**HỌC VIỆN KỸ THUẬT MẬT MÃ**

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO MẬT MÃ ỨNG DỤNG

TRONG AN TOÀN THÔNG TIN

ĐỀ TÀI:

**XÂY DỰNG ỨNG DỤNG SỬ DỤNG THƯ VIỆN OPENSSL**

***Cán bộ hướng dẫn: Ths Bùi Hữu Đông***

***Sinh viên thực hiện:***

* ***Trần Thị Ngọc***
* ***Phan Hoàng Trung***
* ***Lê Thanh Lâm***
* ***Võ Lê Huy***

***Lớp: ATUD***

**THÁNG 12, 2020**

**HỌC VIỆN KỸ THUẬT MẬT MÃ**

KHOA AN TOÀN THÔNG TIN

BÁO CÁO MẬT MÃ ỨNG DỤNG

TRONG AN TOÀN THÔNG TIN

ĐỀ TÀI:

**XÂY DỰNG ỨNG DỤNG SỬ DỤNG THƯ VIỆN OPENSSL**

Nhận xét của cán bộ hướng dẫn:.................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

Điểm chuyên cần:........................................................................................................

Điểm báo cáo:..............................................................................................................

**Xác nhận của cán bộ hướng dẫn**

Contents

[LỜI MỞ ĐẦU 4](#_Toc59038728)

[CHƯƠNG 1. GIỚI THIỆU THƯ VIỆN OPENSSL 5](#_Toc59038729)

[1.1. Nền tảng? 5](#_Toc59038730)

[1.2. SSL là gì? 5](#_Toc59038731)

[1.3. Giới thiệu về OpenSSL 6](#_Toc59038732)

[1.4. Các phiên bản của OpenSSL 6](#_Toc59038733)

[CHƯƠNG 2. KIẾN TRÚC OPENSSL 8](#_Toc59038734)

[2.1. Kiến trúc 8](#_Toc59038735)

[2.1.1. Kiến trúc thư mục openssl 8](#_Toc59038736)

[2.1.2. Kiến trúc thư mục crypto 9](#_Toc59038737)

[2.1.3. Kiến trúc Interface 10](#_Toc59038738)

[2.2. Các chức năng đơn giản của chương trình OpenSSL 10](#_Toc59038739)

[2.2.1. Sử dụng để băm dữ liệu 11](#_Toc59038743)

[2.2.2. Mã hóa dữ liệu 11](#_Toc59038748)

[2.2.3. Tạo mật khẩu đã được mã hóa 11](#_Toc59038754)

[2.2.4. Ký và xác thực chữ ký 12](#_Toc59038772)

[2.2.5. Tạo khóa RSA , DSA hay EC 12](#_Toc59038780)

[2.2.6. Kiểm tra số nguyên tố 12](#_Toc59038781)

[2.2.7. Kiểm tra hiệu suất phần cứng của hệ thống 13](#_Toc59038792)

[CHƯƠNG 3. CÁC THÀNH PHẦN MẬT MÃ TRONG OPENSSL 14](#_Toc59038793)

[3.1. Công cụ dòng lệnh 14](#_Toc59038794)

[3.2. Các thuật toán hỗ trợ 14](#_Toc59038797)

[3.2.1. Mật mã khóa bí mật 14](#_Toc59038803)

[3.2.3. Mật mã khóa bất đối xứng 22](#_Toc59039057)

[3.3. Các thành phần mật mã 23](#_Toc59039076)

[3.3.1. Mật mã khóa bí mật 23](#_Toc59039083)

[3.3.2. Mật mã khóa bất đối xứng 28](#_Toc59039142)

# LỜI MỞ ĐẦU

Công nghệ thông tin trong giai đoạn hiện nay đang có những bước phát triển như vũ bão trên mọi lĩnh vực hoạt động ở khắp nơi trên toàn thế giới, điều xảy ra trong vài năm vừa qua ở Việt Nam là sự đầu tư ồ ạt vào công nghệ. Tin học đã và đnag là một trong những vấn đề không thể thiếu đối với bất kỳ một tổ chức, công ty thậm chí cá nhân nào. Đặc biệt tin học ngày càng có vai trò quan trọng trong quá trình quảng bá cũng như giới thiệu về công ty hoặc cá nhân, do đó việc phát triển một website để quảng bá công ty hay một website cá nhân không còn gì xa lạ.

OpenSSL là 1 thư viện phần mềm cho toàn bộ các ứng dụng bảo mật truyền thông qua mạng máy tính phòng chống nghe trộm hoặc cần phải xác định phe truyền thông ở phía đầu bên kia. Chúng được ứng dụng rộng rãi trong các máy chủ web Internet nhằm phục vụ đa số các website hiện nay.

Do đó nhóm em đã lựa chọn “Xây dựng ứng dụng sử dụng thư viện OpenSSL” là đề tài trong báo cáo của mình.

**Nhóm em xin chân thành cảm ơn!**

# GIỚI THIỆU THƯ VIỆN OPENSSL

## Nền tảng?

Cầu nối tiếp cận OpenSSL API gồm có các nền tảng chính là ngôn ngữ lập trình C và mật mã học.

## SSL là gì?

SSL là từ viết tắt của Secure Sockets Layer, là tiêu chuẩn đằng sau giao tiếp an toàn trên Internet đã tích hợp mật mã học vào giao thức. Dữ liệu được mã hóa trước khi rời khỏi máy tính và chỉ được giải mã khi nó đến đích. Chứng thực và thuật toán mã hóa hiện diện ở tất cả các hoạt động, trong thực tế cả hai tương hỗ cho nhau và được sử dụng chung thay vì tách riêng.

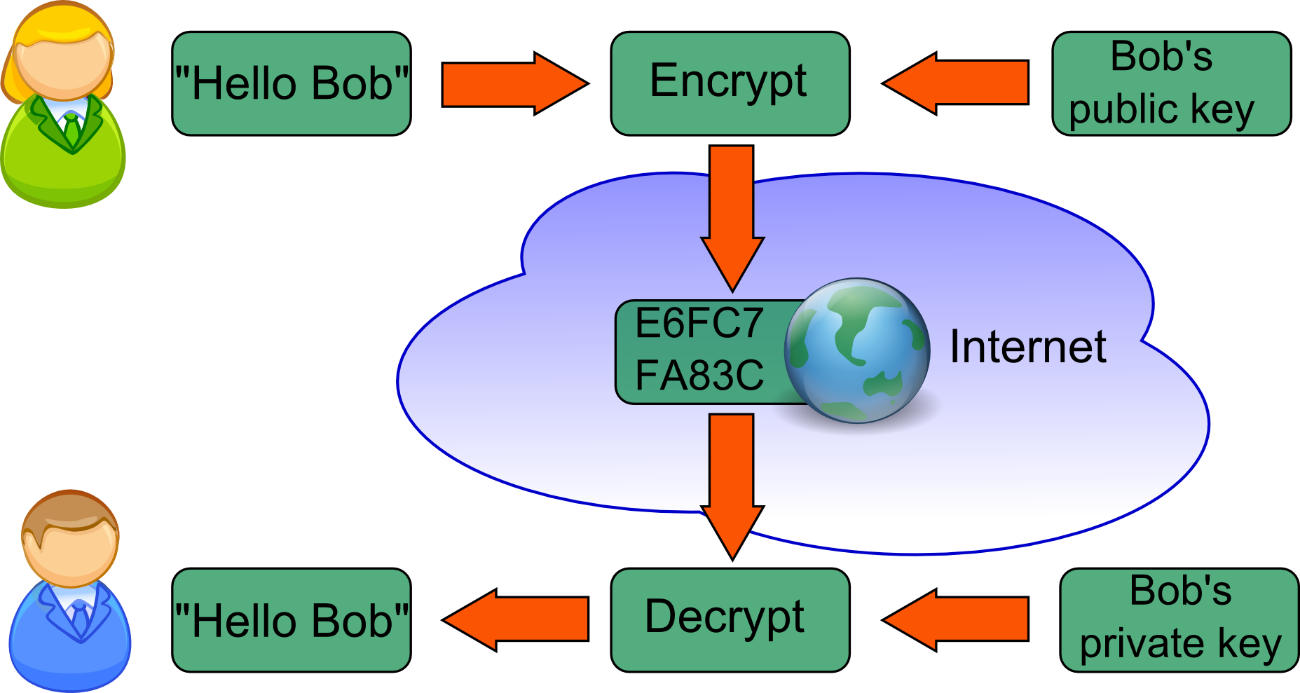


Figure 1. Mã hóa dữ liệu trong môi trường Internet.

Về mặt lý thuyết, nếu dữ liệu mã hóa bị chặn hoặc bị nghe trộm trước khi đến được người nhận thì kẻ tấn công gặp khó khăn trong việc giải mã dữ liệu vì không có bất kỳ hệ mã nào tránh khỏi việc bị bẻ khóa bởi phương pháp bạo lực. Nhưng hiệu năng của các hệ thống phần cứng nói riêng và máy tính nói chung ngày càng có tốc độ nhanh hơn, với những tiến bộ mới trong nghiên cứu toán học và phân tích mật mã được tạo ra thì xác suất bẻ khóa các hệ mật mã được sử dụng trong giao thức an toàn SSL đang bắt đầu tăng lên.

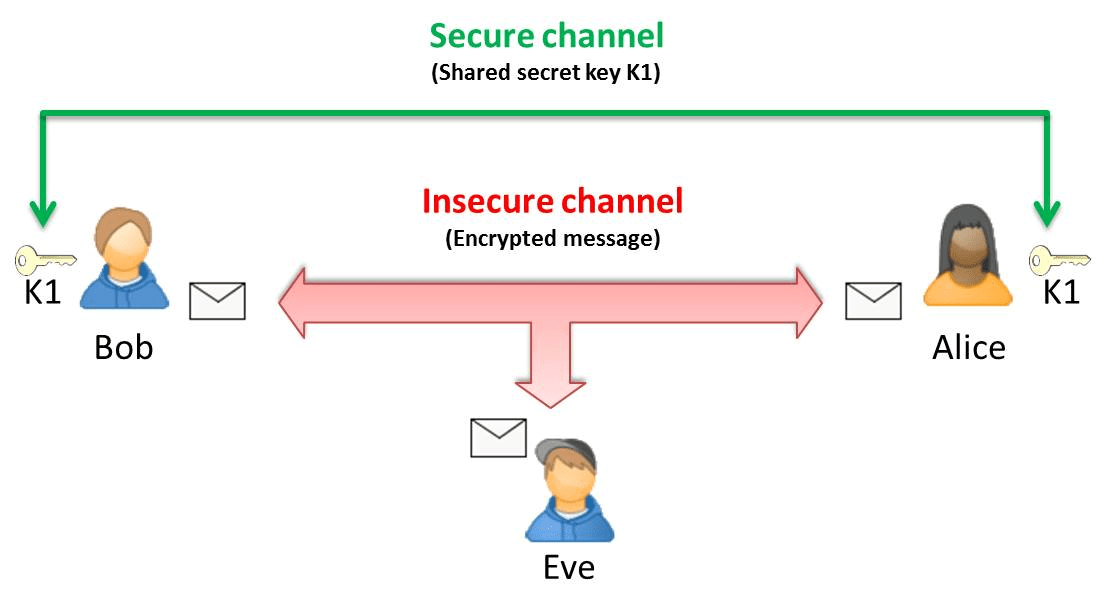


Figure 2. Kẻ tấn công bẻ khóa lấy dữ liệu.

SSL và các các giao thức kết nối an toàn có thể được sử dụng cho bất kỳ loại giao thức nào trên trên không gian mạng, cho dù đó là HTTP, POP3 hay FTP. Giao thức SSL cũng có thể được sử dụng để bảo mật các phiên telnet. Mặc dù mọi kết nối đều có thể được bảo mật bằng SSL, nhưng không nhất thiết phải sử dụng SSL trên mọi loại kết nối, SSL nên được sử dụng nếu kết nối mang thông tin nhạy cảm.

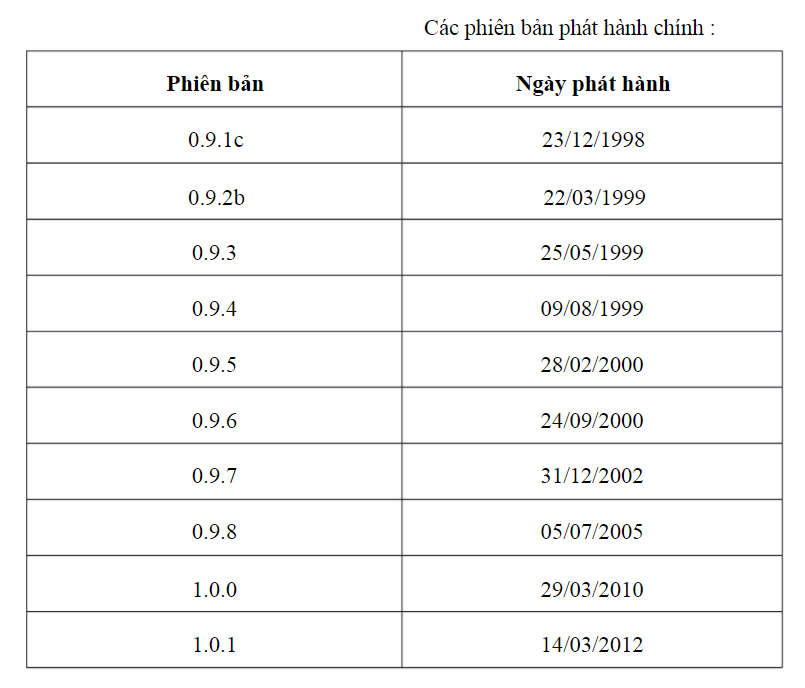
* 1. Giới thiệu về OpenSSL

OpenSSL là thư viện mã nguồn mở, không chỉ là có SSL nó còn có khả năng chạy thuật toán phức tạp để đào dữ liệu, mã hóa và giải mã các tệp, chứng chỉ số, chữ ký số. OpenSSL API tập hợp rất nhiều các cách giải quyết vấn đề, khả năng mà nó mang lại nhiều hơn những gì mà thư viện chuẩn ngôn ngữ lập C hiện có.

OpenSSL không chỉ là API hỗ trợ trong lập trình mà còn là một công cụ dòng lệnh. Công cụ dòng lệnh có thể làm những điều tương tự như API nhưng đã được tích hợp hỗ trợ cho việc kết nối máy chủ và máy khách. Nhờ vào API nó sẽ cung cấp việc nghiên cứu phát triển ý tưởng dựa trên nên tảng mà nó mang lại.

## Các phiên bản của OpenSSL

Các thư viện của OpenSSL được viết bằng ngôn ngữ lập trình C, nó hoạt động trên tất cả các nền tảng lớn, bao gồm tất các các hệ điều hành Unix (Linux, Solaris) và tất cả các phiên bản của Microsoft Windows (Windows XP, Vista, 7).



# KIẾN TRÚC OPENSSL

## Kiến trúc

### Kiến trúc thư mục openssl

Thư viện phân mục rất rõ ràng và mạch lạc bao gồm 7 mục chính:

* Apps: nơi chứa công cụ dòng lệnh OpenSSL, tức chương trình mà phần gới thiệu đã đề cập.
* Crypto: nơi chứa thư viện mật mã.
* Ssl: nơi chứa thư viện SSL dùng trong an toàn liên lạc ở tầng Session trong mô hình OSI.
* Demos: nơi chứa những bản demo được thể hiện ở dạng mã nguồn, là thành phẩm của việc lắp ráp và sử dụng api tạo nên một sản phẩm nhỏ.
* Docs: nơi chứa tài liệu tham khảo về các api.
* Engines: nơi chứa cung cấp driver hỗ trợ sử dụng tài nguyên phần cứng trong việc tăng tốc hiệu năng.
* Include: nơi chứa tệp tin header \*.h.
* VMS, ms: nơi chứa chỉ định nền tảng, tên sẽ được thay đổi tùy theo hệ điều hành được sử dụng. Ví dụ: MACOS nếu sử dụng hệ điều hành của Apple, ...
* Test: nơi chứa những bài test, chủ yếu là các thuật toán được hỗ trợ.
* Util: Per shell được tạo ra bởi mã C.

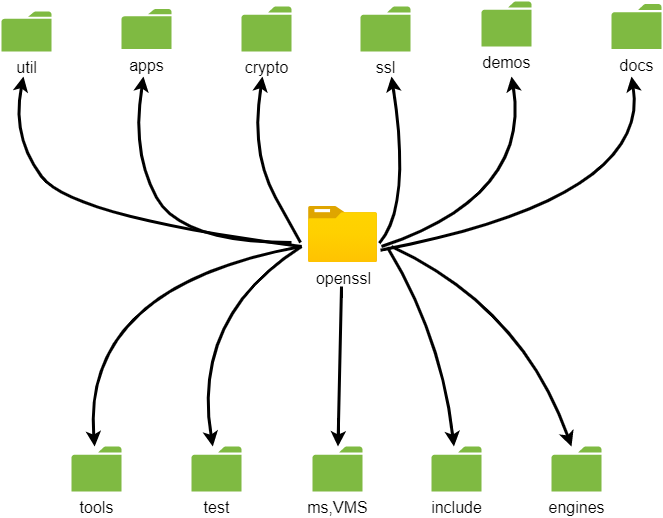


Figure 3. Kiến trúc trong thư mục openssl.

### Kiến trúc thư mục crypto

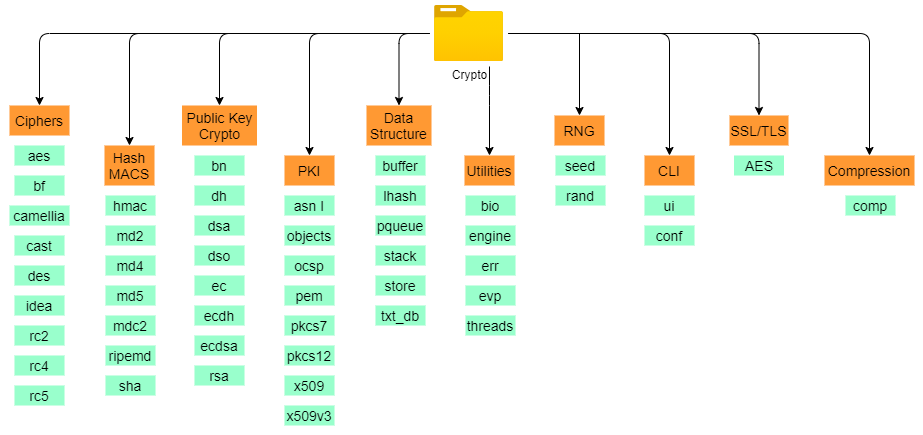


Figure 4. Kiến trúc mật mã học trong thư mục crypto.

### Kiến trúc Interface

Thư viện OpenSSL cung cấp 2 interface để làm việc với các hàm mật mã được gọi là interface thấp (low interface) và interface cao (high interface).

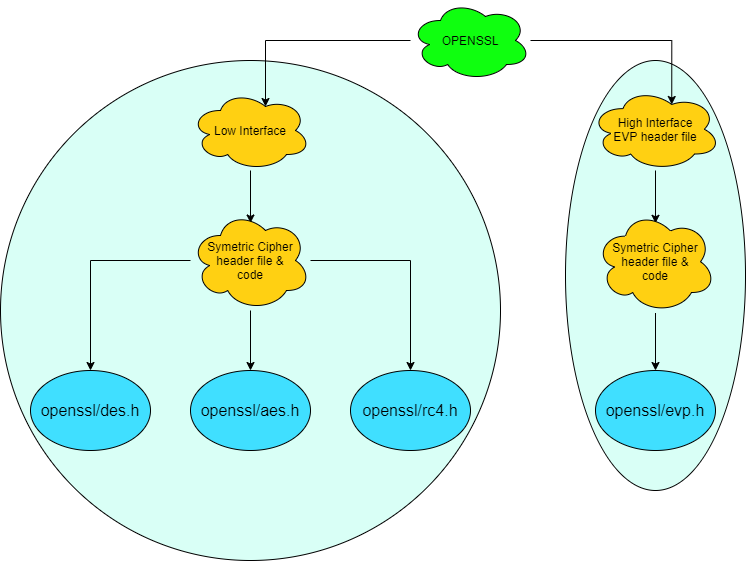


Figure 5. Kiến trúc Interface.

Giống như ngôn ngữ bậc thấp C và ngôn ngữ bậc cao C#, với high interface vẫn được lập trình bằng ngôn ngữ C nhưng lại mang tinh thần hướng đối tượng giúp cho việc chuyển đổi, bảo trì mã nguồn một cách nhanh chóng và dễ dàng. Nhưng không vì thế mà phủ nhận low interface, cũng như phần giới thiệu đã nêu rằng OpenSSL có tài liệu rất ít và không phổ biến bằng các thư viện chuẩn có sẵn trong C# hay Java, low interface giúp nghiên cứu và debug chương trình, hiểu được tường tận cách mà mật mã học hoạt động.

## Các chức năng đơn giản của chương trình OpenSSL

Chương trình OpenSSL là một công cụ dòng lệnh để sử dụng các chức năng mật mã của các thư viện Crypto của OpenSSL.Nói tóm lại, OpenSSL là một thư viện phần mềm cung cấp một bộ công cụ mật mã đầy đủ tính năng như một triển khai của SSL. Phiên bản Unix, mà chúng tôi sẽ sử dụng, cung cấp hai thư viện, một cho SSL / TLS (ssl) và một cho bộ công cụ mật mã (tiền điện tử) cũng như công cụ dòng lệnh (OpenSSL).



### Sử dụng để băm dữ liệu

OpenSSL hỗ trợ rất nhiều thuật toán băm: MD5, SHA, MDC2, RIPEMD160,… Để băm dữ liệu, sử dụng lệnh: “dgst”

Cú pháp: openssl dgst [Kiểu băm] [Tên file]

Ví dụ: openssl dgst –md5 demo.txt

Các kiểu băm được hỗ trợ là: md2, md4, md5, sha, sha1, sha256, sha512, mdc2.



### Mã hóa dữ liệu

Dữ liệu có thể mã hóa bằng các thuật toán mã hóa đối xứng như: aes, base64, des, blowfish… bằng cách sử dụng lệnh “enc”.

Cú pháp: openssl enc [Các tùy chọn]

Các tùy chọn cơ bản là:

* -in: tệp tin cần mã hóa
* -out: tệp tin sẽ tạo ra
* -e: mã hóa
* -d: giải mã
* -des, -des3, -aes256… là các thuật toán dùng để mã hóa

Ví dụ: openssl end –e –aes256 –in demo.txt –out demo.txt.ae.



### Tạo mật khẩu đã được mã hóa

Lệnh này hữu ích khi cần sửa chửa trực tiếp các file chứa mật khẩu của Linux chẳng hạn như /etc/passwd chẳng hạn. Cú pháp lệnh là:

openssl passwd [Chuỗi cần mã hóa]

Các tùy chọn cơ bản:

* -crypt: chuẩn Linux
* -1: mã hóa dựa trên md5

Ví dụ: openssl passwd “password123”



### Ký và xác thực chữ ký

Để ký cho file demo.tar.gz ta sẽ dùng khóa riêng. Lệnh sau sẽ cho ra file demo.tar.gz.sha1.

openssl dgst -sha1 -sign mykey.pem -out demo.tar.gz.sha1 demo.tar.gz

Khi xác thực bạn sử dụng lệnh sau:

openssl dgst -sha1 -verify pubkey.pem -signature \ demo.tar.gz.sha1 demo.tar.gz.



### Tạo khóa RSA , DSA hay EC

Để tạo khóa bí mật RSA ta có thể dùng lệnh:

openssl genrsa-out mykey.pem 1024

Trong đó mykey.pem là file lưu khóa, 1024 là số bit khóa. Nếu muốn mã hóa luôn file chứa khóa ta có thể dùng lệnh sau:

openssl genrsa-des3 ra mykey.pem 1024

Tạo khóa công khai RSA từ khóa bí mật:

openssl genrsa-in mykey.pem –pubout –out mykey.pub

Tạo các thông số cho khóa DSA:

openssl dsaparam ra dsaparam.pem 1024

Tạo khóa DSA thứ nhất:

openssl gendsa ra key1.pem dsaparam.pem

Tạo khóa DSA thứ hai:

openssl gendsa key2.pem dsaparam.pem

### Kiểm tra số nguyên tố

Chúng ta có thể kiểm tra xem một số phải là số nguyên tố hay không bằng lệnh:

openssl prime –option [number]

Trong đó:

* –option có thể là hex , dec...
* [number] số cần kiểm tra.



### Kiểm tra hiệu suất phần cứng của hệ thống

Chúng ta có thể đo hiệu suất hoạt động của hệ thống phần cứng bằng cách dùng lệnh:

openssl speed

Cách đo ở đây là OpenSSL sẽ kiểm tra xem trong một thời điểm hệ thống có thể chạy được bao nhiêu thuật toán chứ không phải kiểm tra xem một thuật toán thì hệ thống chạy hết bao nhiêu thời gian.

Ngoài ra, OpenSSL còn hỗ trợ các chức năng khác như thiết lập kết nối với một máy chủ SMTP an toàn , thiết lập một máy chủ SSL từ dòng lệnh, xác minh, mã hóa và ký một thông điệp S/MIME.

# CÁC THÀNH PHẦN MẬT MÃ TRONG OPENSSL

## Công cụ dòng lệnh

Chương trình OpenSSL là một công cụ dòng lệnh để sử dụng các mật mã khác nhau của thư viện của thư viện mật mã trong OpenSSL. Một số chức năng mà chương trình có thể làm được:

* Tạo khóa RSA, DH và DSA.
* Tạo chứng chỉ X.509, CSRs và CRLs.
* Tính toán bản tóm tắt thông điệp – Message Digests.
* Mã hóa và giải mã.
* Xây dựng nhanh mô hình Server – Client thông qua giao thức SSL/TLS.
* Xử lý S/MIME đã ký và mail được mã hóa.



## Các thuật toán hỗ trợ







### Mật mã khóa bí mật

Mật mã đối xứng là hệ mật mà quá trình mã hóa và quá trình giải mã dùng chung một khóa bí mật, vì thế phát sinh thêm câu chuyện bảo vệ khóa bí mật, việc bảo mật bản tin cũng phụ thuộc vào việc lưu khóa này.

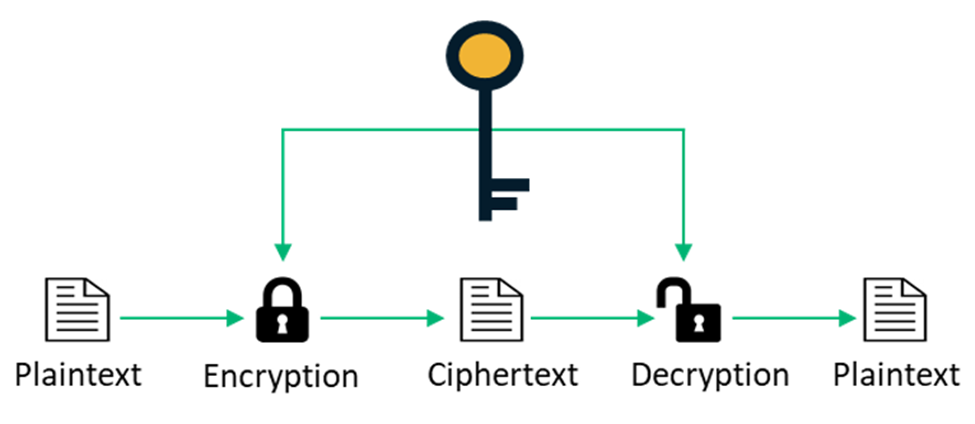


Figure 6. Mô hình mật mã đối xứng.

Như mô hình phía trên, bản tin plantext là đối tượng cần được bảo vệ, trước khi chuyển đến nơi nhận nó phải được mã hóa, cách thức mã hóa là tổng hợp các phép biến đổi với sự tham gia của khóa bí mật để cho ra được bản mã ciphertext. Để giải mã bản mã sử dụng cùng cách thức mã hóa sẽ cho ra bản tin như ban đầu. Việc lựa chọn thuật toán mã hóa sẽ được lựa chọn sao cho phù hợp với tình huống để đảm bảo được hiệu xuất và tính dễ dàng triển khai.

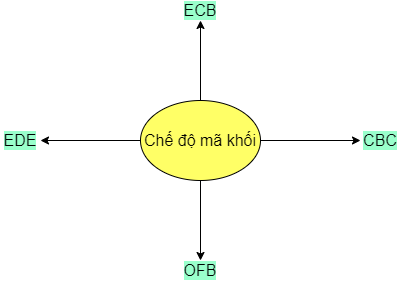


Figure 7. Các sơ đồ mã khối tiêu biểu

Thuật toán mã hóa và giải mã áp dụng lên từng khối dữ liệu đó cũng là tiền đề sơ khai của sơ đồ khối ECB. Ngoài ra còn các một số sơ đồ mã khối khác như CBC, CTR, EDE, v.v ...



#### Chế độ mã khối ECB

Trong sơ đồ mã khối ECB, mỗi lần mã hay giải mã bản tin đều sử dụng chung một khóa K, tức xuyên suốt trong quá trình khóa K không bị biến đối dẫn đến sẽ có một số khối đầu ra giống nhau.

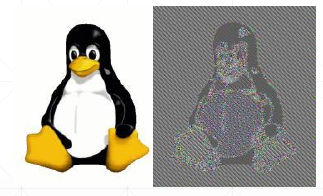


Figure 8. Bản mã không được mã hóa tốt

Điều này tạo nên điểm yếu của ECB, người phá mã có thể dựa vào một số đặc tính thống kê của dữ liệu để tiến hành đoán thu nhỏ phạm vi phỏng đoán và phá mã.

#### Chế độ mã khối CBC

Trong sơ đồ mã khối CBC, bản tin được mã sẽ được dùng để mã hóa khối kế tiếp, cho thấy được mắt xích chặt chẽ rằng bản mã không chỉ phụ thuộc vào bản rõ và khóa như ECB mà còn phụ thuộc vào các kết quả nền tảng trước.

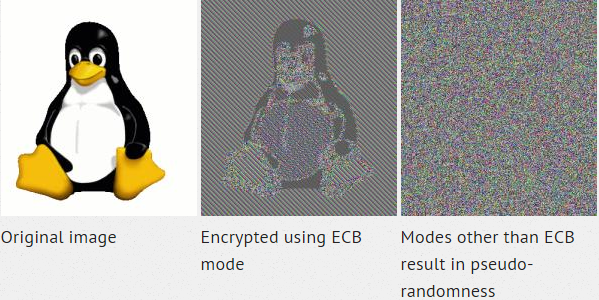


Figure 9. Bản mã được che giấu tốt hơn ECB.

Tương tự với các mã khối OFB, CFB, EDE, v.v ... đều có chung một tình thần giống CBC, tất cả đều có cách thức riêng để cho ra bản mã được che giấu tốt nhất.

#### Mật mã thuộc loại mã khối

Là tổ hợp lệnh toán học (hoán vị, thay thế, ...) biến đổi dãy bit đầu vào thành đầu ra có cùng kích thuốc với sự tham gia của khóa bí mật.



##### DES (Data Encryption Standard)

Mã DES là mã thuộc hệ mã Feistel gồm 16 vòng, ngoài ra DES còn thêm một hoán vị khởi tạo trước khi vào vòng 1 và một hoán vị khởi tạo sau vòng 16.

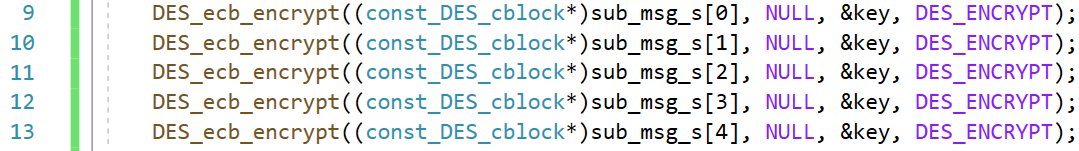


Figure 10. Mật mã DES ECB với low interface.

Nó thuộc vào loại mã khối có kích thước khối là 64 bit được biểu diễn theo mã ASCII, mã DES sẽ chỉ làm việc với từng khối 64 bit của bản tin, tức chia bản tin thành các khối con và làm việc cho đến khi hoàn tất việc mã hóa hoặc giải mã.

Đoạn mã trên đặt trọng tâm vào việc dễ tiếp cận, để làm được việc này thì cần sự rõ ràng được định nghĩa low interface, Khi đã nắm bắt được thì việc triển khai trong ứng dụng trở nên dễ dàng hơn với high interface với EVP được OpenSSL cung cấp.



##### 3DES (Triple DES)

Tương tự với DES nhưng Triple DES cần đến 3 khóa. Ví dụ như sau:

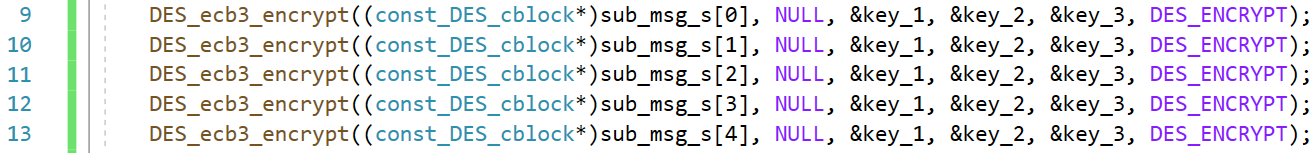


Figure 11. Mật mã Triple DES với low interface.

Là mã được sinh ra để khắc phục nhược điểm độ an toàn của khóa của mã DES bằng cách tạo ra 3 khóa thay vì tăng chiều dài khóa lên gấp 3 lần, sử dụng ba khóa đó cho ba lần mã hoặc giải mã. Chính vì chiều dài được tăng lên thành 168 bit nên sẽ gây phức tạp hơn nhiều cho việc phá mã.



##### AES (Advanced Encryption Standard)

Mã AES là mã tiêu chuẩn mới của Cục Tiêu Chuẩn Quốc Gia Hoa Kỳ thay thế cho mã DES và còn là kết quả của sự kế thừa bởi khắc phục nhược điểm khóa có yếu tố bảo mật kém bởi độ dài. Giống như DES, mã AES là một mã khối gồm nhiều vòng, mã AES với khóa có kích thước 256 bit được cho là an toàn trong tốc độ phát triển nghành kỹ thuật máy tính ngày nay. Độ an toàn của AES được đặt kỳ vọng nhiều hơn DES và TripleDES và được dự đoán sẽ thay thế vai trò của DES và TripleDES trong tương lai.

Một số đặc điểm chính của AES:

* Dễ dàng có thể hiểu được các thuật ngữ AES 128/192/256 chính là đầu vào sẽ được chia thành các khối có độ dài 128/192/256 bit.
* Ngoài ra nó còn cho phép độc lập về mặt kích thước giữa đầu vào và khóa.
* Số lượng vòng có thể thay đổi từ 10 đến 14 vòng tùy thuộc vào kích thước của khóa.



##### Blowfish

Là thuật toán có độ an toàn cao và dễ thực hiện hơn DES, giống với DES thuật toán này có 16 vòng được xây dựng trên cơ sở sơ đồ Feistel. Mã Blowfish mã khối dữ liệu 64 bit với chiều dài khóa từ 32 đến 128 bit.



##### Camellia

Là loại mật mã được phát triển bởi công ty Mitsubishi và NTT tại Nhật Bản vào năm 2000. Camellia được thiết kế để mang lại hiệu quả cho cả việc triển khai phần mềm, phần cứng và nó được sử dụng trong nhiều thiết bị khác nhau từ thẻ thông minh chi phi thấp đến giao thức mạng tốc độ cao.

Camellia sử dụng khóa có 3 kích thước là 128, 192 và 256 bit. Khối thông tin được mã có chiều dài 182 bit.



##### CAST

Là mật mã được sử dụng trong một số sản phẩm mặc định trong một số phiên của của GPG và PGP. CAST được chính phủ Canada được sử dụng trong Cơ quan An ninh truyền thông. Thuật toán này được tạo vào năm 1996 và được viết tắt bởi bởi Carlisle Adams và Stafford Tavares.

Một số thành viên của họ nhà mật mã CAST là CAST-256 đươc phát triển từ CAST-128 - một ứng cử viên cạnh tranh trực tiếp với AES cho nỗ lực trở thành mật mã tiêu chuẩn của Cục tiêu chuẩn quốc gia Hoa Kỳ.

CAST-128 cũng được xây dựng dựa trên sơ đồ Feistel với 12 hoặc 16 vòng, mã bản tin có kích thước khối 64 bit và kích thước khóa từ 40 đến 128 bit, tuy nhiên đối với Tiêu chuẩn quốc gia Việt Nam TCVN 11367 (ISO/IEC-18033) chỉ sử dụng phương án khóa 128 bit.



##### Idea

Là hệ mật được xây dựng bởi hai thành viên của Viện công nghệ Thụy Điển là Xuejla Lai và James Massey. Phiên bản đầu tiên được công bố vào năm 1990 dưới cái tên PES (Proposed Encryption Standard). Một năm sau đó, sau khi Biham và Shamir công bố phương pháp thám mã mới là “thám vi phân” thì các tác giả đã cái biên thuật toán PES để chống lại. Bản cải tiến được đặt tên là IPES (Improved PES) và cuối cùng được đổi tên thành IDEA vào năm 1992.

IDEA mã bản tin và giải mã theo từng khối 64 bit. Có chiều dài khóa 128 bit và thực hiện 9 vòng.



##### RC2

Tác giả của mật mã RC2 tên là Ron Rivest, một trong 3 tác giả của hệ mật nổi tiếng RSA. RC2 sử dụng khối có độ dài 64 bit, khóa có độ dài từ 8 đến 1024 bit. Thuật toán được thiết kế để có thể dễ dàng và hiệu quả trong việc triển khai trong hệ thống với bộ vi xử lý 16 bit. Tốc độ của RC2 nhanh hơn rất nhiều so với mã DES. Mã RC2 không phải là mã nguồn mở, nó thuộc sở hữu của công ty RSA Security Inc.



##### RC5

Giống như RC2 đây cũng là sản phẩm của tác giả Ron Rivest đề xuất năm 1994. Nhưng hệ này tỏ ra mềm dẻo hơn trong việc lựa chọn tham số cho hệ mật, cụ thể là kích thuốc khối mã có thể là 32, 64, 128 và khóa có độ dài dưới 2040 bit, số vòng từ 0 đến 255. Ban đầu mã được đề xuất sử dụng khối mã 64 bit, khóa 128 bit và số vòng là 12.



##### Hàm băm

Khi thế giới xuất hiện những người tấn công với ý tưởng tinh vi làm thay đổi mã nguồn của tác giả trước khi kịp đến với người sử dụng, nhu cầu kiểm tra chứng thực phần mềm nói riêng và chương trình nói chung ngày càng cần thiết. Ý tưởng đầu tiên xuất hiện để kiểm tra giữa mã nguồn sạch và mã độc là so sánh từng bit nhưng trở ngại đầu tiên là về mặt kích thước chương trình. Vấn đề này sẽ được hàm băm giải quyết thông qua cố định kích thước đầu ra.

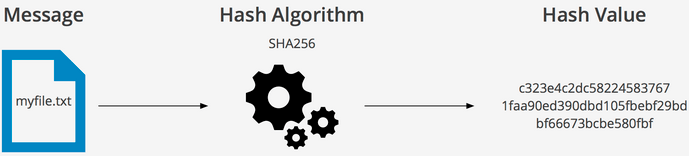


Figure 12. Sơ đồ hoạt động hàm băm.

Người tấn công sẽ phải tìm cách tiêm mã độc mà vẫn cho ra kết quả băm giống như công bố mã nguồn sạch từ tác giả, nhưng các khoa học đã thiết kế ra những hàm băm có tính chất nhạy cảm, chỉ cần có một bit thay đổi ở đầu vào sẽ cho ra kết quả khác hoàn toàn so với bản gốc.



###### MD

MD5 (MD5 Message-Digest Algorithm) là một thuật toán tóm tắt thông điệp, là một hàm băm mã hóa được dùng để chứng thực sự toàn vẹn của nội dung.

Nội dung sau khi băm qua thuật toán MD5 kết quả thu được là một Checksum, được biết với tên khác là hash code hoặc hash value – có kích thước cố định 32 ký tự hexa tương đương với 128 bit, được dùng để so sánh thay vì phải dùng nội dung dữ liệu để so sánh.

MD5 có một số lỗi nhất định chính vì vậy nó thường không được dùng cho mã hóa nâng cao. Ronald Rivest công bố MD5 vào năm 1992 trên máy tính 32bit, nó là một trong 3 thuật toán của ông, trước đó đã có MD2 (1989) trên máy tính 8 bit và MD4 (1990) trên máy tính 32 bit. MD2 chứa nhiều sự phàn nàn về tốc độ trong khi MD4 có tốc độ nhanh hơn MD5, nhưng MD5 được dùng nhiều hơn vì nó bù lại được sự bảo mật.



###### SHA

Là một trong những thuật toán băm mã hóa, được dùng trong việc kiểm tra tính toàn vẹn của dữ liệu ở phía người nhận. cũng giống như MD5, nhạy cảm ở đầu vào, bất kỳ sự thay đổi bit nào cũng dẫn đến kết quá khác hoàn toàn. SHA vượt trội hơn trong vấn đề bảo mật nhưng cũng vì thế cần có thời gian nhiều hơn để xử lý.

SHA-0 có kích 160 bit và là phiên bản đầu tiên của thuật toán này. Giá trị băm SHA-0 dài 40 ký tự hexa tương ứng với 20 byte. Nó được xuất bản lần đầu tiên với tên SHA vào năm 1993 nhưng không được sử dụng trong nhiều ứng dụng vì nó nhanh chóng được thay thế bằng SHA-1 vào năm 1995 do lỗi bảo mật.

SHA-1 là một phiên bản trong các phiên bản của thuật toán băm Secure Hash Algorithm (SHA). Hầu hết được phát triển bởi Cục An Ninh Quốc Gia (NSA) và được Viện Tiêu Chuẩn Và Kỹ Thuật Quốc Gia (NIST) công bố.

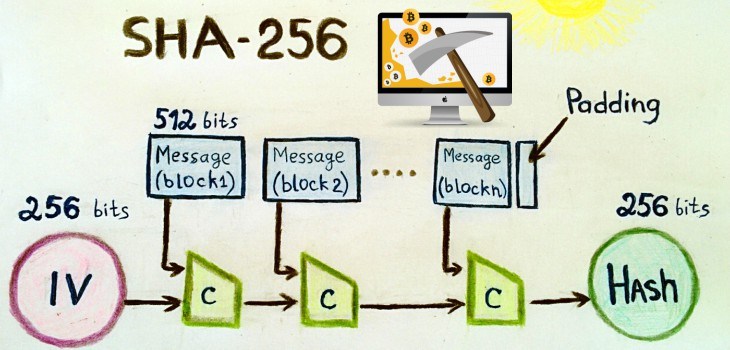


Figure 13. Hàm băm SHA-256.

SHA-256 cũng giống như SHA-1 nhằm để tóm tắt dữ liệu, SHA-256 và tăng cường hơn khả năng bảo mật.



###### MAC và HMAC

MAC là một thông điệp được dùng để chứng thực, là một dạng checksum của mã hóa.

HMAC cũng có thể dùng để tính MAC bằng cách truyền thêm khóa bí mật vào hàm băm, đầu ra lúc này được gọi là HMAC.



#### Mật mã thuộc loại mã dòng

Là hệ mã đối xứng, trong đó từng ký tự của bản rõ được biến đổi thành ký tự của bản mã, phụ thuộc không chỉ vào khóa sử dụng mà còn vào vị trí của nó trong bản rõ.



##### RC4

Thuộc hệ mã dòng với chiều dài khóa biến đổi, tác giả của nó là Ronald Rivest xuất bản năm 1987. Mấu chốt chính của thuật toán là cách thức tạo khóa dòng, hàm sẽ tạo dãy bit sau đó cộng với bản tin theo module 2 để thu được bản mã. RC4 làm việc ở chế độ OFB.



### Mật mã khóa bất đối xứng

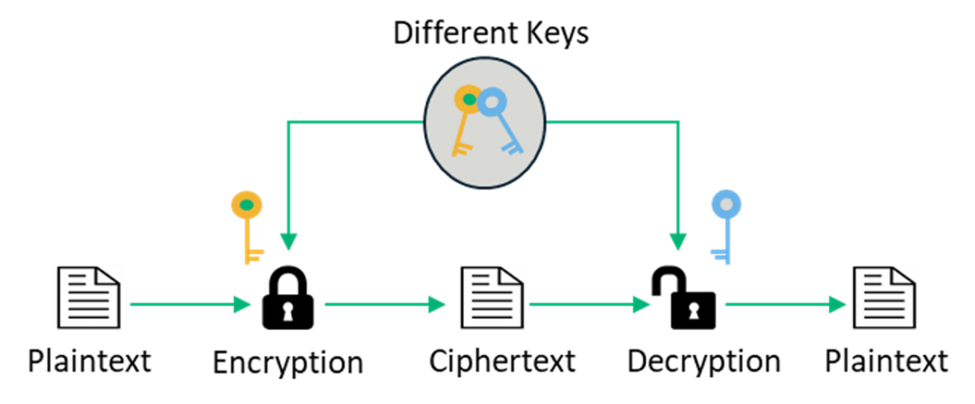


Figure 14. Mã hóa bất đối xứng.



#### DH (Diffie – Hellman)

Là một phương pháp trao đổi khóa dùng để thiết lập một khóa bí mật giữa người gửi và người nhận mà không cần dùng đến mã hóa công khai. Là một trong những phương pháp tạo nên độ an toàn của RSA.

#### RSA

RSA được xây dựng bởi tác giả Ron Rivest, Adi Shamir và LenAdleman tại Học viện công nghệ MIT vào năm 1977. Ngày nay mật mã RSA vẫn được sử dụng và hơn hết là tính phổ biến của nó rất cao trong ứng dụng.

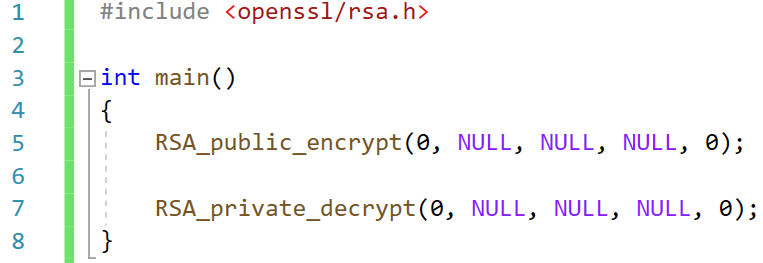


Figure 15. Hai hàm mã hóa và giải mã.

Lưu ý rằng trường hợp sử dụng chu trình dùng khóa công khai để mã hóa và khóa bí mật để giải mã. Nếu dùng ngược lại sẽ mang một ý nghĩa khác chính là chứng thức bằng chữ ký số với khóa bí mật được dùng để mã hóa và khóa công khai được dùng để giải mã.



Figure 16. Quá trình tạo chữ ký số.



## Các thành phần mật mã



### Mật mã khóa bí mật



#### Mật mã khóa đối xứng

Mã hóa đối xứng trong OpenSSL được thực hiện thông qua thư viện high interface EVP cung cấp đầy đủ các chức năng như low interface. Thư viện OpenSSL hỗ trợ ngôn ngữ lập trình C với tinh thần hướng đến OOP dễ giảm bớt vấn đề thay đổi qua nhiều mã nguồn trong khi thực tế việc đơn giản chỉ cần đổi hệ mã.

EVP sử dụng loại dữ liệu gọi là EVP\_CIPHER\_CTX để quản lý sử dụng loại mật mã, khóa, IV, v.v ...



##### Khởi tạo hệ mật

Để khởi tạo ngữ cảnh cho một mật mã, EVP có cú pháp như sau :

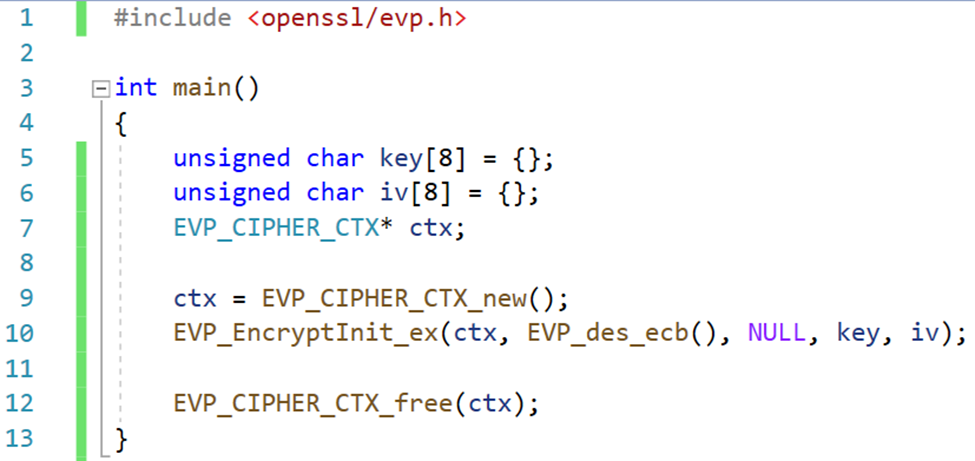


Figure 17. Khởi tạo ngữ cảnh bằng EVP.

Hàm **EVP\_CIPHER\_CTX\_new** sẽ tạo và trả về context. Sử dụng các hệ mật đối xứng thông qua biến ctx.

Hàm **EVP\_EncryptInit\_ex** sẽ chỉ định mật mã, khóa, iv được dùng. Lưu ý chương trình sẽ không bị crash hoặc gây lỗi nếu như hệ mật mã không không hỗ trợ IV, ví dụ chế độ mã khối ECB không hỗ trợ IV và kết quả hàm trả về vãn là giá trị 1 đại diện cho thành công, ngược lại là thất bại.

Hàm **EVP\_CIPHER\_CTX\_free** là một phiên bản nâng cấp so với hàm. **EVP\_CIPHER\_CTX\_cleanup** trước đó, ngoài việc trả bộ nhớ xin cấp phát về cho hệ điều hành thì trước đó nó sẽ tẩy tất cả thông tin hệ mật và không để lại bất kỳ thông tin mang yếu tố nhạy cảm còn lại trên bộ nhớ.



##### Sử dụng hệ mật

Để sử dụng hệ mật, OpenSSL EVP cung cấp duy nhất một hàm làm mọi thao tác từ việc chọn chế độ mã hóa hay giải mã đến việc tự động thao tác hết khối đầu vào mà không cần phải tạo thêm vòng lặp như low interface.

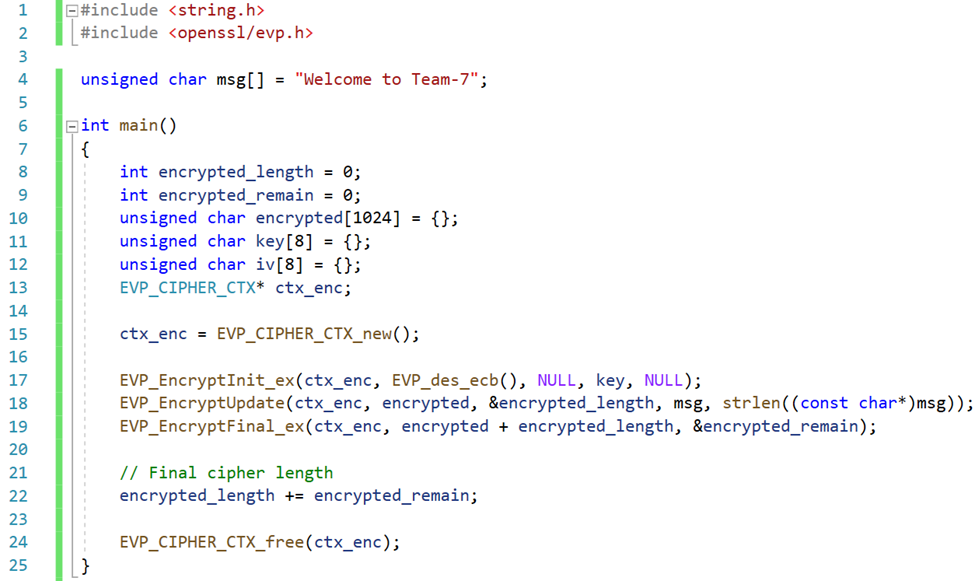


Figure 18. Mã hóa DES chế độ ECB bằng EVP

EVP cung cấp chu trình gồm 2 bước là mã hóa dữ liệu cần thiết và những dữ liệu được padding.

Hàm **EVP\_EncryptUpdate** có đối số thứ nhất là ngữ cảnh mã hóa tức thông báo cho hàm biết sử dụng hệ mật mã và chế độ mã khối đã được khai báo để làm công việc của chính nó. Đối số thứ hai là nơi chứa bản mã. Đối số thứ ba là chiều dài mà hàm đã thực hiện mã bản tin. Đối số cuối cùng để khai báo IV nhưng đối với chế độ mã khối ECB thì hàm sẽ bỏ qua IV cho dù có khai báo.

Hàm **EVP\_EncryptUpdate** mã hóa chiều dài dữ liệu lớn nhất thỏa bội số của kích thước khối, phần dữ liệu chưa được mã hóa do không đầy đủ về kích thước nên không thể tham gia vào thuật toán mã hóa sẽ được xử lý bởi hàm mang tên final.

Trong thực tế sẽ không có đầu vào hoàn hảo là bội số của của kích thước khối, chính vì thế phần dữ liệu chưa được đầy đủ sẽ được gọi bởi thuật toán padding và mã hóa bởi **EVP\_EncryptFinal\_Ex.**

Tương tự đối với quá trình giải mã, các hàm phải được hay thay thế một ít về mặt từ ngữ và vẫn sẽ tuân thủ theo quy trình khởi tạo, mã đầu vào và mã đầu vào không chuẩn còn lại thông qua 3 hàm:

EVP\_DecryptInit\_ex

EVP\_DecryptUpdate

EVP\_DecryptFinal\_ex

Vì để rút gọn thêm quy trình này EVP đã cho ra các hàm vẫn tuân thủ theo quy trình cũ, nhưng ở chế độ mã hóa và giải mã sẽ là một tùy chọn trong đối số, vì thế rút gọn được quá trình phát triển ứng dụng.

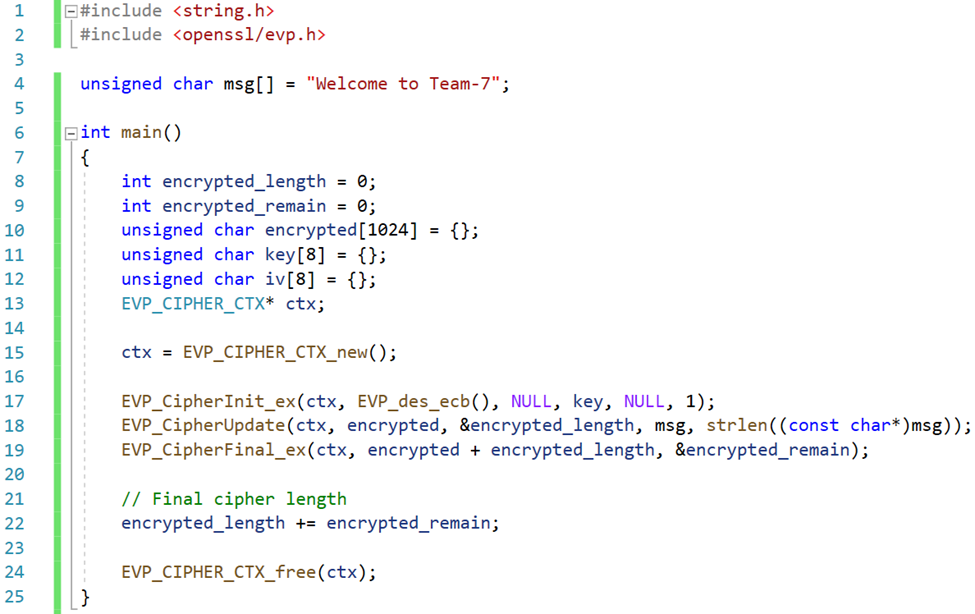


Figure 19. Mã hóa DES chế độ ECB bằng EVP tối ưu.

Giá trị 1 được đưa vào **EVP\_CipherInit\_ex** chỉ định cho việc mã hóa, ngược lại sẽ là 0.



#### Hàm băm (Tóm tắt thông điệp)



##### Khởi tạo ngữ cảnh tóm tắt

Khi sử dụng hàm băm tóm tắt thông điệp, EVP yêu cầu khởi tạo một cấu trúc ngữ cảnh để có thể phân biệt được thuật toán sắp sử dụng. Ví dụ: MD5 và SHA1.

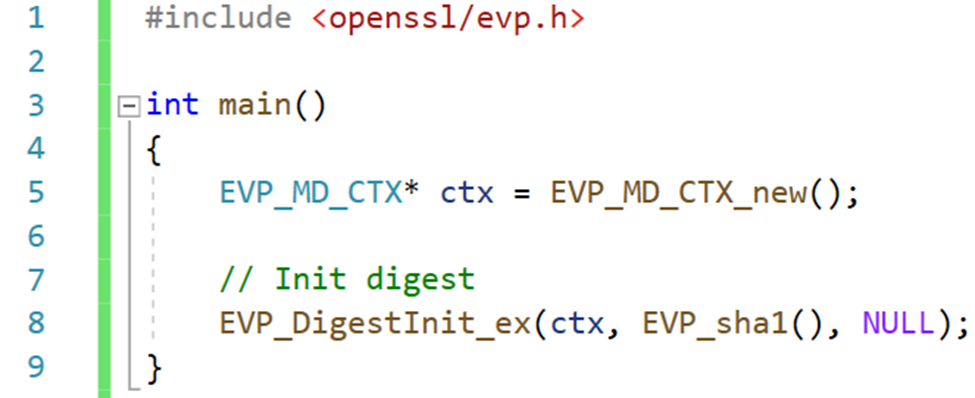


Figure 20. Khởi tạo hàm băm tóm tắt SHA bằng EVP.



##### Sử dụng hàm băm

Giống như mã bản tin, hàm băm làm việc theo khối, mỗi khối dữ liệu được băm sẽ được thêm vào nơi lưu trữ, quản lý bởi biến digest. Hàm **EVP\_DigestUpdate** cần được chỉ đỉnh khối dữ liệu đầu vào. Khác với mã bản tin, đầu ra sẽ chỉ được chỉ định ở hàm **EVP\_DigestFinal\_ex**.

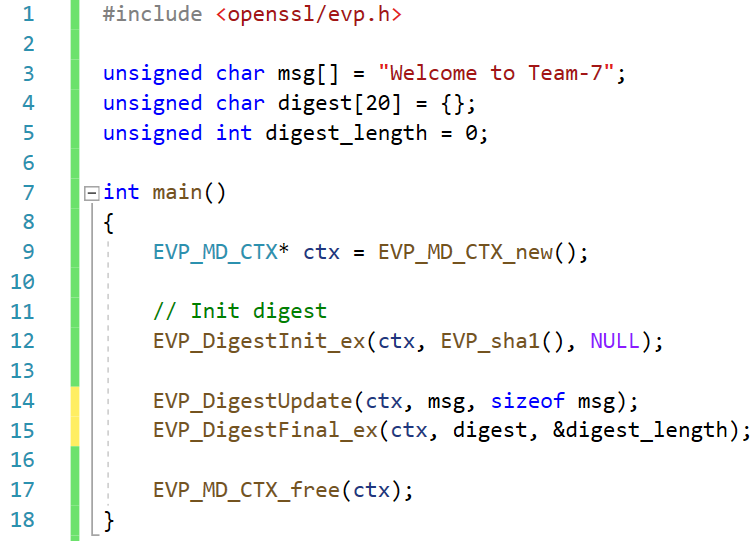


Figure 21. Thủ tục băm dữ liệu với SHA1.

Vì trên lý thuyết SHA1 cho ra kết quả có độ dài 160 bit tương ứng với 20 byte của bộ đệm digest.



### Mật mã khóa bất đối xứng



#### Hàm PEM (Privacy Enhanced Mail)

PEM là một định dạng cho việc encode các đối tượng mật mã, được viết tắt là Privacy Enhanced Mail. Ban đầu PEM được thiết kế cho PGP, nhưng nó đã trở thành một chuẩn định dạng để lưu trữ bản mã nhị phân trên ổ cứng. Đây là một định dạng tiêu chuẩn được sử dụng cho các khóa công khai và khóa riêng, chứng chỉ số, v.v ... Tệp tin có định dạng mở rộng pem.

Thư viện OpenSSL cung cấp các hàm, các cấu trúc dữ liệu để chuyển đổi từ đối tượng PEM sang OpenSSL và ngược lại. Ví dụ để đọc khóa RSA được lưu trong tệp tin định dạng PEM.



Figure 22. Đọc khóa bí mật từ tệp tin.

Mục đích của mã nguồn sẽ đọc khóa bí mật từ tệp tin có định dạng PEM và chuyển đổi sang đối tượng quản lý RSA trong OpenSSL thông qua con trỏ FILE đang hỗ trợ chế độ đọc. Nếu tệp tin không đúng định dạng hoặc không tồn tại, sử dụng cơ chế thông báo lỗi của thư viện để có được thông tin lỗi chi tiết.

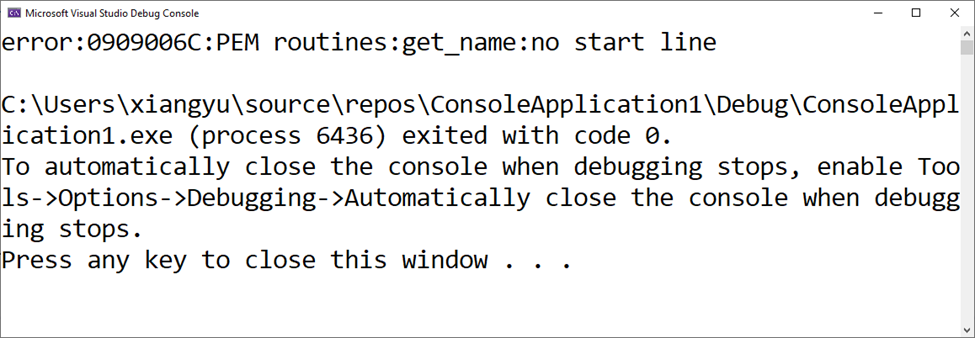


Figure 23. Kết quả khi đọc sai định dạng PEM.

Ở trường hợp ví dụ hàm **PEM\_read\_RSAPrivateKey** như trên thì 3 đối số sau dùng được đưa vào là NULL. Đối với các đối số thứ hai, cho phép chỉ định không gian lưu trữ đối tượng quản lý khóa. Đối số thứ ba và thứ tư chỉ định một hàm gọi lại (callback) trong trường hợp cần mật khẩu để mở tệp tin PEM bị mã hóa. Giá trị NULL sẽ thông báo cho thư viện sử dụng hàm gọi lại (callback) mặc định và không cần mật khẩu để giải mã tệp tin PEM.

Tệp tin header PEM có hỗ trợ chuyển đổi ngược lại từ kiểu dữ liệu quản lý của OpenSSL sang dữ liệu trên ổ cứng. Ví dụ: lưu chứng chỉ X.509 dưới dạng tệp tin.

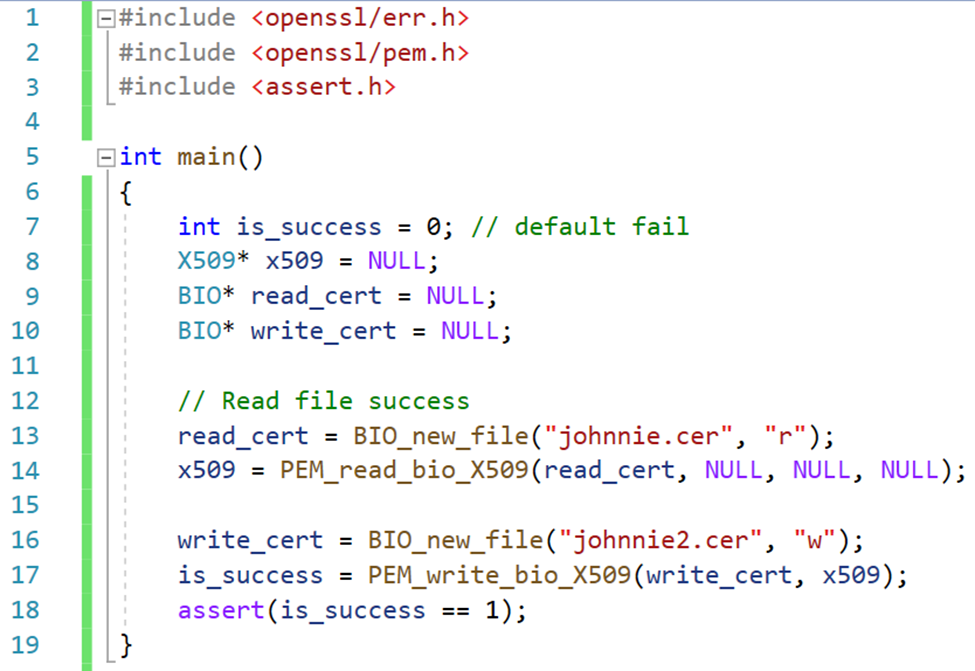


Figure 24. Ghi đối tượng quản lý X509 vào tệp tin.

Hàm **PEM\_write\_bio\_X509** sẽ trả về giá trị 1 nếu như ghi tệp tin thành công, ngược lại là giá trị 0 nếu thất bại. Hàm assert có chức năng gây crash chương trình, là một cách thông báo/debug cực kỳ rõ ràng dành cho lập trình viên.



#### Sinh khóa

Thư viện OpenSSL sử dụng 2 loại cấu trúc khóa cho mã hóa khóa công khai. Đầu tiên, nó sử dụng cấu trúc hàm để lưu trữ khóa cho thuật toán cụ thể nào đó, ví dụ RSA hoặc DSA. Sau đó, nó sử dụng cấu trúc EVP\_PKEY để lưu trữ khóa để sử dụng cho bất kỳ thuật toán nào.

Trong cả hai trường hợp, OpenSSL không tạo riêng cấu trúc để phân biệt giữa khóa bí mật và khóa công khai. Ở ví dụ trên sử dụng cấu trúc dữ liệu RSA để lưu trữ khóa, đặt tên gợi nhớ là khóa riêng tư để sử dụng cho các hàm sau đó. Ví dụ sau đây tạo khóa bí mật RSA có độ dài 1024 bit.

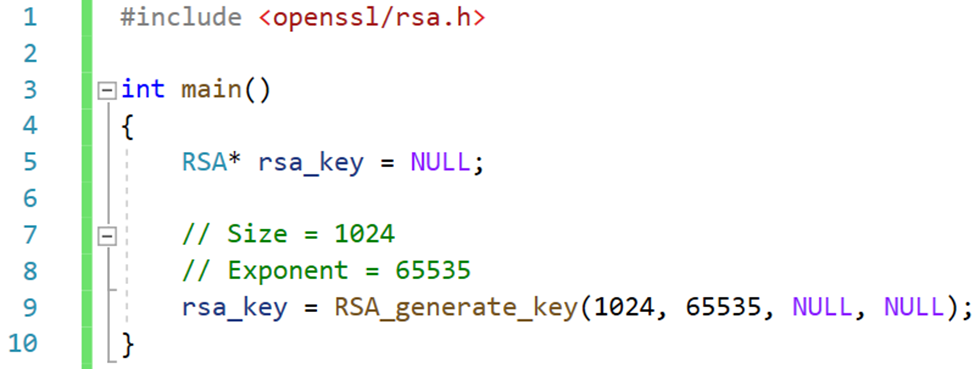


Figure 25. Tạo khóa RSA.

Ở trong ngữ cảnh high interface, thay vì sử dụng RSA như low interface, EVP cung cấp một cấu trúc riêng để lưu trữ khóa công khai tên là EVP\_PKEY sử dụng cho mã hóa khóa công khai. Ví dụ: tạo khóa thông qua low interface.



Figure 26. Tạo khóa theo cấu trúc EVP thông hàm gán.

Đoạn code trên đã phân tách riêng biệt khóa công khai trong cặp khóa sang cho biến evp\_key lưu trữ.



#### Mã hóa và giải mã

Mã hóa khóa công khai ở trường hợp này sử dụng low interface dễ dàng cho việc tiếp cận. Theo trình tự mã hóa và giải mã, hàm **RSA\_public\_encrypt** được sử dụng để mã bản tin, hàm **RSA\_private\_decrypt** để giải mã.



Figure 27. Mã hóa và giải mã khóa công khai.

Ở đối số thứ nhất và thứ hai, hàm **RSA\_public\_encrypt** yêu cầu cần được cung cấp chiều dài và thông tin bản rõ, đối số thứ ba là nơi sẽ chứa bản mã sau khi mã hóa bằng khóa công khai được lưu trong cấu trúc của biến rsa\_key, đối số cuối cùng cung cấp cho thuật toán RSA sử dụng thuật toán padding PKCS#1, thông tin trả về sẽ là chiều dài bản mã.

Nhờ vào thông tin chiều dài bản mã được mã hóa, hàm **RSA\_private\_decrypt** sử dụng khóa bí mật được lưu ở bên trong cấu trúc rsa\_key để giải mã và cho ra bản rõ.