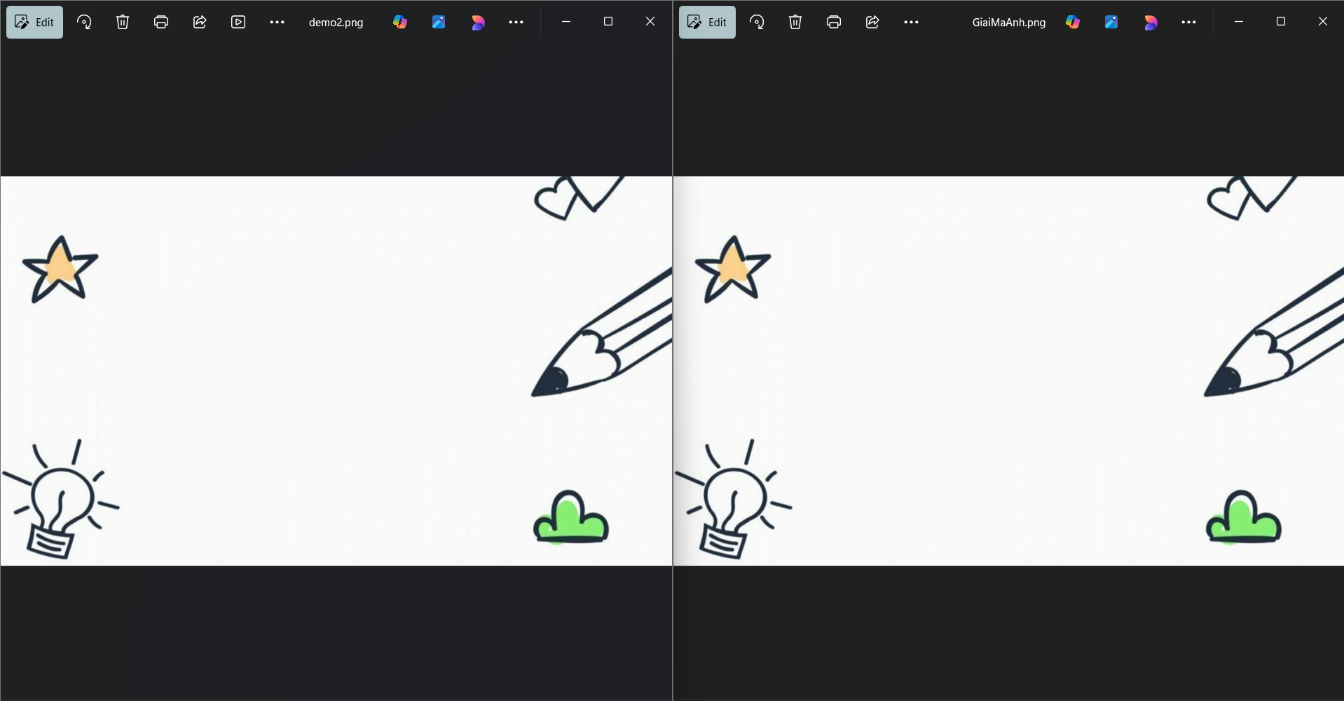
👉 Để khắc phục, ứng dụng xuất file mã hóa hình ảnh dưới dạng **.bin**. Đây là định dạng dữ liệu nhị phân thô, đảm bảo:

* Không gây nhầm lẫn với file ảnh gốc.
* Người dùng dễ dàng nhận biết file nào là **dữ liệu đã mã hóa**.
* Tránh lỗi định dạng khi mở file bằng phần mềm không phù hợp.
* **Khi mã hóa:**
  + Người dùng tải lên file ảnh (PNG, JPG, JPEG, BMP, v.v…).
  + Ứng dụng áp dụng thuật toán OTP → xuất file mã hóa dưới dạng .bin.
* **Khi giải mã:**
  + Người dùng cung cấp file .bin cùng với Key & Log.
  + Ứng dụng khôi phục dữ liệu nhị phân ban đầu và xuất ra đúng định dạng ảnh gốc (ví dụ: .png hoặc .jpg).



📌 *(Hình 3.22.2: Mã hóa một ảnh PNG/JPG → file mã hóa .png/.jp*

📌 *(Hình 3.22.2: Mã hóa một ảnh PNG/JPG → file mã hóa .bin)*

**  
📌 *(Hình 3.23: Giải mã file .bin trở lại ảnh JPG ban đầu)*

**3.2.4. Sự tối ưu khi xử lý văn bản và hình ảnh**

* **Văn bản:** giữ nguyên định dạng .txt giúp thuận tiện kiểm tra, dễ dàng mở và so sánh kết quả.
* **Hình ảnh:** việc chuyển file mã hóa sang .bin thay vì png/jpg là một giải pháp tối ưu:
  + Tránh xung đột định dạng với ảnh gốc.
  + Tăng tính trực quan khi phân biệt file gốc và file đã mã hóa.
  + Đảm bảo file mã hóa không bị lỗi khi người dùng mở nhầm bằng trình xem ảnh.

Nhờ cơ chế này, ứng dụng đáp ứng được nhu cầu mã hóa dữ liệu **đa dạng (text + binary)** mà vẫn đảm bảo **an toàn, tiện dụng và dễ phân biệt**.

### 3.3. Triển khai ứng dụng lên môi trường web

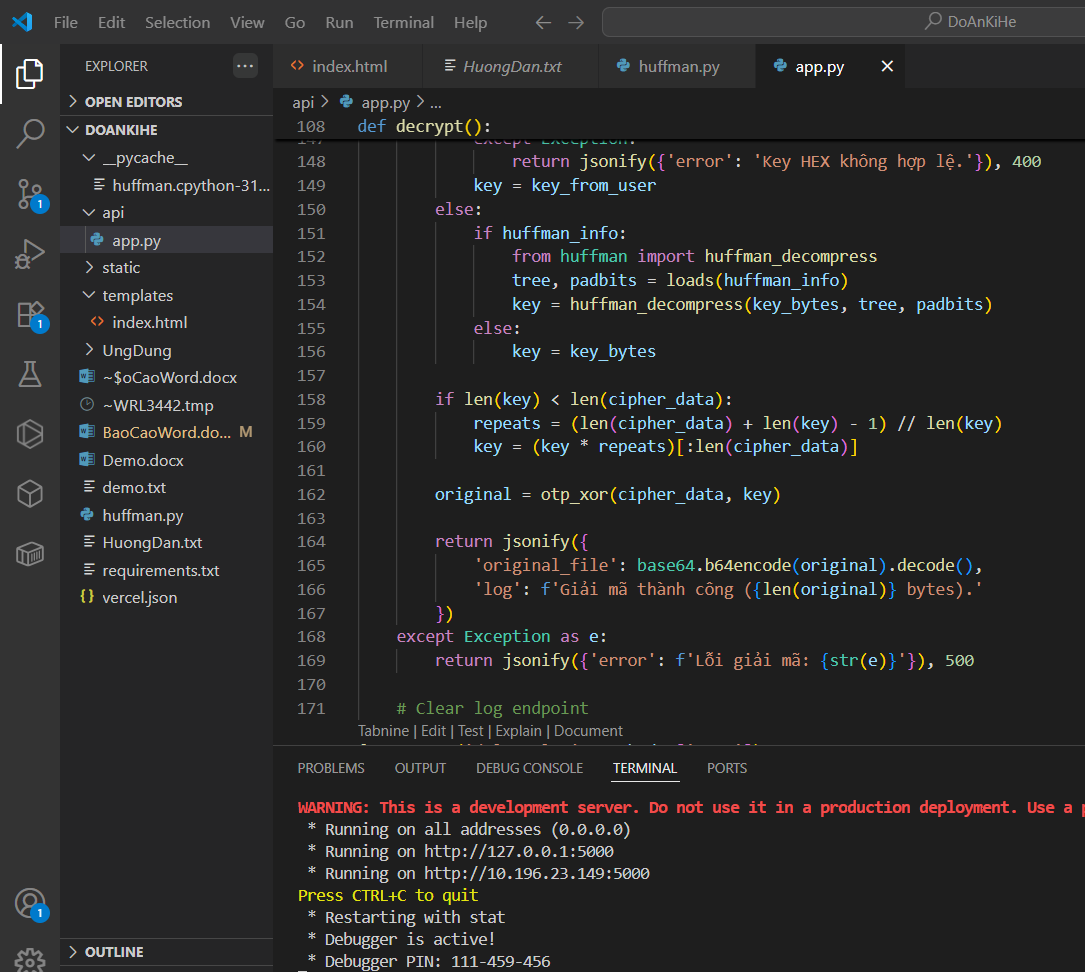
Ứng dụng được triển khai trên nền tảng **Vercel** theo mô hình **Jamstack**, giúp:

* Triển khai nhanh chóng, không cần server backend phức tạp., toàn bộ chạy phía client.
* **Tích hợp CI/CD (Continuous Integration/Continuous Deployment)**: mỗi khi có thay đổi hoặc bổ sung code trên GitHub, hệ thống tự động build và deploy phiên bản mới.
* **Đảm bảo tốc độ tải nhanh, bảo mật cao**, nhờ hạ tầng CDN phân tán toàn cầu và hỗ trợ HTTPS mặc định.

**Quy trình triển khai chi tiết:**

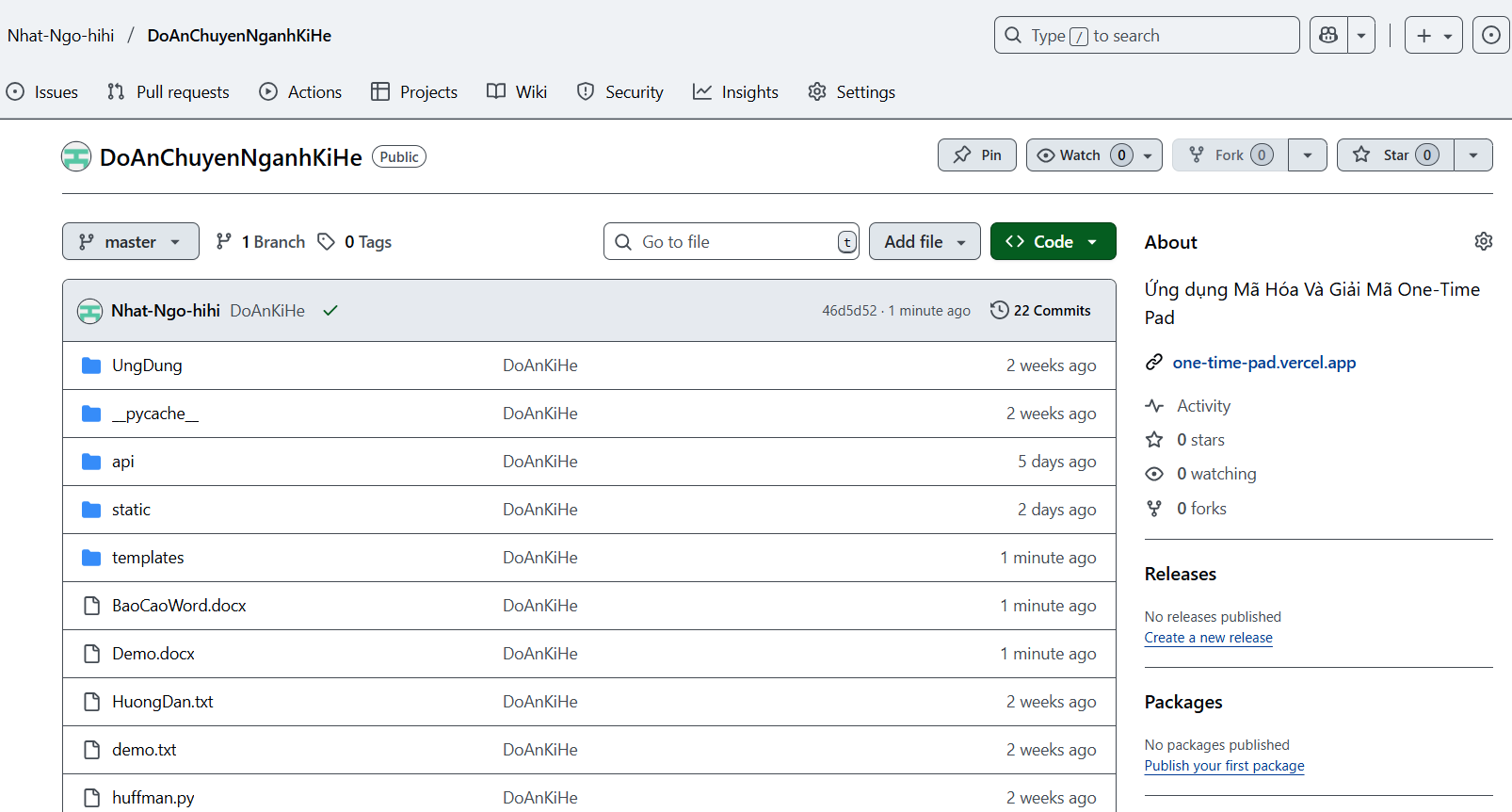
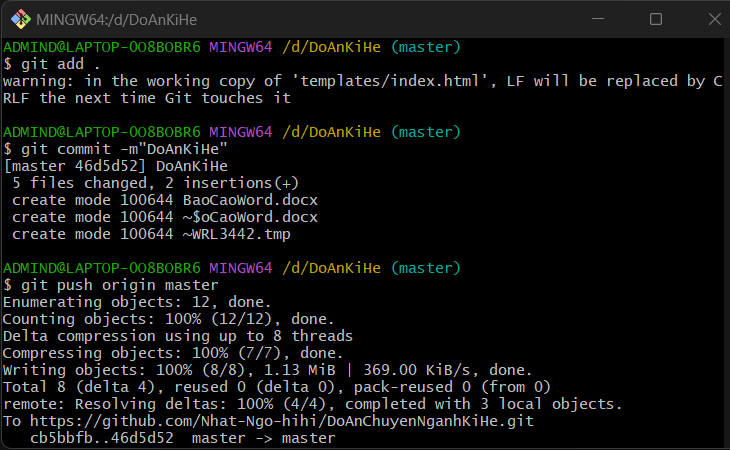
1. Phát triển trên Visual Studio Code (VS Code):

* Toàn bộ mã nguồn được viết và kiểm thử trên VS Code.
* Các file quan trọng: app.py (xử lý thuật toán OTP + AES), huffman.py (xử lý nén/giải nén khóa), index.html (giao diện web).
* Trong quá trình lập trình, ứng dụng được kiểm thử cục bộ bằng cách chạy lệnh python app.py, sau đó truy cập localhost trên trình duyệt để quan sát kết quả.

📌 (Hình 3.24: Môi trường lập trình trên VS Code)

2. Đưa mã nguồn lên GitHub bằng Git Bash:

* Khởi tạo repository local trên máy tính cá nhân.
* Thực hiện commit từng phần thay đổi để quản lý phiên bản dễ dàng.
* Push toàn bộ mã nguồn lên GitHub thông qua Git Bash.

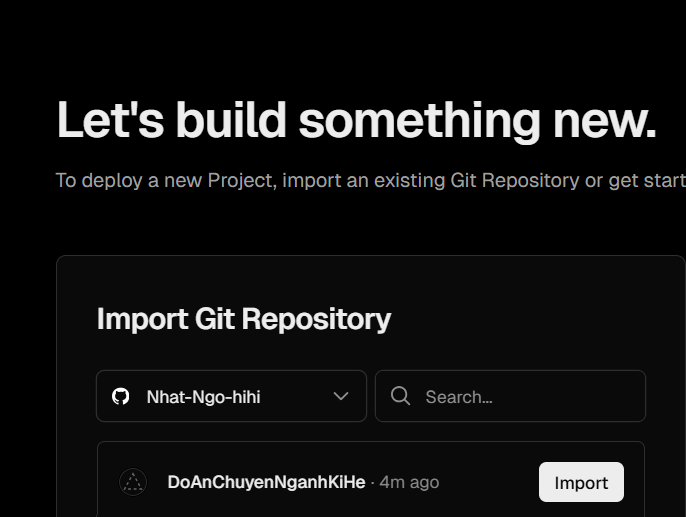
📌 (Hình 3.25: Sử dụng Git Bash để push code lên GitHub)

📌 (Hình 3.26: Sau khi đẩy toàn bộ sourd code thành công lên github)

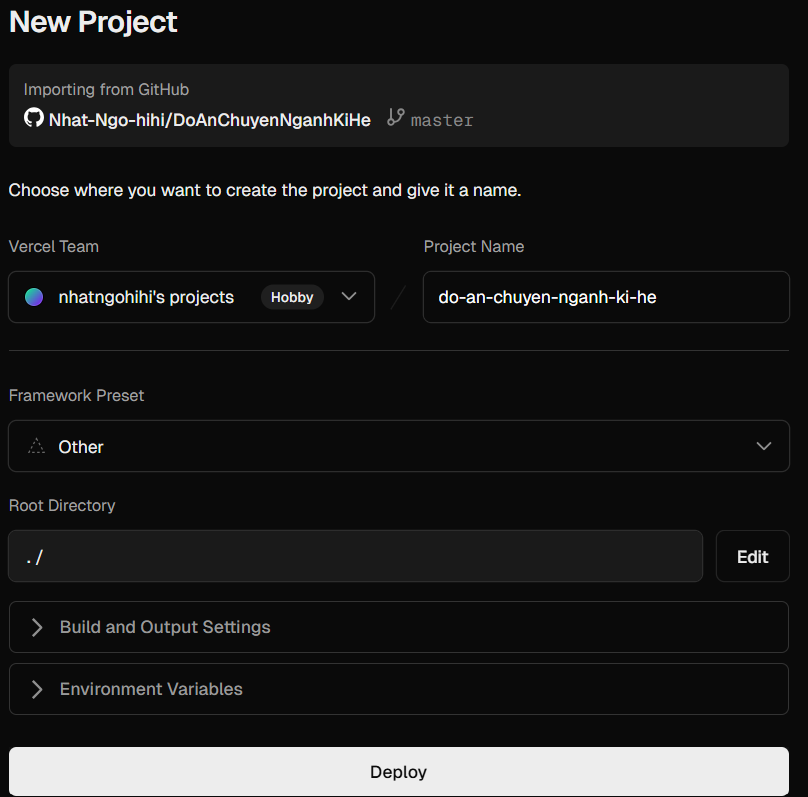
* Mã nguồn được công khai tại địa chỉ:  
  👉 [https://github.com/Nhat-Ngo-hihi/DoAnChuyenNganhKiHe](https://github.com/Nhat-Ngo-hihi/DoAnChuyenNganhKiHe?utm_source=chatgpt.com)

**3. Kết nối GitHub với Vercel:**

Sau khi mã nguồn đã được quản lý và đẩy lên GitHub, bước tiếp theo là triển khai ứng dụng lên môi trường **Vercel** để chạy trực tuyến. Quy trình cụ thể như sau:

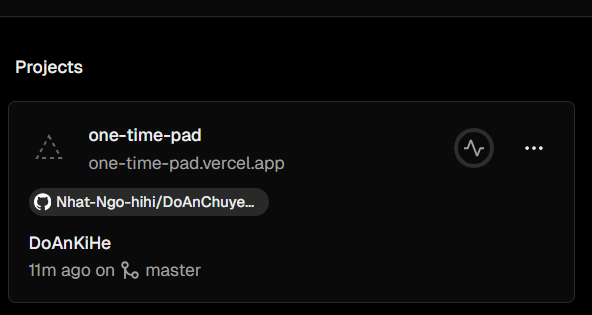
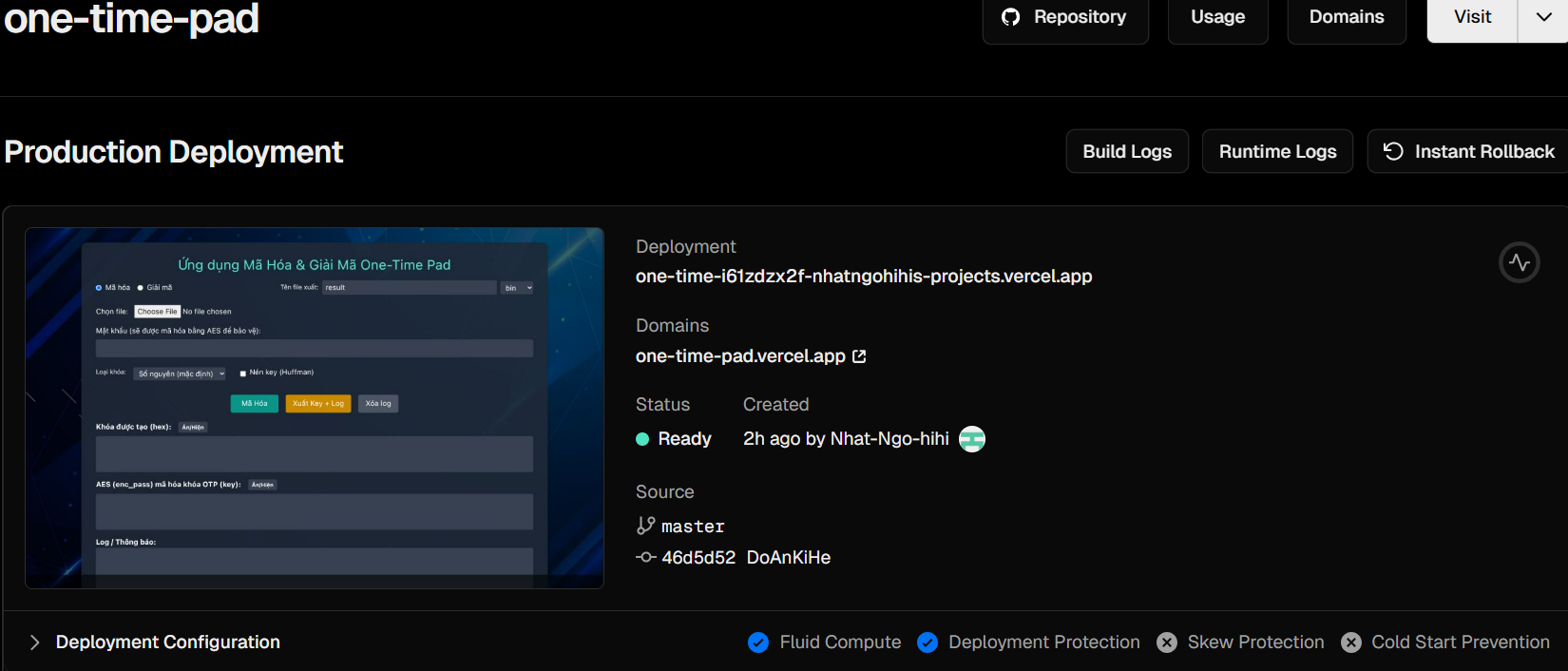
* **Đăng nhập Vercel**: Người phát triển đăng nhập vào tài khoản Vercel thông qua trình duyệt web. Đây là nền tảng hỗ trợ triển khai ứng dụng theo kiến trúc Jamstack, tối ưu cho các project HTML/CSS/JS.
* **Import project từ GitHub**: Trong giao diện Vercel, người dùng lựa chọn kết nối trực tiếp đến repository GitHub chứa mã nguồn của đồ án (ví dụ: https://github.com/Nhat-Ngo-hihi/DoAnChuyenNganhKiHe). Nhờ đó, quá trình triển khai luôn gắn liền với phiên bản code mới nhất trên GitHub.

📌 (Hình 3.27.1: Tiến hành import dự án lên Vercel)



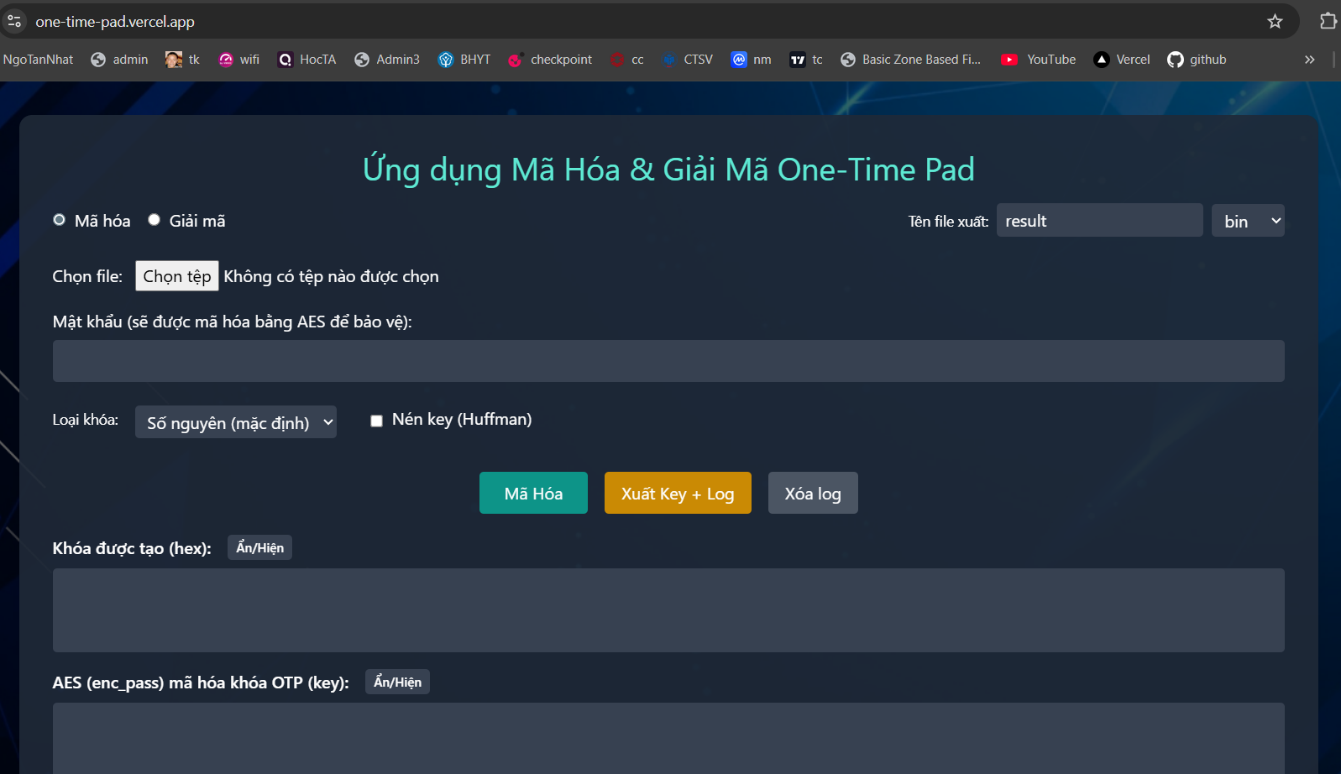
📌 (Hình 3.27.2.: Quá trình build & deploy trên Vercel

**Theo dõi log và trạng thái**

* Mỗi lần build và deploy, Vercel đều cung cấp **log chi tiết** (thành công/thất bại).
* Nếu có lỗi trong quá trình build (ví dụ: thiếu file, lỗi cấu hình), log sẽ hiển thị vị trí lỗi để người phát triển dễ dàng sửa chữa.
* Trạng thái triển khai hiển thị **“Deployment completed / Success”** kèm dấu tick xanh báo hiệu đã triển khai thành công.
* Vercel cung cấp ngay liên kết **Production URL** (ví dụ: https://one-time-pad.vercel.app/) để người dùng có thể truy cập trực tiếp vào ứng dụng.

📌 (Hình 3.27.3: Sau khi build & deploy trên Vercel thành công)

**4. Ứng dụng online:**

* Sau khi triển khai thành công, ứng dụng được chạy trực tuyến và có thể truy cập trên mọi thiết bị qua địa chỉ: 👉 [https://one-time-pad.vercel.app/](https://one-time-pad.vercel.app/?utm_source=chatgpt.com)
* Đây là phiên bản chính thức của ứng dụng, cho phép người dùng ở bất kỳ đâu có thể thực hiện mã hóa/giải mã mà không cần cài đặt phần mềm.

📌 (Hình 3.28: Giao diện ứng dụng khi chạy trên web)

* Ngoài ra, hệ thống cũng tạo thêm **Preview URL** cho từng lần commit, giúp nhóm phát triển dễ dàng kiểm thử trước khi đưa bản chính thức lên production.
* Nhờ cơ chế **CI/CD tích hợp sẵn**, mỗi lần commit/push mới trên GitHub đều được Vercel tự động kích hoạt lại quy trình build & deploy, đảm bảo ứng dụng luôn ở trạng thái mới nhất mà không cần thao tác thủ công.

Điều này minh chứng cho ưu điểm lớn của việc triển khai trên Vercel: **nhanh chóng – an toàn – tự động hóa**.

# KẾT LUẬN

**1. Hướng phát triển trong tương lai**

Trong phạm vi đồ án, ứng dụng mới chỉ dừng lại ở mức minh hoạ nguyên lý của thuật toán One-Time Pad và kết hợp với một số cơ chế bổ sung như nén khoá hay bảo vệ bằng AES. Tuy nhiên, để hoàn thiện và tiến tới ứng dụng thực tiễn, đề tài có thể phát triển theo một số hướng sau:

* **Tối ưu hoá quản lý và phân phối khoá**

Hiện nay, khoá OTP có kích thước rất lớn, bằng đúng dữ liệu cần mã hoá. Điều này gây khó khăn trong việc lưu trữ và chia sẻ khoá. Hướng phát triển trong tương lai là nghiên cứu các cơ chế quản lý khoá an toàn hơn, chẳng hạn như sử dụng các kỹ thuật phân tán (chia khoá thành nhiều phần và chỉ khôi phục khi có đủ thành phần) hoặc triển khai lưu trữ khoá trên thiết bị chuyên dụng (HSM – Hardware Security Module). Ngoài ra, cần cải tiến bộ sinh số ngẫu nhiên bằng cách sử dụng CSPRNG hoặc nguồn entropy phần cứng để đảm bảo độ ngẫu nhiên tuyệt đối của khoá, tránh tình trạng sinh khoá giả ngẫu nhiên dễ bị dự đoán.

* **Kết hợp với các phương pháp mã hoá hiện đại**

Mặc dù OTP mang tính an toàn lý thuyết, song nhược điểm lớn là tính khả thi thấp trong triển khai. Do đó, một hướng mở rộng là xây dựng mô hình lai (hybrid model), kết hợp giữa OTP và các thuật toán mã hoá hiện đại. Ví dụ: sử dụng AES hoặc RSA để mã hoá và phân phối khoá OTP, sau đó OTP mới được dùng để mã hoá dữ liệu. Cách tiếp cận này vừa giữ được tính bảo mật của OTP, vừa giải quyết phần nào bài toán truyền khoá trong thực tế.

* **Mở rộng hệ thống ứng dụng và giao diện**

Ở phiên bản hiện tại, ứng dụng chủ yếu chạy ở môi trường cục bộ với giao diện cơ bản. Trong tương lai, có thể phát triển thành hệ thống web hoặc ứng dụng đám mây, cho phép mã hoá/giải mã trực tuyến, đồng thời tích hợp cơ chế xác thực người dùng, phân quyền và ghi log bảo mật. Việc bổ sung các chức năng như cảnh báo độ mạnh mật khẩu, tự động xoá khoá sau khi dùng, hoặc kiểm soát hoạt động người dùng cũng giúp nâng cao mức độ an toàn tổng thể của hệ thống.

* **Ứng dụng trong thực tiễn**

Cuối cùng, một hướng quan trọng là kiểm thử ứng dụng OTP trên các loại dữ liệu nhạy cảm trong thực tế, chẳng hạn như hình ảnh y tế, hồ sơ pháp lý hay tài liệu nghiên cứu khoa học mật. Xa hơn, OTP có thể được kết hợp với các giao thức truyền thông để xây dựng hệ thống nhắn tin an toàn tuyệt đối, hoặc truyền file bí mật trong môi trường quân sự, quốc phòng.

**2. Những điều chưa làm được**

Mặc dù ứng dụng đã minh hoạ được cơ chế của thuật toán One-Time Pad và tích hợp thêm các tính năng hỗ trợ như bảo mật khoá bằng AES hay nén khoá bằng Huffman, đề tài vẫn còn một số hạn chế như sau:

* **Chưa giải quyết triệt để vấn đề phân phối khoá OTP**

Trong mô hình hiện tại, khoá OTP vẫn được tạo và lưu trữ trên cùng hệ thống với dữ liệu. Điều này chưa phản ánh đúng thách thức lớn nhất của OTP trong thực tế: làm thế nào để phân phối và chia sẻ khoá cho hai bên một cách an toàn tuyệt đối.

* **Bộ sinh số ngẫu nhiên còn đơn giản**

Ứng dụng hiện mới sử dụng bộ sinh số ngẫu nhiên mặc định của môi trường lập trình, vốn không đảm bảo độ ngẫu nhiên tuyệt đối (true randomness). Vì vậy, độ an toàn lý thuyết của OTP chưa được tái hiện hoàn toàn trong phiên bản này.

* **Hiệu quả nén khoá còn hạn chế**

Thuật toán Huffman chỉ có hiệu quả khi dữ liệu khoá có sự phân bố không đều. Với khoá thực sự ngẫu nhiên, hiệu quả nén gần như bằng 0. Do đó, tính năng nén khoá trong ứng dụng hiện nay chủ yếu mang tính minh hoạ, chưa có giá trị thực tiễn cao.

* **Giao diện và trải nghiệm người dùng còn cơ bản**

Ứng dụng mới chỉ hỗ trợ mã hoá/giải mã file cục bộ với giao diện đơn giản. Chưa có các tính năng nâng cao như xác thực nhiều lớp, cảnh báo độ mạnh mật khẩu theo thời gian thực, hay triển khai trên nền tảng web/đám mây để phục vụ nhiều người dùng.

* **Chưa kiểm thử trên dữ liệu và môi trường đa dạng**

Các thử nghiệm chủ yếu tập trung vào một số định dạng file (txt, pdf, jpg, png). Chưa có kiểm chứng với các hệ thống truyền thông thực tế (chat, truyền file trực tuyến), cũng như chưa tiến hành đo lường hiệu năng khi áp dụng cho dữ liệu dung lượng lớn.

## TÀI LIỆU THAM KHẢO

1. Mã Một Lần Và Bảo Mật Tuyệt Đối | CodeLearn. [Online]. Truy cập ngày 31/08/2025.  
   [https://codelearn.io/sharing/ma-mot-lan-va-bao-mat-tuyet-doi](https://codelearn.io/sharing/ma-mot-lan-va-bao-mat-tuyet-doi?utm_source=chatgpt.com)
2. Crypto Museum, *One-Time Pad*. [Online]. Truy cập ngày 31/08/2025.  
   [https://www.cryptomuseum.alibaba.sk/crypto/otp.htm](https://www.cryptomuseum.alibaba.sk/crypto/otp.htm?utm_source=chatgpt.com)
3. Crypto Museum, *Proof of the Perfect Secrecy of OTP*. [PDF]. Truy cập ngày 31/08/2025.  
   [https://www.cryptomuseum.com/manuf/mils/files/mils\_otp\_proof.pdf](https://www.cryptomuseum.com/manuf/mils/files/mils_otp_proof.pdf?utm_source=chatgpt.com)
4. Wikipedia, *One-Time Pad*. [Online]. Truy cập ngày 31/08/2025.  
   [https://en.wikipedia.org/wiki/One-time\_pad](https://en.wikipedia.org/wiki/One-time_pad?utm_source=chatgpt.com)
5. TechTarget, *One-Time Pad Definition*. [Online]. Truy cập ngày 31/08/2025.  
   [https://www.techtarget.com/searchsecurity/definition/one-time-pad](https://www.techtarget.com/searchsecurity/definition/one-time-pad?utm_source=chatgpt.com)
6. Tutorialspoint, *Cryptography – One Time Pad Cipher*. [Online]. Truy cập ngày 31/08/2025.  
   [https://www.tutorialspoint.com/cryptography/cryptography\_one\_time\_pad\_cipher](https://www.tutorialspoint.com/cryptography/cryptography_one_time_pad_cipher?utm_source=chatgpt.com)
7. Claude E. Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal, 1949.
8. Vernam, G. S., *Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications*, Journal of the AIEE, 1919.
9. William Stallings, *Cryptography and Network Security: Principles and Practice*, 7th Edition, Pearson, 2016.
10. Daemen, J., & Rijmen, V., *The Design of Rijndael: AES - The Advanced Encryption Standard*, Springer, 2002.
11. Huffman, D. A., *A Method for the Construction of Minimum-Redundancy Codes*, Proceedings of the IRE, 1952.
12. Trần Văn Hòa, *Giáo trình An toàn và Bảo mật Thông tin*, Nhà xuất bản Khoa học và Kỹ thuật, Hà Nội, 2018.
13. 200lab.io, *Các chuẩn mã hóa và khuyến nghị sử dụng trong bảo mật hiện đại*. [Online]. Truy cập ngày 30/08/2025.  
    [https://200lab.io/](https://200lab.io/?utm_source=chatgpt.com)
14. JAMstack.org, *What is JAMstack?*. [Online]. Truy cập ngày 30/08/2025.  
    [https://jamstack.org/](https://jamstack.org/?utm_source=chatgpt.com)
15. Vercel, *Deploy web applications with Vercel*. [Online]. Truy cập ngày 30/08/2025.  
    [https://vercel.com/](https://vercel.com/?utm_source=chatgpt.com)
16. Dự án demo của nhóm chúng em: One-Time Pad, *Ứng dụng mã hóa dựa trên OTP kết hợp AES và Huffman*. [Online]. Truy cập ngày 30/08/2025.  
    [https://one-time-pad.vercel.app/](https://one-time-pad.vercel.app/?utm_source=chatgpt.com)