**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---

# BÁO CÁO

## An toàn bảo mật hệ thống thông tin

## Đề tài: Tìm hiểu về giao thức bảo mật SSL/TLS

**Nhóm 8 - Lớp 01**

| Tên thành viên | Mã sinh viên |
|---|---|
| Trịnh Vinh Tuấn Đạt | B21DCCN031 |
| Trần Xuân Đạt | B21DCCN223 |
| Hoàng Anh Vũ | B21DCCN795 |
| Lò Văn Dương | B21DCCN283 |
| Nguyễn Hoàng Hiệp | B21DCCN343 |

**Giảng viên hướng dẫn:** Ninh Thị Thu Trang

**Hà Nội, 3/ 2024**

# MỤC LỤC

# DANH MỤC HÌNH ẢNH

## 1. TỔNG QUAN VỀ SSL/TLS

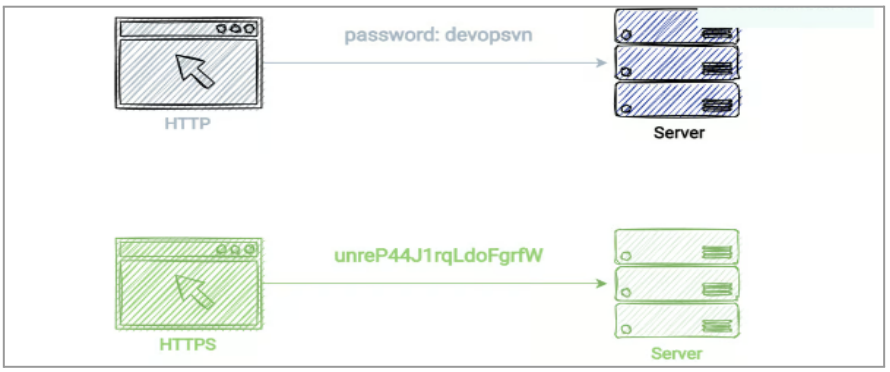
SSL (Secure Socket Layer) là một giao thức bảo mật Internet dựa trên mã hóa do công ty Netscape phát minh năm 1993. Giao thức này cung cấp quyền riêng tư, phương thức xác thực, và tính toàn vẹn dữ liệu cho các giao tiếp trên Internet. Các phiên bản SSL được phát triển bao gồm: phiên bản 1.0 phát hành năm 1993, phiên bản 2.0 phát hành năm 1995 và phiên bản 3.0 phát hành năm 1996. Giao thức SSL chính là tiền thân của một giao thức TLS.

TLS (Transport Layer Security) được phát triển vào năm 1999 dựa trên SSL 3.0. Sự khác biệt giữa bản 3.0 của SSL với phiên bản đầu tiên của TLS là không đáng kể; việc thay đổi tên được áp dụng để biểu thị sự thay đổi quyền chủ sở hữu. Các phiên bản của TLS gồm: phiên bản 1.0 phát hành năm 1999, phiên bản 1.1 phát hành năm 2005, phiên bản 1.2 phát hành năm 2008, phiên bản 1.3 được phát hành chính thức cho vào tháng 10 năm 2017. Hiện nay phiên bản TLS 1.3 được sử dụng rộng rãi nhất.

SSL/TLS sử dụng chứng chỉ (certificates) để thiết lập một liên kết mã hoá giữa một máy chủ (server) và một máy khách (client). Điều này cho phép những thông tin nhạy cảm như như thẻ tín dụng, thông tin cá nhân,.. được truyền đi một cách bí mật qua Internet, tránh lộ thông tin.

Chứng chỉ của giao thức sẽ chứa một khóa công khai (public key) để xác thực danh tính của trang web và cho phép trang giải mã dữ liệu được chuyển đến thông qua mật mã bất đối xứng hoặc mật mã công khai. Khóa bí mật (private key) phù hợp sẽ được giữ bí mật trong máy chủ.

Một ví dụ phổ biến nhất về việc sử dụng SSL đó là: nếu ta truy cập một trang web mà ta thấy giao thức của nó là HTTP, thì giao tiếp giữa trình duyệt và máy chủ đang không được bảo mật, dữ liệu đang được truyền đi theo dạng văn bản thô, có thể bắt được gói tin và xem trực tiếp nội dung bên trong mà không cần bất kỳ một thủ thuật giải mã nào. Còn nếu ta truy cập một trang web mà thấy giao thức của nó là HTTPS, có nghĩa là nó đang có sử dụng SSL và dữ liệu truyền đi giữa trình duyệt và máy chủ đã được mã hóa.




*Hình 1. Mã hóa gói tin bằng giao thức SSL*

SSL/TLS thường được coi là giao thức bảo mật quan trọng nhất ở tầng ứng dụng, và nó đóng vai trò quan trọng trong việc bảo mật dữ liệu truyền qua mạng Internet.

## 2. CẤU TRÚC VÀ CÁCH HOẠT ĐỘNG CỦA SSL/TLS

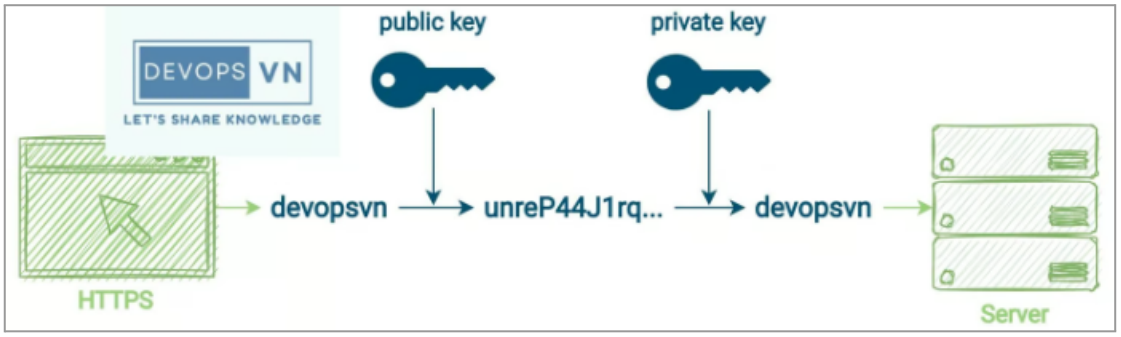
Trước khi tìm hiểu sâu hơn về SSL, ta cần hiểu trước về hai khái niệm sau: Mã hóa đối xứng (Symmetric Cryptography) và Mã hóa bất đối xứng (Asymmetric Cryptography)

- Mã hóa bất đối xứng (Asymmetric Cryptography):

  + Mã hóa bất đối xứng hay còn gọi là mã hóa khóa công khai là cách mã hóa dữ liệu sử dụng một cặp khóa là khóa công khai (public key) và khóa riêng (private key).

  + Khóa công khai sẽ được chia sẻ ra bên ngoài cho ai muốn giao tiếp, còn khóa riêng được giữ ở máy chủ và không được chia sẻ.

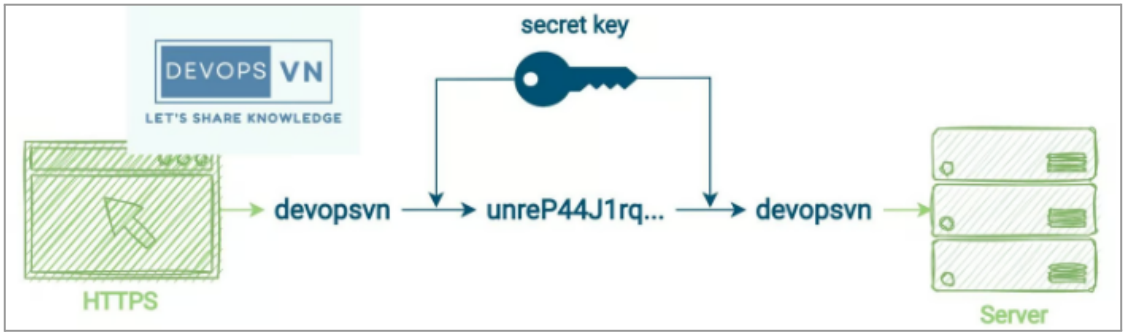
  + Khi giao tiếp thì bên gửi sẽ dùng khóa công khai để mã hóa dữ liệu, và khi dữ liệu được gửi tới thì bên nhận sẽ dùng khóa riêng để giải mã dữ liệu.




*Hình 2. Minh họa mã hóa bất đối xứng*

- Mã hóa đối xứng (Symmetric Cryptography):

  + Mã hóa đối xứng cũng giống với bất đối xứng, chỉ khác ở điểm là thay vì sử dụng một cặp khóa thì nó chỉ sử dụng duy nhất một khóa cho cả việc mã hóa và giải mã dữ liệu.

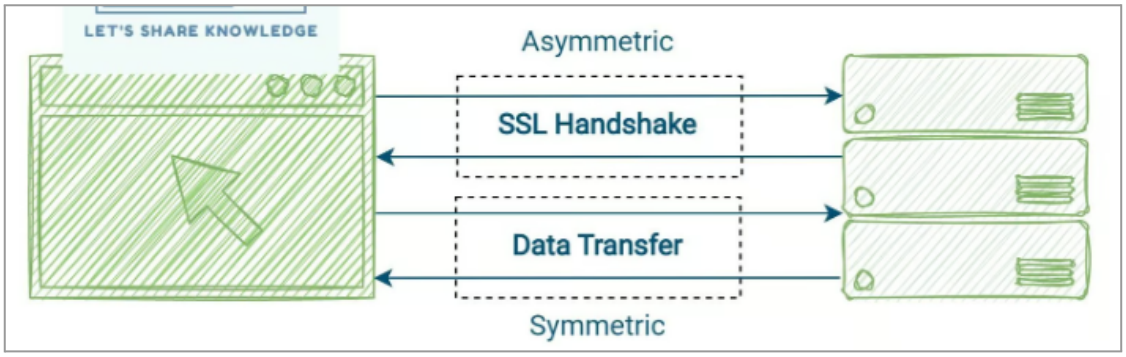



*Hình 3. Minh họa mã hóa đối xứng*

## 2.1. Cách hoạt động cơ bản của SSL/TLS

Hoạt động của SSL bao gồm hai phần chính: quá trình thiết lập kết nối bảo mật (SSL Handshake) và quá trình truyền dữ liệu an toàn (SSL Record Protocol).

Mã hóa bất đối xứng được sử dụng trong bước SSL handshake. Mã hóa đối xứng được sử dụng cho việc truyền dữ liệu sau bước SSL handshake.




*Hình 4. Minh họa hai hoạt động chính của giao thức SSL*

### *2.1.1. Quá trình thiết lập kết nối bảo mật (SSL Handshake):*

Quá trình này xảy ra khi hai bên máy chủ và máy khách muốn thiết lập một kết nối bảo mật. SSL Handshake bao gồm các bước sau:

- Xác định thuật toán và phiên bản SSL được sử dụng.
- Xác thực danh tính của máy chủ (và có thể cả máy khách) sử dụng chứng chỉ số.
- Sử dụng mã hoá bất đối xứng để thỏa thuận về khóa đối xứng và các thông số mã hoá khác.
- Gửi thông báo hoàn thành để bắt đầu quá trình truyền dữ liệu an toàn.

### *2.1.2. Quá trình truyền dữ liệu an toàn (SSL Record Protocol):*

Sau khi kết nối bảo mật được thiết lập, dữ liệu được truyền qua mạng thông qua SSL Record Protocol.

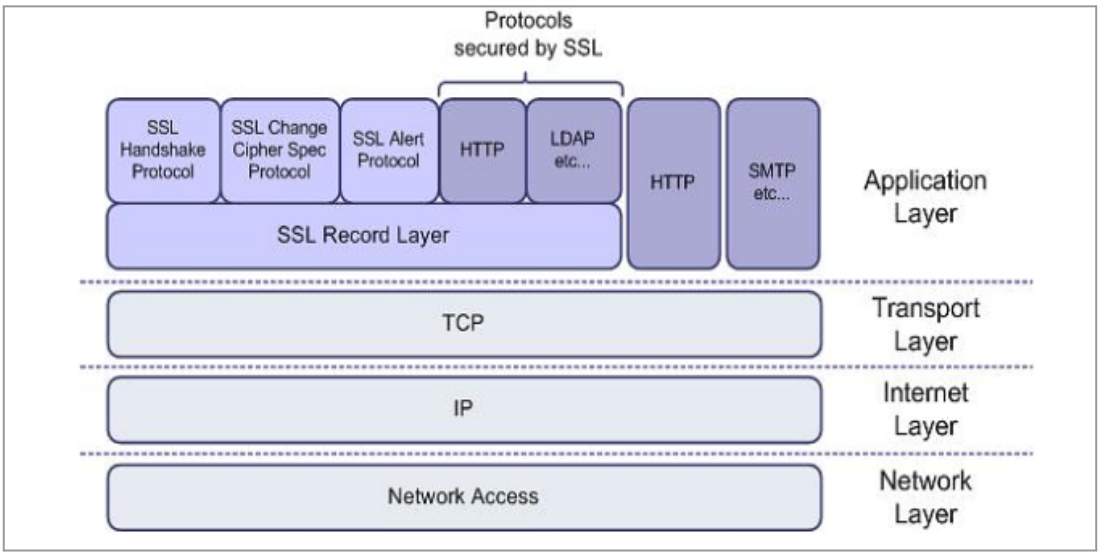
Dữ liệu được chia thành các gói tin và mã hóa bằng khóa đối xứng đã được thỏa thuận.

SSL Record Protocol cũng thêm các thông tin điều khiển như mã hoá, kiểm tra tính toàn vẹn và mã độc lập với dữ liệu trước khi gửi.

### *2.2. Cấu trúc và cách hoạt động chi tiết của SSL/TLS*

Các giao thức con SSL được sắp xếp lớp trên SSL Record Protocol để cung cấp sự hỗ trợ cho việc quản lý session SSL và thiết lập nối kết. Như vậy, cấu trúc SSL tổng cộng gồm 4 giao thức con sau:

- SSL Handshake Protocol: Giao thức bắt tay của SSL có nhiệm vụ trao đổi các thông điệp xác thực thực thể và thiết lập các thông số cho phiên làm việc;
- SSL Change Cipher Spec Protocol: Giao thức thiết lập việc sử dụng các bộ mã hóa được hỗ trợ bởi cả 2 bên tham gia phiên truyền thông;
- SSL Alert Protocol: Giao thức cảnh báo của SSL;
- SSL Record Protocol: Giao thức bản ghi của SSL có nhiệm vụ tạo đường hầm an toàn để chuyển thông tin đảm bảo tính bí mật, toàn vẹn và xác thực.




*Hình 5. Các giao thức con của SSL/TLS*

#### *2.2.1. Giao thức bắt tay (handshake protocol)*

SSL Handshake Protocol là một phần quan trọng của SSL, nó cung cấp ba dịch vụ cho các kết nối SSL giữa client và server. Handshake Protocol cho phép client/server thống nhất về phiên bản giao thức, xác thực mỗi bên bằng cách thi hành một MAC và thỏa thuận về một thuật toán mã hoá và các khóa lập mã cho việc bảo vệ các dữ liệu gửi đi trong một SSL record trước khi giao thức ứng dụng truyền đi hay nhận được byte dữ liệu đầu tiên.

Giao thức bắt tay (Handshake Protocol) bao gồm một dãy các thông điệp được trao đổi bởi client và server. Hình 6 minh hoạ sự trao đổi các thông điệp handshake cần thiết để thiết lập một kết nối logic giữa client và server. Nội dung và ý nghĩa mỗi thông điệp được mô tả chi tiết trong các phần sau:

1. SSL Client gửi thông điệp “client hello” và thông tin mã hóa (Cryptographic information) đến SSL Server;

2. SSL Server gửi thông điệp “server hello”, các bộ mã hóa hỗ trợ (CipherSuite) và chứng chỉ máy chủ (Server certificate) đến SSL Client. SSL Server cũng có thể gửi yêu cầu máy khách cung cấp chứng chỉ máy khách (Client certificate) nếu cần thiết;

3. Nhận được yêu cầu, SSL Client kiểm tra chứng chỉ máy chủ và kiểm tra các tham số mã hóa. Hai bên thống nhất sử dụng các bộ mã hóa tốt nhất cùng hỗ trợ cho phiên làm việc. Nếu chứng chỉ máy chủ không hợp lệ quá trình quá trình khởi tạo phiên kết thúc không thành công. Nếu chứng chỉ máy chủ hợp lệ tiếp tục bước tiếp theo;

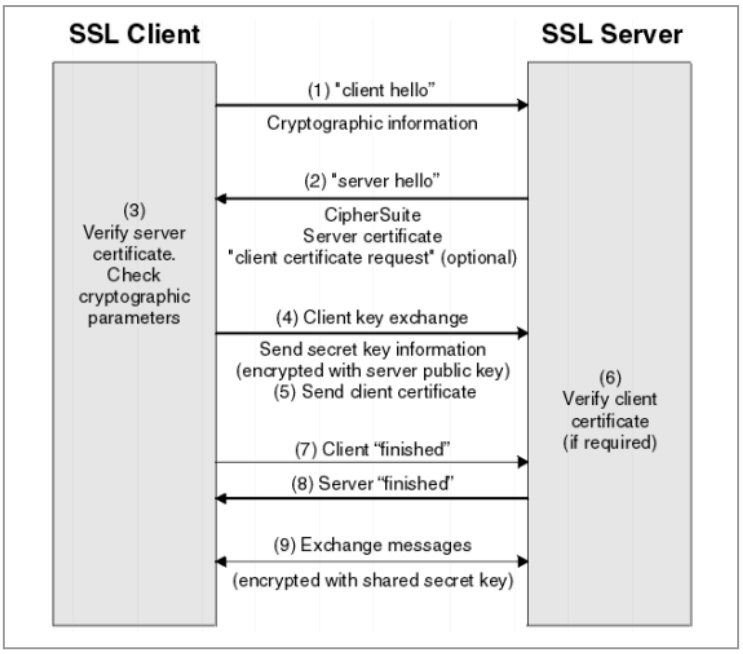
4. Trao đổi khóa máy khách (Client key exchange). SSL Client sinh khóa phiên (session key), sau đó mã hóa khóa phiên sử dụng khóa công khai của SSL Server lấy từ chứng chỉ máy chủ và gửi cho SSL Server;

5. SSL Client cũng có thể gửi chứng chỉ máy khách cho máy chủ nếu được yêu cầu;

6. SSL Server sử dụng khóa riêng của mình để giải mã khôi phục khóa phiên gửi từ SSL Client. SSL Server cũng có thể kiểm tra chứng chỉ máy khách nếu cần thiết;

7. Client gửi thông điệp kết thúc khởi tạo phiên “Finished”;

8. Server gửi thông điệp kết thúc khởi tạo phiên “Finished”.




*Hình 6. Khởi tạo phiên làm việc với SSL/TLS*

Khi client và server quyết định sử dụng lại phiên trước hoặc tạo một bản sao của phiên đang tồn tại (thay vì phải thỏa thuận các tham số an toàn mới), thì luồng thông điệp hoạt động như sau: client gửi một ClientHello có sử dụng Session ID của phiên

được dùng lại server kiểm tra nơi lưu trữ phiên (session cache) tương ứng. Nếu có, server sẽ thiết lập lại kết nối dưới trạng thái phiên được chỉ định, server sẽ gửi một thông điệp ServerHello có giá trị Session ID giống hệt. Mỗi khi quá trình thiết lập lại được hoàn thành thì client và server có thể trao đổi dữ liệu lớp ứng dụng. Ngược lại nếu Session ID không tìm thấy thì server sẽ tạo ra một Session ID mới, SSL client và server thực hiện quá trình bắt tay thông thường như ở trên.

## A. Giai đoạn 1: Các thông điệp Hello cho kết nối logic:

Client gửi một thông điệp ClientHello tới server, server phải đáp ứng lại bằng một thông điệp ServerHello, hoặc ngược lại nếu một lỗi xảy ra thì kết nối sẽ thất bại. ClientHello và ServerHello được sử dụng để tăng tính an toàn giữa client và server. ClientHello và ServerHello thiết lập các thuộc tính: Protocol Version, Session ID, Cipher Suite và Compression Method. Ngoài ra hai giá trị ngẫu nhiên được tạo ra và được trao đổi là: ClientHello.random và ServerHello.random. Các thông điệp giai đoạn chào hỏi được sử dụng để trao đổi các thuộc tính thừa kế bảo mật giữa client và server.

### a. Thông điệp HelloRequest

Thông điệp này được gửi bởi server tại bất kỳ thời điểm nào, nhưng có thể bị client bỏ qua nếu Handshake Protocol đã đang thực hiện. Nếu client nhận một HelloRequest trong một trạng thái thỏa thuận bắt tay thì client đơn giản là bỏ qua thông điệp này.

### b. Thông điệp ClientHello

Thông điệp ClientHello bắt đầu phiên truyền thông SSL giữa hai bên. Client sử dụng thông điệp này hỏi server để bắt đầu thỏa thuận các dịch vụ bảo mật được sử dụng SSL. Sau đây là các thành phần quan trọng của một thông điệp ClientHello:

➢ Version: Phiên bản SSL cao nhất mà client hỗ trợ. Phiên bản SSL hiện tại là 3.0. Chú ý rằng một server có thể thừa nhận rằng client có thể hỗ trợ tất các phiên bản SSL cao hơn và bao gồm giá trị của trường này. Ví dụ, nếu một client gửi một ClientHello với Version có giá trị 3 tới server chỉ hỗ trợ SSL version 2.0, server có thể đáp ứng bằng các thông điệp version 2.0 và nó hy vọng client hiểu. Client có thể quyết định tiếp tục với phiên SSL sử dụng version 2.0, hoặc có thể bỏ qua.

➢ Random: Một cấu trúc ngẫu nhiên được sinh ra, bao gồm một nhãn thời gian 32-bit và 28 byte được sinh bởi bộ sinh số ngẫu nhiên bảo mật. Nhãn thời gian-32 bit bao gồm thời gian và ngày, để chắc chắn rằng client không bao giờ sử dụng cùng một giá trị ngẫu nhiên hai lần. Sử dụng phương pháp này để chống lại một sự sao chép bất hợp pháp các thông điệp SSL từ một client bất hợp pháp và sử dụng lại chúng để thiết lập một phiên giả mạo. 28 byte còn lại là một số ngẫu nhiên "bảo mật mật mã", sử dụng

một công nghệ được biết đến như là "sinh số giả ngẫu nhiên" để tạo ra các số ngẫu nhiên.

➢ Session ID: Một định danh phiên chiều dài biến. Một giá trị khác không thể hiện rằng client muốn cập nhật các tham số của kết nối đang tồn tại hoặc tạo một kết nối mới trong phiên này. Một giá trị bằng không thể hiện rằng client muốn thiết lập một kết nối mới trên một phiên làm việc mới.

➢ CipherSuite: Đây là một danh sách bao gồm các sự kết hợp của các thuật toán lập mã được hỗ trợ bởi client, xếp tăng theo thứ tự tham chiếu. Mỗi phần tử của danh sách định nghĩa cả hai thuật toán trao đổi khóa và một CipherSpec.

➢ Compression Method: danh sách các phương pháp nén client hỗ trợ.

**c. ServerHello**

Khi server nhận được thông điệp ClientHello, nó gửi đáp ứng lại với một thông điệp ServerHello. Nội dung của ServerHello tương tự như ClientHello, tuy nhiên, cũng có một số điểm khác biệt quan trọng.

➢ Version: Lưu giá trị phiên bản thấp nhất được chấp nhận bởi client trong ClientHello và phiên bản cao nhất được hỗ trợ bởi server. Version trong ServerHello quyết định phiên bản SSL mà kết nối sẽ sử dụng.

➢ Random: Giá trị ngẫu nhiên được sinh ra bởi server, bốn byte đầu là nhãn thời gian (để tránh các giá trị ngẫu nhiên lặp lại); các byte còn lại sẽ được tạo bởi bộ sinh số ngẫu nhiên bảo mật lập mã.

➢ Session ID: Trường Session ID của ServerHello có thể lưu một giá trị, không giống như trường Session ID trong ClientHello đã được nói tới. Giá trị này định danh duy nhất truyền thông SSL cụ thể hoặc session. Lý do chính cho việc định danh một phiên SSL cụ thể là để sau này có thể dùng lại. Nếu server không chỉ định ra phiên đã từng sử dụng, nó có thể bỏ qua trường Session ID trong thông điệp ServerHello.

➢ CipherSuite : Quyết định các tham số lập mã, các thuật toán và các cỡ khoá, được sử dụng trong một phiên. Server phải chọn một bộ mã đơn từ danh sách mà client đưa cho trong thông điệp ClientHello.

➢ CompressionMethod: Server sử dụng trường này để định danh việc nén dữ liệu sử dụng cho phiên. Một lần nữa, server phải lấy chúng từ danh sách đã được liệt kê trong ClientHello. Tuy nhiên, các phiên bản SSL hiện tại không định nghĩa bất kỳ phương pháp nén nào, vì thế trường này không có ý nghĩa thiết thực cụ thể.

**B. Giai đoạn 2: Xác thực Server và trao đổi khoá:**

Sau các thông điệp hello, server bắt đầu giai đoạn này bằng cách gửi chứng chỉ của nó nếu nó cần được xác thực. Thêm vào đó, một thông điệp ServerKey Exchange có thể

được gửi nếu nó được yêu cầu. Nếu server đã được xác thực, nó có thể yêu cầu một chứng chỉ từ client, nếu nó thích hợp với bộ mã được chọn. Sau đó, server sẽ gửi thông điệp ServerHelloDone, thể hiện rằng giai đoạn thông điệp hello của handshake đã hoàn thành. Server sau đó sẽ đợi đáp ứng của client. Nếu server đã gửi một thông điệp yêu cầu chứng chỉ  Certificate request message, client phải gửi lại thông điệp chứng chỉ (certificate message).

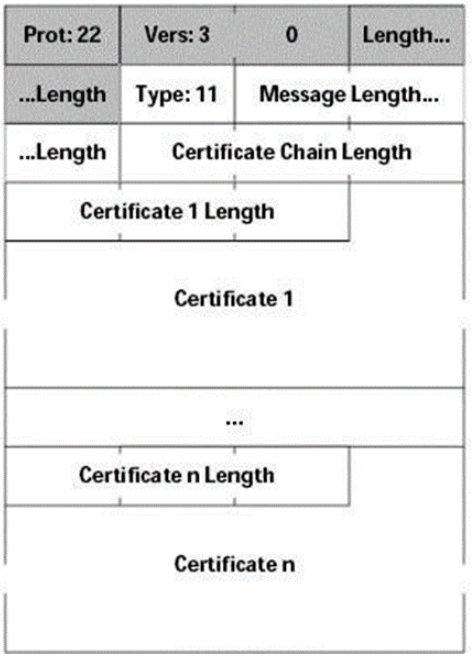
**a. Thông điệp Server Certificate**

Nếu server đã được xác thực, nó phải gửi một chứng chỉ ngay sau thông điệp ServerHello. Kiểu chứng chỉ phải phù hợp với thuật toán chuyển đổi khoá của bộ mã (cipher suite) đã lựa chọn, và nó thường là một chứng chỉ X.509 v3. Nó phải chứa một khoá phù hợp với phương pháp chuyển đổi khoá (key exchange method). Thuật toán ký cho chứng chỉ phải giống như thuật toán cho khóa chứng chỉ (certificate key).

Kiểu thông điệp giao thức Handshake của nó là 11, và nó bắt đầu với kiểu thông điệp và chiều dài thông điệp bắt tay chuẩn. Phần thân của thông điệp bao gồm một dãy các chứng chỉ khóa công khai. Dãy này bắt đầu với 3 byte thể hiện chiều dài. (Giá trị chiều dài dãy luôn là 3 nhỏ hơn giá trị chiều dài thông điệp). Mỗi chứng chỉ trong dãy cũng bắt đầu với một trường 3 byte lưu cỡ của chứng chỉ. Thông điệp đầu tiên thể hiện toàn bộ chiều dài của dãy chứng chỉ. Phần sau đó thể hiện chiều dài của mỗi chứng chỉ với 3 byte ngay trước chứng chỉ đó.

Dãy các thông điệp cho phép SSL hỗ trợ các hệ chứng chỉ. Chứng chỉ đầu tiên trong dãy luôn là của bên gửi. Chứng chỉ tiếp theo là của bên cấp chứng chỉ cho chứng chỉ của bên gửi. Chứng chỉ thứ ba thuộc về CA cho quyền đó, và cứ tiếp tục như thế.

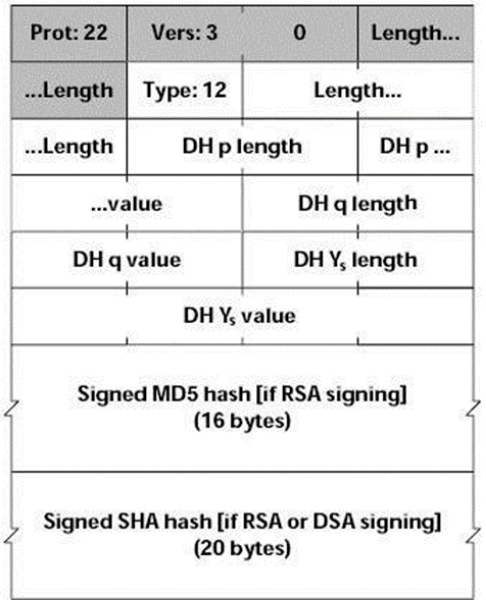
Dãy chứng chỉ sẽ tiếp tục cho tới khi nó tìm thấy một chứng chỉ cho quyền chứng chỉ gốc (root).




*Hình 7. Thông điệp certificate*

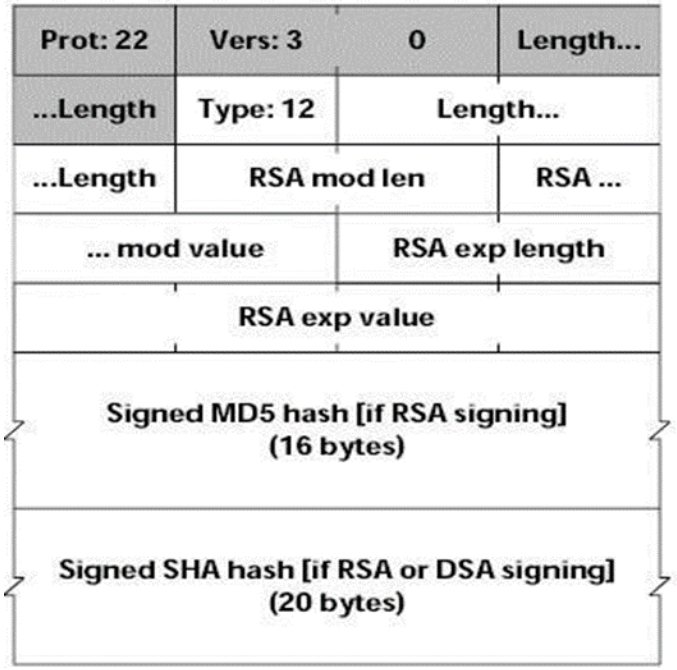
**b. Thông điệp Server Key Exchange**

Thông điệp Server Key Exchange truyền thông tin khóa từ server tới client. Định dạng chính xác của nó tùy thuộc vào các thuật toán lập mã được sử dụng để trao đổi thông tin khoá. Các định dạng khác nhau phù hợp với các giao thức khóa khác nhau. Trong mọi trường hợp, kiểu thông điệp handshake đều có giá trị 12. Các client phải sử dụng kiến thức đã nắm được từ các thông điệp handshake trước (thuật toán trao đổi khoá từ bộ mã đã chọn trong thông điệp ServerHello và thuật toán ký, nếu liên quan từ thông điệp Certificate) để hiểu một thông điệp Server Key Exchange một cách chính xác.

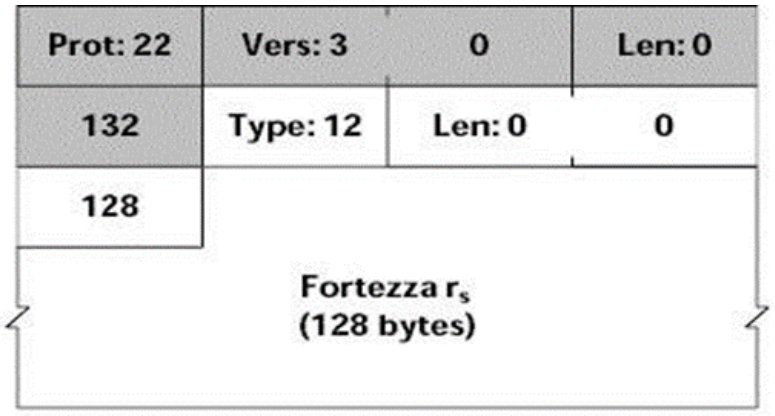



*Hình 8. Server Key Exchange mang các tham số Diffie-Hellman*

Các thông điệp trao đổi khóa RSA được yêu cầu trong trường hợp server sử dụng RSA nhưng có một khóa RSA chỉ để ký. Client không thể gửi một khoá bí mật đã được mã hoá với khoá công khai của server. Thay vào đó, server phải tạo ra một cặp khóa public/private RSA tạm và sử dụng thông điệp Server Key Exchange để gửi khoá công khai. Nội dung thông điệp bao gồm hai tham số của khoá công khai RSA tạm (môđun và số mũ) cộng với một ký số của các tham số đó. Mỗi tham số trong đó được lưu trong thông điệp theo định dạng: chiều dài, theo sau là giá trị.




*Hình 9. Server Key Exchange mang các tham số RSA*



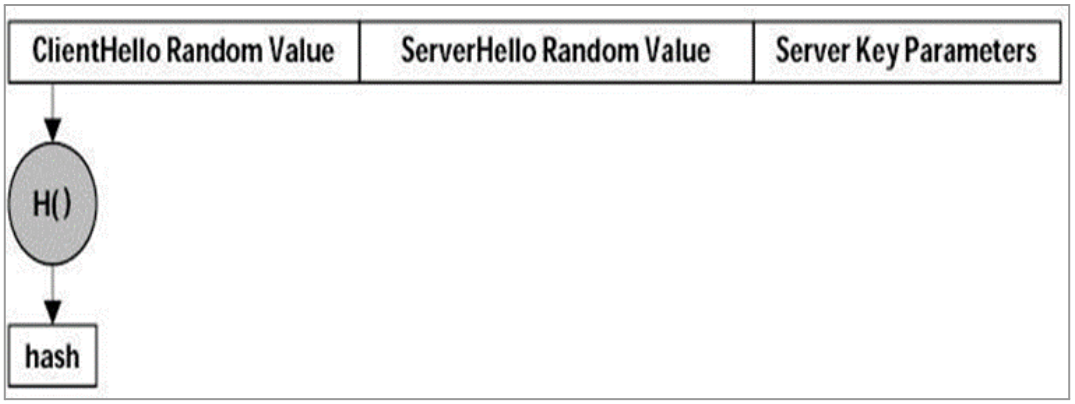

*Hình 10. Server Key Exchange sử dụng Fortezza*

Khi các hệ thống thi hành trao đổi khóa Fortezza/DMS, thông điệp ServerKeyExchange mang giá trị Fortezza rs, cỡ 128 byte.

ServerKeyExchange có thể bao gồm các tham số đã được ký. Định dạng chính xác của các tham số đó phụ thuộc vào thuật toán ký số mà server hỗ trợ. Nếu xác thực

server không phải là một phần của một phiên SSL cụ thể thì không cần ký và thông điệp ServerKeyExchange kết thúc với các tham số Diffie-Hellman, RSA hay Fortezza.

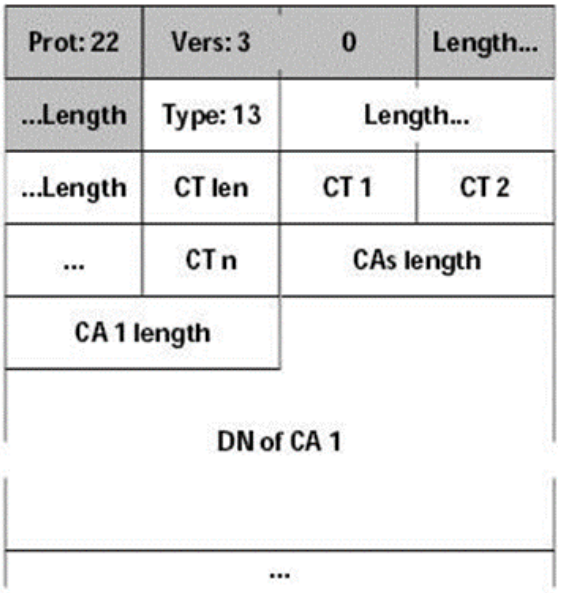
Tuy nhiên, nếu server không ẩn danh và gửi đi một thông điệp Certificate, thì định dạng các tham số được ký tùy thuộc vào thuật toán ký số được thể hiện trong chứng chỉ của server. Nếu chứng chỉ của server dùng ký RSA, thì các tham số được ký bao gồm dãy hai hàm băm: một hàm băm MD5 và một hàm băm SHA. Chú ý rằng một ký số đơn cho các hàm băm kết hợp được gộp vào, không phải là các ký số riêng biệt cho từng hàm. Nếu chứng chỉ của server là ký DSA, thì các tham số được ký chỉ bao gồm một hàm băm SHA. Trong từng trường hợp, đầu vào của các hàm băm là dữ liệu bao gồm giá trị ngẫu nhiên của client (từ ClientHello), tiếp theo là giá trị ngẫu nhiên của server (trong ServerHello), tiếp theo nữa là các tham số trao đổi khóa (các tham số Diffie-Hellman hoặc các tham số RSA). Không có tham số nào được thêm vào cho trao đổi khoá Fortezza/DMS.




*Hình 11. Server ký một hàm băm của các tham số ServerKeyExchange*

**c. Thông điệp CertificateRequest:**

Để xác thực định danh của client, đầu tiên server gửi một thông điệp CertificateRequest. Thông điệp này không chỉ yêu cầu client gửi các chứng chỉ của nó (và các thông tin ký sử dụng khoá riêng cho chứng chỉ đó), nó cũng đưa ra cho client các chứng chỉ mà server có thể chấp nhận.




*Hình 12. Thông điệp CertificateRequest*

Thông điệp CertificateRequest là kiểu thông điệp handshake 13, sau kiểu handshake và chiều dài thông điệp bao gồm một danh sách các kiểu chứng chỉ có thể chấp nhận được. Danh sách kiểu này bắt đầu với chiều dài của từng cái (một giá trị một byte), sau đó là một hay nhiều giá trị byte đơn định danh kiểu chứng chỉ cụ thể. Bảng dưới đây liệt kê các giá trị kiểu chứng chỉ và ý nghĩa của chúng.

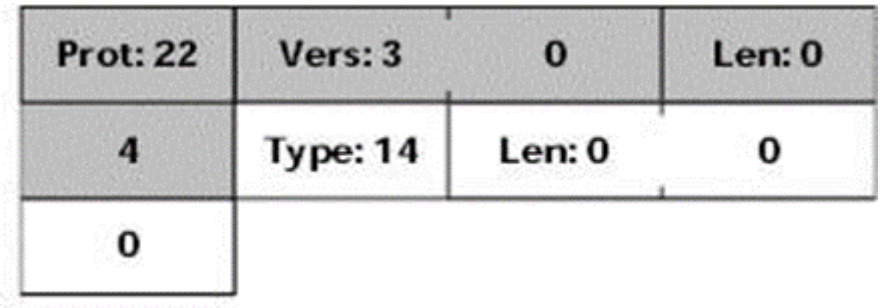
| Giá trị | Kiểu chứng chỉ |
|---|---|
| 1 | Ký số và trao đổi khóa RSA |
| 2 | Ký số DSA |
| 3 | Ký số RSA với trao đổi khoá fixed Diffie-Hellman |
| 4 | Ký số DSA với trao đổi khoá fixed Diffie-Hellman |
| 5 | Ký số RSA với trao đổi khoá ephemeral Diffie-Hellman |
| 6 | Ký số DSA với trao đổi khoá ephemeral Diffie-Hellman |
| 20 | Ký số và trao đổi khoá Fortezza/DMS |

Bên cạnh các kiểu chứng chỉ, thông điệp CertificateRequest cũng thể hiện các quyền chứng chỉ mà server cho là thích hợp. Danh sách này bắt đầu với trường chiều dài 2 byte và sau đó bao gồm một hay nhiều cái tên khác nhau. Mỗi một tên có trường chiều dài riêng, và các định danh một quyền chứng chỉ.

**d. Thông điệp ServerHelloDone**

Thông điệp này được gửi bởi server báo hiệu kết thúc giai đoạn ServerHello và các thông điệp kết hợp. Sau khi gửi thông điệp này, server sẽ chờ đáp ứng của client. Thông điệp này có nghĩa là server đã hoàn thành việc gửi các thông điệp hỗ trợ trao đổi khoá, và client có thể bắt đầu giai đoạn trao đổi khoá của nó. Trước khi nhận được thông điệp

ServerHelloDone, client sẽ kiểm tra rằng server đã cung cấp một chứng chỉ hợp lệ nếu được yêu cầu và kiểm tra các tham số server hello có thể chấp nhận được. Nếu tất cả đã thoả mãn, client sẽ gửi một hay nhiều thông điệp trở lại cho server. Định dạng thông điệp ServerHelloDone rất đơn giản, kiểu thông điệp Handshake là 14 và chiều dài thông điệp là 0 (vì nó không mang bất kỳ thông tin nào cả).




*Hình 13. Thông điệp ServerHelloDone*

**C. Giai đoạn 3: Xác thực Client và trao đổi khoá**

Nếu server đã gửi một thông điệp CertificateRequest, client phải gửi thông điệp chứng chỉ của mình, sau đó là thông điệp ClientKeyExchange. Nội dung của thông điệp đó tuỳ thuộc vào thuật toán khóa công khai được lựa chọn trong thoả thuận ClientHello và ServerHello. Nếu client đã gửi một chứng chỉ với các thuộc tính ký, một thông điệp CertificateVerify được gửi để xác thực lại chứng chỉ.

**a. Thông điệp ClientCertificate**

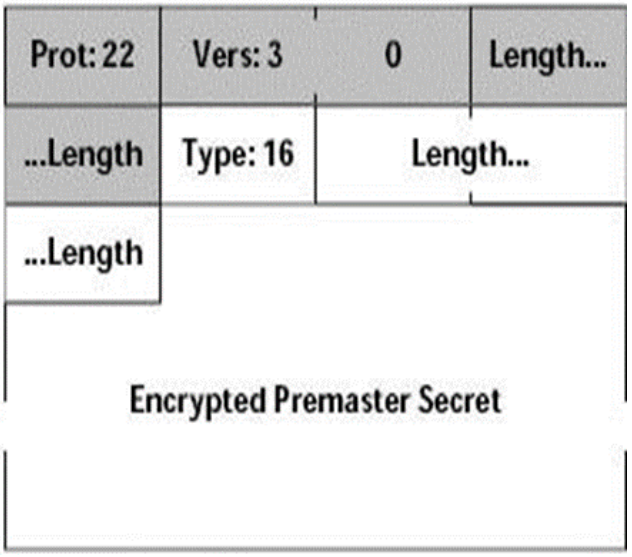
Đây là thông điệp đầu tiên client có thể gửi cho sau khi nhận được thông điệp ServerHelloDone. Thông điệp này được gửi chỉ khi server yêu cầu. Nếu không có chứng chỉ phù hợp, client sẽ gửi một thông điệp không chứa chứng chỉ nào. Nếu xác thực client được yêu cầu bởi server để quá trình bắt tay được tiếp tục, nó có thể đáp ứng lại với một cảnh báo lỗi bắt tay. Loại thông điệp tương tự và cấu trúc sẽ được sử dụng cho các đáp ứng của client tới một thông điệp ClientRequest. Chú ý rằng, một client có thể không gửi chứng chỉ nếu nó không có chứng chỉ phù hợp để gửi đáp ứng tới yêu cầu xác thực của server. Các chứng chỉ Diffie-Hellman của client phải phù hợp với các tham số Diffie-Hellman cụ thể của server.

**b. Thông điệp ClientKeyExchange**

Thông điệp này luôn được gửi bởi client, theo ngay sau thông điệp ClientCertificate (nếu nó được gửi). Hay nói cách khác, nó là thông điệp đầu tiên client gửi đi sau khi nhận được thông điệp ServerHelloDone. Với thông điệp này, client cung cấp cho server các nguyên liệu khoá cần thiết cho bảo mật truyền thông; định dạng chính xác của thông điệp tuỳ thuộc vào thuật toán trao đổi khóa cụ thể mà các bên sử

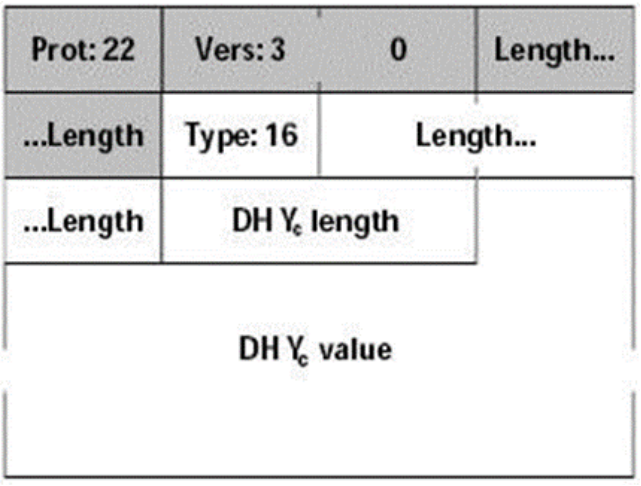
dụng. Ba khả năng có thể mà SSL cho phép là trao đổi khóa RSA, Diffie-Hellman, và Fortezza/DMS.

Định dạng thông điệp đầu tiên là trao đổi khóa RSA. Thông điệp có loại thông điệp handshake là 16, và chiều dài thông điệp handshake chuẩn. Phần thân thông điệp bao gồm duy nhất **premaster secret** đã mã hoá. **Premaster secret** được mã hoá sử dụng khoá công khai của server, đã nhận được trong ServerKeyExchange hay thông điệp Certificate.




*Hình 14. Thông điệp ClientKeyExchange với RSA*

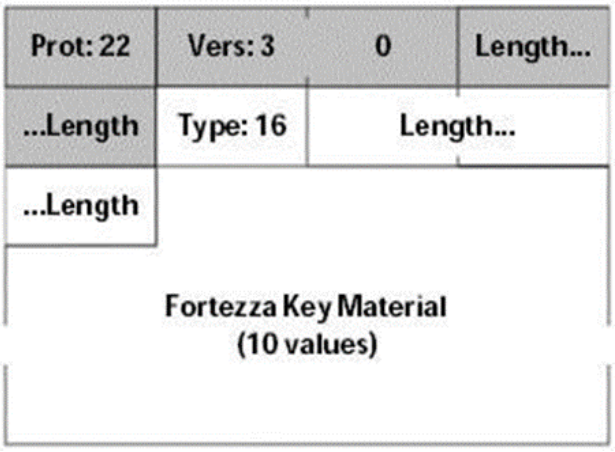
Với trao đổi khóa RSA, **premaster secret** đơn giản là hai byte lưu phiên bản SSL mà client hỗ trợ (3 và 0 cho phiên bản 3.0) tiếp theo là 46 byte ngẫu nhiên được sinh ra an toàn.




*Hình 15. Thông điệp ClientKeyExchange với Diffie-Hellman*

Với giao thức trao đổi khoá là Diffie-Hellman, có hai trạng thái có thể của thông điệp ClientKeyExchange. Phần thân thông điệp bao gồm giá trị Yc của client, xếp theo

chiều dài giá trị. Nếu trao đổi Diffie-Hellman là chi tiết, giá trị Yc được lưu trong chứng chỉ của client, khi đó ClientKeyExchange sẽ trống.

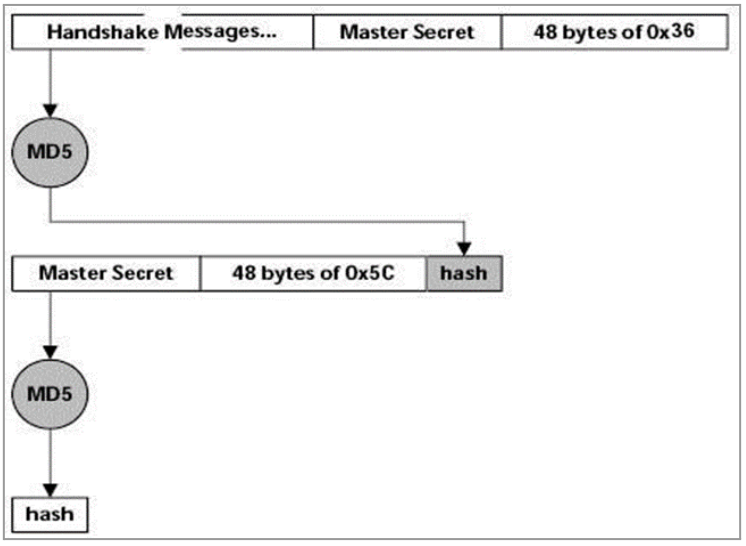



*Hình 16. Thông điệp ClientKeyExchange với Fortezza*

Với trao đổi khóa Fortezza/DMS, thông điệp ClientKeyExchange yêu cầu một tập hợp các tham số (10 giá trị).

**c. Thông điệp CertificateVerify**

Thông điệp này được sử dụng để cung cấp một sự xác thực rõ ràng hơn về chứng chỉ của client. Thông điệp chỉ được gửi khi bất kỳ một chứng chỉ client nào có khả năng ký số. Khi được gửi, nó sẽ theo ngay sau thông điệp ClientKeyExchange. Thông điệp này ký trên một hàm băm mã dựa trên các thông điệp, và cấu trúc của nó được định nghĩa như sau:




*Hình 17. Tạo thông điệp CertificateVerify*

Định dạng chính xác của thông tin tùy thuộc vào việc chứng chỉ của client dùng ký RSA hay DSA. Với các chứng chỉ RSA, hai hàm băm riêng biệt được kết hợp lại và ký:

một hàm băm MD5 và một hàm băm SHA. Với các chứng chỉ DSA, chỉ một hàm băm SHA được tạo và được ký.

Trong mọi trường hợp, thông tin mà server đưa vào các hàm băm là như nhau. Client xây dựng thông tin trong ba bước. Đầu tiên, họ tính một giá trị đặc biệt gọi là **master secret**. Để tính giá trị của **master secret**, client tính toán theo tiến trình sau:

1. Bắt đầu với 48 byte **premaster secret**. Client tạo ra giá trị này và gửi nó tới server trong thông điệp ClientKeyExchange.
2. Tính toán hàm băm SHA của ký tự ASCII 'A' theo sau **premaster secret**, giá trị ngẫu nhiên của client (từ ClientHello) và giá trị ngẫu nhiên của server (từ ServerHello).
3. Tính hàm băm MD5 của **premaster secret** và đầu ra của bước 2.
4. Tính hàm băm SHA của hai ký tự ASCII 'BB', **premaster secret**, giá trị ngẫu nhiên của client, giá trị ngẫu nhiên của server.
5. Tính hàm băm MD5 của premaster secret và đầu ra của bước 4.
6. Kết hợp kết quả bước 3 và bước 5.
7. Tính hàm băm SHA của ba ký tự ASC 'CCC', premaster secret, giá trị ngẫu nhiên của client, giá trị ngẫu nhiên của server.
8. Tính hàm băm MD5 của premaster secret và đầu ra của bước 7.
9. Kết hợp kết quả bước 8 và bước 6

Một khi client đã có giá trị **master secret**, nó chuyển tới giai đoạn tiếp theo là tạo một hàm băm nội dung đầy đủ của các thông điệp handshake SSL trước đó đã được trao đổi trong suốt một phiên, theo sau là **master secret**, và tiếp theo nữa là giá trị byte đơn 001100110, được lặp lại 48 lần cho MD5 và 40 lần cho SHA. Trong bước thứ ba, client tạo một hàm băm mới sử dụng **master secret** tương tự, theo sau là giá trị nhị phân 01011100, được lặp lại 48 lần cho MD5, và 40 lần cho SHA, cuối cùng là đầu ra của hàm băm.

```
master secret = MD5(premaster secret + SHA('A' + premaster secret +
                        ClientHello.random + ServerHello.random))
                +
                MD5(premaster secret + SHA('BB' + premaster secret +
                        ClientHello.random + ServerHello.random))
                +
                MD5(premaster secret + SHA('CCC' + premaster secret +
                        ClientHello.random + ServerHello.random))
```

**D. Giai đoạn 4: Kết thúc kết nối bảo mật**

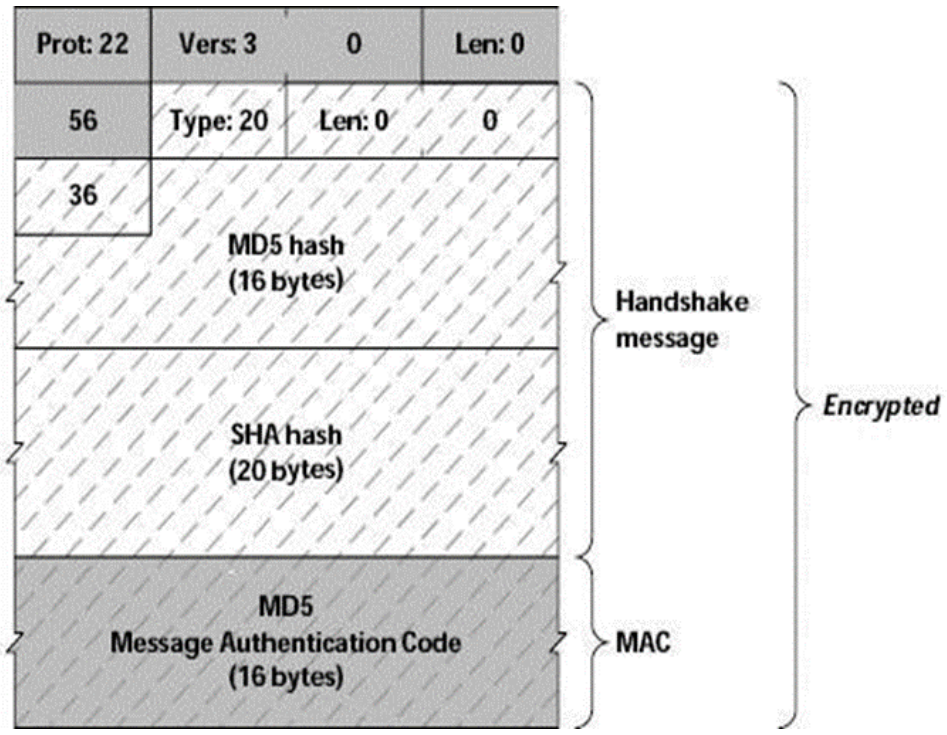
Lúc này, một thông điệp ChangeCipherSpec được gửi bởi client, và client sao chép lại CipherSpec trạng thái chờ vào CipherSpec hiện tại. Sau đó client gửi thông điệp kết thúc dưới các thuật toán mới, các khoá. Để đáp ứng lại, server sẽ gửi thông điệp ChangeCipherSpec của mình, chuyển CipherSpec chờ thành CipherSpec hiện tại, sau đó gửi thông điệp kết thúc dưới CipherSpec mới. Vào lúc này, giai đoạn bắt tay kết thúc và client và server có thể bắt đầu trao đổi dữ liệu tầng ứng dụng.

**a. Thông điệp ChangeCipherSpec**

Client gửi một thông điệp ChangeCipherSpec và sao chép trạng thái CipherSpec chờ vào CipherSpec hiện tại. Thông điệp này được gửi ngay sau thông điệp CertificateVerify. Nó thể hiện rằng một thông điệp ChangeCipherSpec được nhận giữa các thông điệp handshake và các thông điệp kết thúc. Sẽ xảy ra một lỗi nếu thông điệp ChangeCipherSpec không được theo sau bởi thông điệp Finished.

**b. Thông điệp Finished**

Thông điệp này luôn được gửi ngay sau thông điệp CipherSpec để kiểm tra lại các tiến trình xác thực và trao đổi khoá đã thành công. Kiểu thông điệp handshake kết thúc này là 20. Thông điệp Finish bản thân nó được mã hoá sử dụng các tham số cipher suite.

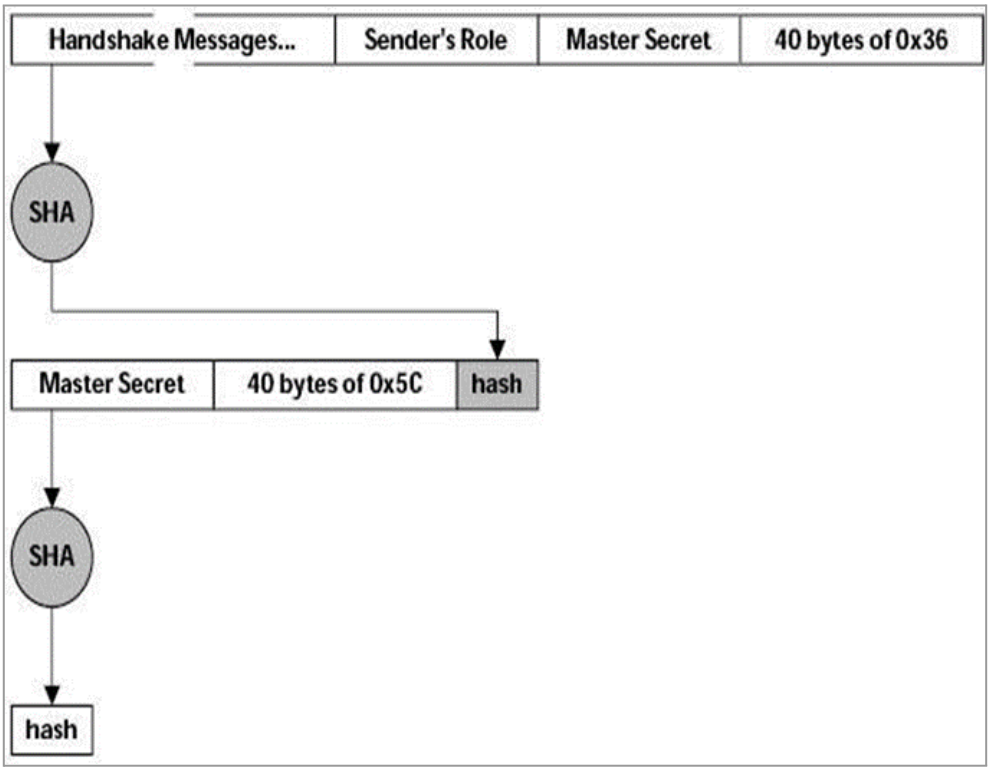



*Hình 18. Thông điệp Finished*

Phần thân thông điệp Finish bao gồm kết quả của hai hàm băm, một sử dụng thuật toán băm MD5, và một sử dụng SHA. Cả hai sự tính toán này đều sử dụng đầu vào như nhau, và đều tính trong hai bước. Ngoài ra còn có một giá trị MAC được tạo ra từ

MD5/SHA và khóa bí mật để nhằm xác minh thông điệp không bị thay đổi và được tạo ra bởi một client/server xác định

Đầu tiên người gửi tạo một hàm băm nội dung đầy đủ các thông điệp handshake SSL đã trao đổi trước đó trong suốt một phiên, theo sau là một quy tắc của người gửi, master secret và padding. Quy tắc của người gửi là một giá trị hexa 434c4e54 nếu người gửi là client, 53525652 nếu là server. Padding là giá trị nhị phân 001100110, lặp lại 48 lần cho MD5, 40 lần cho SHA. Bước thứ hai, người gửi tạo ra một hàm băm mới sử dụng master secret, theo sau là một padding thay đổi và đầu ra của hàm băm trước. Padding bước hai là một giá trị nhị phân 01011100, lặp lại 48 lần cho MD5 và 40 lần cho SHA.




*Hình 19. Thông điệp Finished bao gồm một hàm băm*

Chú ý rằng, có một sự giống nhau giữa tính toán này và tính toán hàm băm cho thông điệp CertificateVerify. Tuy nhiên cũng có hai điểm khác biệt. Đầu tiên, hàm băm Finished bao gồm quy tắc của người gửi trong khi CertificateVerify thì không. Thứ hai, tập hợp các thông điệp handshake sẽ khác khi hai hàm băm được tính. Trong mỗi trường hợp, chú ý rằng SSL không đề cập tới ChangeCipherSpec, vì thế nội dung của nó không được tính tới trong hàm băm.Sau khi thông điệp Finished của server được gửi đi, quá trình bắt tay hoàn thành và một phiên SSL được thiết lập. Từ lúc này trở đi, việc trao đổi dữ liệu giữa hai bên đều được mã hoá cho tới khi kết thúc một phiên.

### *2.2.2. SSL Change Cipher Spec Protocol*

Giao thức Change Cipher Spec Protocol được coi là giao thức đơn giản nhất trong ba giao thức đặc trưng của SSL. Giao thức này thuộc lớp giao thức trên (upper-layer protocol). ULP là lớp giao thức xuất hiện trong 3 tầng đầu tiên của mô hình mạng OSI, giúp phần mềm chạy trên server và thiết bị

Khi một bên (ví dụ như máy khách hoặc máy chủ) muốn chuyển từ việc sử dụng thông số mật mã cũ sang thông số mật mã mới, nó sẽ gửi một thông điệp CCS là 1 bản tin đến bên kia của kết nối.

Bản tin này chỉ đơn giản là một byte mang giá trị là 1. Khi bản tin đơn này được gửi đi,  nó sinh ra trạng thái tiếp theo để gán vào trạng thái hiện tại và trạng thái hiện tại cập nhật lại bộ mã hoá để sử dụng trên kết nối này

SSL ChangeCipherSpec Protocol được sử dụng để thay đổi giữa một thông số mật mã này và một thông số mật mã khác.Mặc dù thông số mật mã thường được thay đổi ở cuối một sự thiết lập quan hệ SSL,nhưng nó cũng có thể được thay đổi vào bất kỳ thời điểm sau đó

Giao thức này là một phần không thể thiếu của quá trình chuyển đổi từ trạng thái thiết lập kết nối (handshake) sang trạng thái hoạt động trong SSL/TLS. Nó đảm bảo rằng cả hai bên của kết nối đang sử dụng cùng một bộ thông số mật mã và đảm bảo rằng dữ liệu được truyền tải an toàn trong suốt quá trình giao tiếp.

### *2.2.3. SSL Alert Protocol*

Giao thức SSL Alert được dùng để truyền cảnh báo hoặc thông báo lỗi đến đầu cuối của bên kia (có thể là khách hàng, là server hoặc là người dùng cuối), hay chính là việc giáo thức này đưa ra thông báo mang tính ràng buộc trong quá trình giao tiếp giữa trình duyệt web là web server

Tất cả thông báo được gửi đến những thực thể ngang hàng nhau sẽ được nén và mã hoá hoàn toàn, có nghĩa là những thông báo đó sẽ không thể bị lấy hay nhìn thấy từ bên thứ ba.

Có 2 loại Alert:

- **Warning Alert:** Được sử dụng để cảnh báo về các vấn đề nhưng không gây ra việc ngắt kết nối
- **Fatal Alert**: Được sử dụng để báo cáo về các vấn đề nghiêm trọng mà làm cho việc duy trì kết nối an toàn không thể tiếp tục.

Một số các cảnh báo thường gặp trong SSL Alert:

- **unexpected_message:** thông điệp nhận không phù hợp với phiên hiện tại, có thể do không đúng định dạng hoặc không đúng thứ tự

- **bad_record_mac:** mã MAC (Message Authentication Code) hiện tại không chính xác
- **decompression_failure:** lỗi trong quá trình giải nén dữ liệu
- **handshake_failure:** lỗi trong quá trình handshakes
- **unsupported_certificate**: dạng chứng chỉ không được hỗ trợ
- **certificate_revoked**: Chứng chỉ đã bị thu hồi bởi nhà cung cấp
- **certificate_expired:** Chứng chỉ đã hết hạn đăng ký

SSL Alert Protocol được sử dụng để chuyển các cảnh báo thông qua SSL Record Protocol. Mỗi cảnh báo gồm 2 phần, một mức cảnh báo (warning, fatal) và một mô tả cảnh báo.

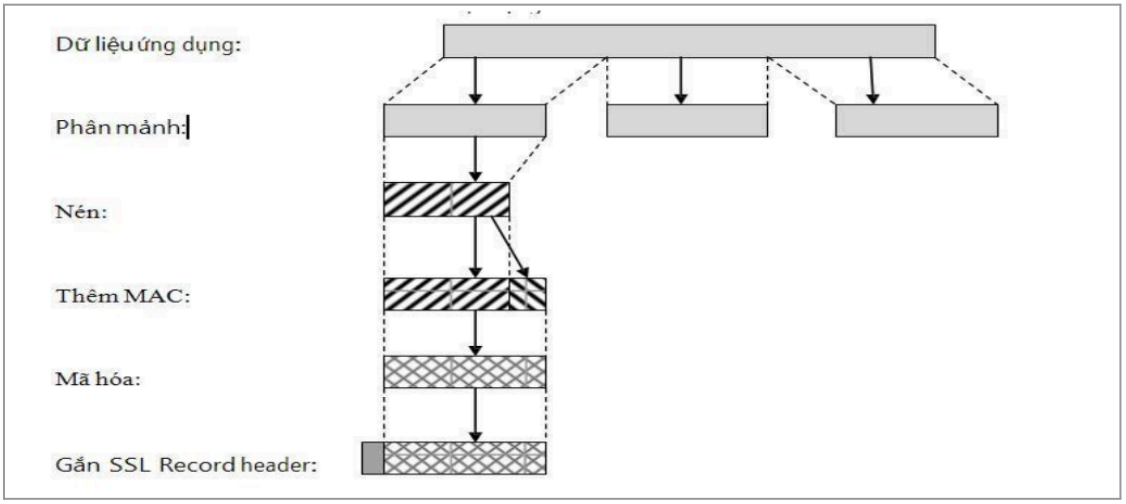
### *2.2.4. SSL Record Protocol*

SSL Record Protocol xác định khuôn dạng cho tiến hành mã hóa và truyền tin hai chiều giữa hai đối tượng.

SSL Record Protocol cung cấp 2 dịch vụ cho kết nối SSL:

- **Tính bảo mật (Confidentiality):** Handshake Protocol định nghĩa 1 khóa bí mật được chia sẻ, khóa này được sử dụng cho mã hóa quy ước các dữ liệu SSL
- **Tính toàn vẹn thông điệp (Message integrity):** Handshake Protocol cũng định nghĩa 1 khóa bí mật được chia sẻ, khóa này được sử dụng để hình thành MAC (mã xác thực message).

Trong SSL Record Layer, dữ liệu ứng dụng được chia thành các mảnh. Các mảnh này được nén và sau đó mã hóa MAC (Message Authentication Code) được tạo bởi các thuật toán như SHA (Giao thức băm an toàn) và MD5 (Thông báo tin nhắn). Cuối cùng, sau khi mã hóa dữ liệu được thực hiện xong thì SSL header sẽ được thêm vào dữ liệu.



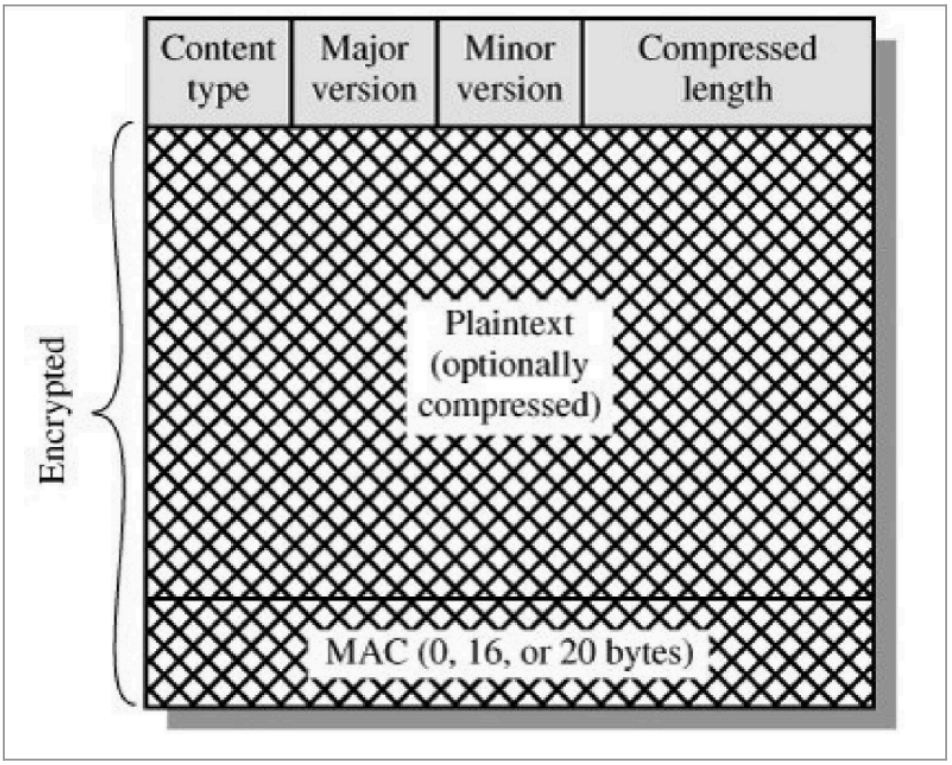

*Hình 20. Toàn bộ hoạt động của SSL Record Protocol*

Bước đầu tiên là phân mảnh. Mỗi thông điệp ở lớp trên được phân mảnh thành các khối có kích thước 16384 bytes trở xuống. Tiếp theo, việc nén được áp dụng tuỳ chọn. Quá trình nén không được mất mát và không làm tăng độ dài của nội dung lên quá 1024 byte. Bước tiếp theo trong quá trình xử lý là tính toán mã MAC trên dữ liệu đã nén. Với bước này, một khoá bí mật chung sẽ được sử dụng.

Tiếp theo, thông điệp được nén cộng với mã MAC được mã hoá sử dụng mã hoá đối xứng, và mã hoá không được tăng độ dài nội dung lên quá 1024 byte, do đó tổng chiều dài nội dung không được vượt quá 16384 + 2048 byte.

Bước cuối cùng là thêm vào phía trước một header, bao gồm các trường như sau:

- Loại nội dung (8 bit): Giao thức lớp cao hơn được sử dụng để xử lý mảnh kèm theo
- Phiên bản chính (8 bit): Cho biết phiên bản chính của SSL đang được sử dụng. Ví dụ, đối với SSLv3, giá trị là 3
- Phiên bản phụ: Cho biết phiên bản phụ đang được sử dụng. Ví dụ, đối với SSLv3, giá trị là 0
- Độ dài sau khi nén (16 bit): Độ dài được tính bằng byte của đoạn văn bản gốc hoặc đoạn được nén nếu sử dụng tính năng nén, giá trị tối đa là 16384 + 2048




*Hình 21. SSL Record format*

### *2.3. Lỗ hổng bảo mật và cách phòng chống:*

#### *2.3.1. POODLE:*


*Hình 22. Chiến lược tấn công POODLE.*

POODLE là viết tắt của Padding Oracle on Downgraded Legacy Encryption. Đây là một chiến lược tấn công được dùng để đánh cắp thông tin bí mật từ các kết nối được bảo mật bằng giao thức SSL. Lỗ hổng này cho phép kẻ tấn công nghe lén giao tiếp HTTPS được mã hóa bằng cách dùng giao thức SSL 3.0.

**Ảnh hưởng:**

Tấn công POODLE có thể được sử dụng để chống lại bất kỳ hệ thống hoặc ứng dụng nào hỗ trợ SSL 3.0 với chế độ mã hóa CBC. Điều này không chỉ gây ảnh hưởng đến hầu hết các trình duyệt và các website hiện nay, mà còn gây hậu quả cho bất kỳ phần mềm nào tham chiếu tới một thư viện SSL/ TLS dễ bị tổn thương (ví dụ: OpenSSL) hoặc thực hiện gói giao thức SSL/ TLS. Bằng cách khai thác lỗ hổng này trong những tình huống có yếu tố liên quan tới website, kẻ tấn công có thể nắm quyền để truy cập được vào những dữ liệu nhạy cảm được truyền trong phiên website được mã hóa, ví dụ như mật khẩu, cookie và các mã thông báo xác thực khác để giành quyền truy cập website một cách đầy đủ hơn (giả mạo người dùng đó, truy cập nội dung cơ sở dữ liệu,...).

**Thực tế:**

Thực tế hiện nay cho thấy, báo cáo về lỗ hổng ứng dụng web Acunetix 2020 gần đây cho thấy có tới 3,9% máy chủ web vẫn dễ bị tấn công bởi POODLE, có nghĩa là chúng vẫn hỗ trợ giao thức SSL 3.0, mặc dù thực tế là giao thức TLS đã được giới thiệu vào năm 1999. Tại sao các máy chủ web lại vẫn hỗ trợ các giao thức cũ? Mặc dù SSL 3.0 là một tiêu chuẩn mã hóa đã trở nên cũ kỹ, ngày nay, SSL thường được thay thế bởi TLS, hầu hết các thực thi SSL/ TLS vẫn tương thích với SSL 3.0 để liên kết hoạt động với các hệ thống kế thừa, vì lợi ích của trải nghiệm người dùng. Ngay cả khi client và server đều được hỗ trợ phiên bản TLS, bộ giao thức SSL/ TLS vẫn cho phép đàm phán phiên bản giao thức (hay còn được gọi là "downgrade dance" (sự hạ cấp)

trong các báo cáo khác). Cuộc tấn công POODLE làm rõ một thực tế đó là, khi xuất hiện một kết nối không an toàn, các servers sẽ trở lại các giao thức cũ hơn, như SSL 3.0.

**Quá trình thực hiện tấn công:**

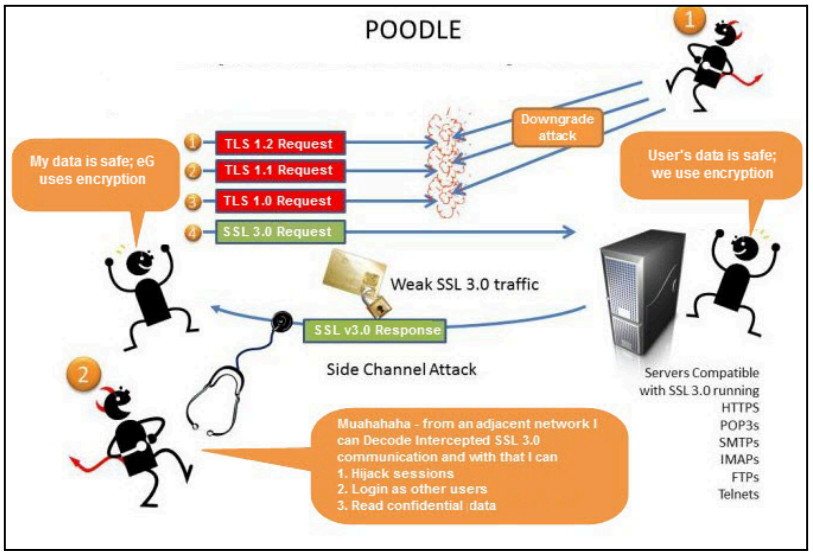
Cuộc tấn công không hề dễ dàng vì nó cần phải thành công trong ba giai đoạn:

+ Ở giai đoạn đầu tiên, kẻ tấn công phải thực hiện thành công cuộc tấn công trung gian (MITM). Kẻ tấn công giờ đây có thể lắng nghe tất cả thông tin liên lạc giữa máy khách và máy chủ cũng như thêm vào thông tin liên lạc này (mạo danh máy khách hoặc máy chủ). Tuy nhiên, nếu đây là kết nối an toàn thì thông tin liên lạc sẽ được mã hóa bằng SSL/TLS, do đó kẻ tấn công không thể hiểu nội dung đang được gửi.

+ Ở giai đoạn thứ hai, kẻ tấn công phải thuyết phục máy chủ sử dụng giao thức SSL 3.0 cũ. Kẻ tấn công có thể thực hiện điều này bằng cách ngắt kết nối - sau một số lần ngừng kết nối như vậy, máy chủ sẽ thử giao thức cũ hơn vì nghĩ rằng máy khách không thể sử dụng giao thức mới hơn như TLS 1.2. Đây được gọi là một cuộc tấn công hạ cấp giao thức hoặc nhảy hạ cấp.

+ Ở giai đoạn thứ ba, khi máy khách và máy chủ đang giao tiếp bằng SSL 3.0, kẻ tấn công có thể sử dụng cuộc tấn công POODLE để giải mã các phần được chọn của giao tiếp và đánh cắp thông tin bí mật.

Để đảm bảo cuộc tấn công POODLE thành công, kẻ tấn công cũng có thể lừa trình duyệt người dùng chạy JavaScript, chẳng hạn như sử dụng social engineering.




*Hình 23. Quá trình thực hiện tấn công POODLE*

**Cách phòng chống:**

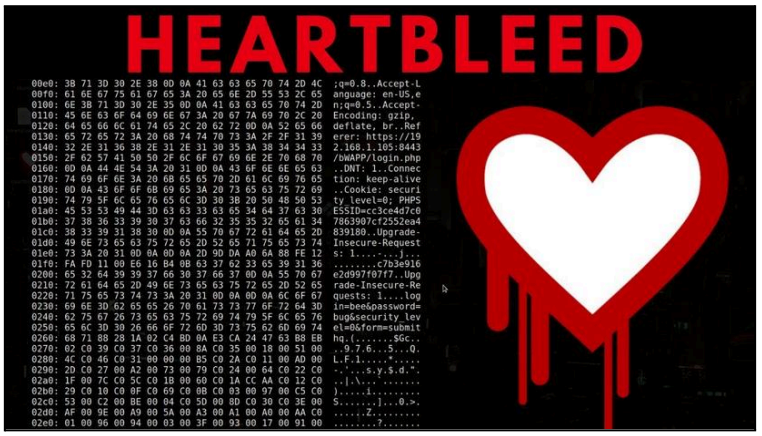
Hiện tại, không có bản sửa lỗi nào cho lỗ hổng SSL 3.0, vì đây là một trong những vấn đề khá cơ bản sẽ gặp phải trong một protocol. Tuy nhiên, vô hiệu hóa hỗ

trợ SSL 3.0 trong các cấu hình của hệ thống/ ứng dụng là giải pháp khả thi nhất hiện có.

Một số nhà nghiên cứu đã phát hiện lỗ hổng, họ đồng thời phát triển một cách sữa lỗi cho một tình huống có điều kiện nhất định như sau đây. TLS_FALLBACK_SCSV là sự mở rộng giao thức, nhằm mục đích ngăn cản những kẻ tấn công MITM không thể ép buộc một sự hạ cấp giao thức (downgrade). OpenSSL đã thêm hỗ trợ TLS_FALLBACK_SCSV vào các phiên bản mới nhất của họ. Cả client và server đều cần hỗ trợ TLS_FALLBACK_SCSV để ngăn chặn các cuộc tấn công hạ cấp (downgrade).

### *2.3.2. Heartbleed:*

Heartbleed là một lỗ hổng được tìm phát hiện vào tháng 4 năm 2014. Heartbleed cho phép những kẻ tấn công truy cập dễ dàng vào thông tin bí mật, nhạy cảm của các dữ liệu cá nhân. Chúng có thể xâm nhập và lấy đi hàng tá thông tin cá nhân bao gồm tất cả tin nhắn, email, mật khẩu, ... và rút lui không một dấu vết.




*Hình 24. Lỗ hổng Heartbleed*

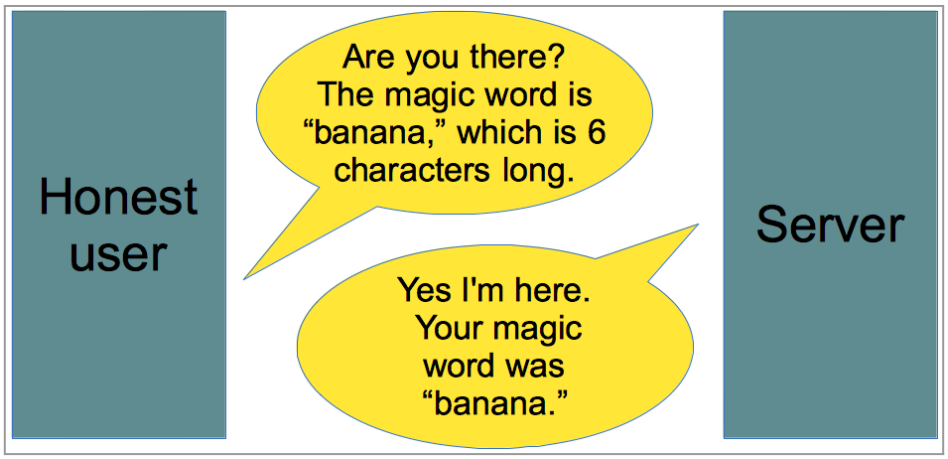
**Ảnh hưởng:**

Heartbleed là một lỗ trong OpenSSL, một thư viện mã nguồn mở dùng để triển khai TLS (Transport Layer Security) và SSL (Secure Sockets Layer), được Google, Facebook, Yahoo, Amazon và rất nhiều trang web lớn trên thế giới sử dụng để bảo vệ việc truyền tải thông tin cá nhân của người dùng. OpenSSL được sử dụng rộng rãi. Một lý do cho điều này là nó đã được tích hợp vào nhiều sản phẩm phần mềm khác. Ví dụ: hai trong số các gói phần mềm máy chủ web phổ biến nhất, được gọi là Apache và nginx, đều sử dụng OpenSSL để mã hóa trang web.

Nói tóm lại, một kẻ tấn công có thể dễ dàng lừa một máy chủ web gặp lỗ hổng này gửi thông tin nhạy cảm cho hắn, bao gồm cả username và password.

**Quá trình thực hiện tấn công:**

Heartbleed hoạt động dựa trên quá trình gọi là “heartbeat” của giao thức SSL, TLS. Ví dụ hai máy tính A và B đang kết nối với nhau. Thỉnh thoảng, máy tính A sẽ gửi một phần dữ liệu được mã hóa, được gọi là “heartbeat request”, đến máy B. Máy tính B sẽ trả lời lại cùng với một phần dữ liệu được mã hóa, chứng minh rằng kết nối vẫn còn. Tính năng này rất hữu ích vì một số bộ định tuyến internet sẽ ngắt kết nối nếu không hoạt động quá lâu.



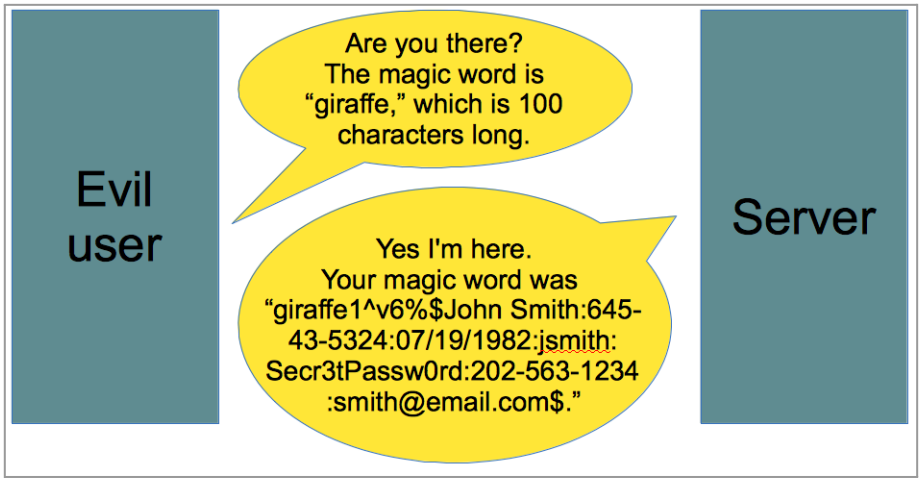

*Hình 25. Minh họa quá trình “heartbeat” của SSL*

Một “heartbeat message” có ba phần:

+ Yêu cầu xác nhận
+ Một tin nhắn ngắn, được chọn ngẫu nhiên (trong trường hợp này là "banana").
+ Số ký tự trong tin nhắn đó.

Máy chủ chỉ cần xác nhận đã nhận được yêu cầu và gửi lại tin nhắn.

Cuộc tấn công Heartbleed lợi dụng thực tế là máy chủ có thể quá tin tưởng. Khi ai đó nói với nó rằng tin nhắn có 6 ký tự, máy chủ sẽ tự động gửi lại 6 ký tự để phản hồi. Một kẻ tấn công có thể lợi dụng sự cả tin của máy chủ:




*Hình 26. Minh họa lợi dụng lỗ hổng Heartbleed*

Rõ ràng, từ "giraffe" không dài 100 ký tự. Nhưng máy chủ không thèm kiểm tra trước khi gửi lại phản hồi nên nó gửi lại 100 ký tự. Cụ thể, nó sẽ gửi lại từ "giraffe" gồm 7 ký tự, theo sau là 93 ký tự bất kỳ được lưu trữ sau từ "giraffe" trong bộ nhớ đệm của máy chủ. Máy tính thường lưu trữ thông tin trong bộ nhớ đệm theo thứ tự lộn xộn nhằm cố gắng gói nó vào bộ nhớ chặt chẽ nhất có thể, do đó không thể biết được thông tin nào có thể được trả về. Trong trường hợp này, đoạn bộ nhớ sau từ "giraffe" chứa thông tin cá nhân nhạy cảm của người dùng John Smith.

Trong cuộc tấn công Heartbleed thực sự, kẻ tấn công không chỉ yêu cầu 100 ký tự. Kẻ tấn công có thể yêu cầu khoảng 64.000 ký tự văn bản thuần túy. Và nó không chỉ hỏi một lần, nó có thể gửi đi gửi lại nhiều tin nhắn nhịp tim độc hại, cho phép kẻ tấn công lấy lại các mảnh bộ nhớ khác nhau của máy chủ mỗi lần. Trong quá trình này, nó có thể thu được vô số dữ liệu nhạy cảm từ máy chủ.

**Cách phòng chống:**

Cách khắc phục lỗ hổng Heartbleed là nâng cấp lên phiên bản OpenSSL mới nhất bởi các bản vá lỗi đã được liên tục tung ra cho OpenSSL ngay khi lỗ hổng được công bố.

Cụ thể, bởi OpenSSL là một mã nguồn mở nên có thể dễ dàng xem được đoạn mã thực hiện sửa lỗi:

```
/* Read type and payload length first */

if (1 + 2 + 16 > s->s3->relent)
    return 0;

/* silently discard */
hbtype = *p++;
n2s(p, payload);

if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0;

/* silently discard per RFC 6520 sec. 4 */
pl = p;
```

*Hình 27. Đoạn mã sửa lỗi khắc phục lỗ hổng Heartbleed*

Đoạn mã sửa lỗi bao gồm 2 phần:

+ Kiểm tra xem tổng độ dài của tin nhắn heartbeat (bao gồm loại, độ dài payload và padding) có vượt quá số byte còn lại trong bộ đệm không. Nếu vượt quá, điều này có nghĩa rằng tin nhắn đầu vào không đủ dài để chứa một tin nhắn heartbeat hợp lệ, và do đó, hàm sẽ trả về 0, chỉ ra rằng tin nhắn không thể được xử lý tiếp.

+ Kiểm tra xem tổng độ dài của tin nhắn heartbeat có vượt quá độ dài của bản ghi TLS không. Nếu vượt quá, tin nhắn heartbeat chứa quá nhiều dữ liệu so với kích thước của bản ghi TLS.

### *2.3.3. BEAST Attack:*

Thuật ngữ BEAST Attack dùng để chỉ một hình thức khai thác bảo mật. BEAST, từ viết tắt của Browser Exploit Against SSL/TLS (Khai thác trình duyệt chống lại SSL/TLS), là mối lo ngại đáng kể về an ninh mạng trong thập kỷ qua. Trọng tâm chính của cuộc tấn công như vậy là giải mã thông tin nhạy cảm, chẳng hạn như số thẻ tín dụng hoặc dữ liệu cá nhân, được cho là được bảo vệ bằng thuật toán mã hóa trong khi truyền từ máy chủ đến người dùng và ngược lại.

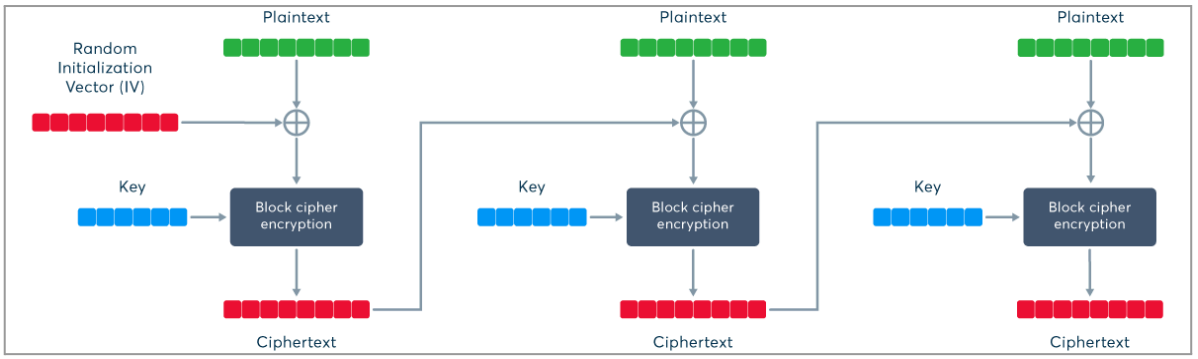
**Ảnh hưởng:**

Lỗ hổng BEAST chủ yếu ảnh hưởng đến SSL 3.0 và TLS 1.0, sử dụng chế độ khai thác chuỗi khối mật mã (CBC). Nghiên cứu tiết lộ rằng 30,7% máy chủ web sử dụng TLS 1.0 bất lực không thể tự vệ trước cuộc tấn công BEAST.

**Giải thích hoạt động:**

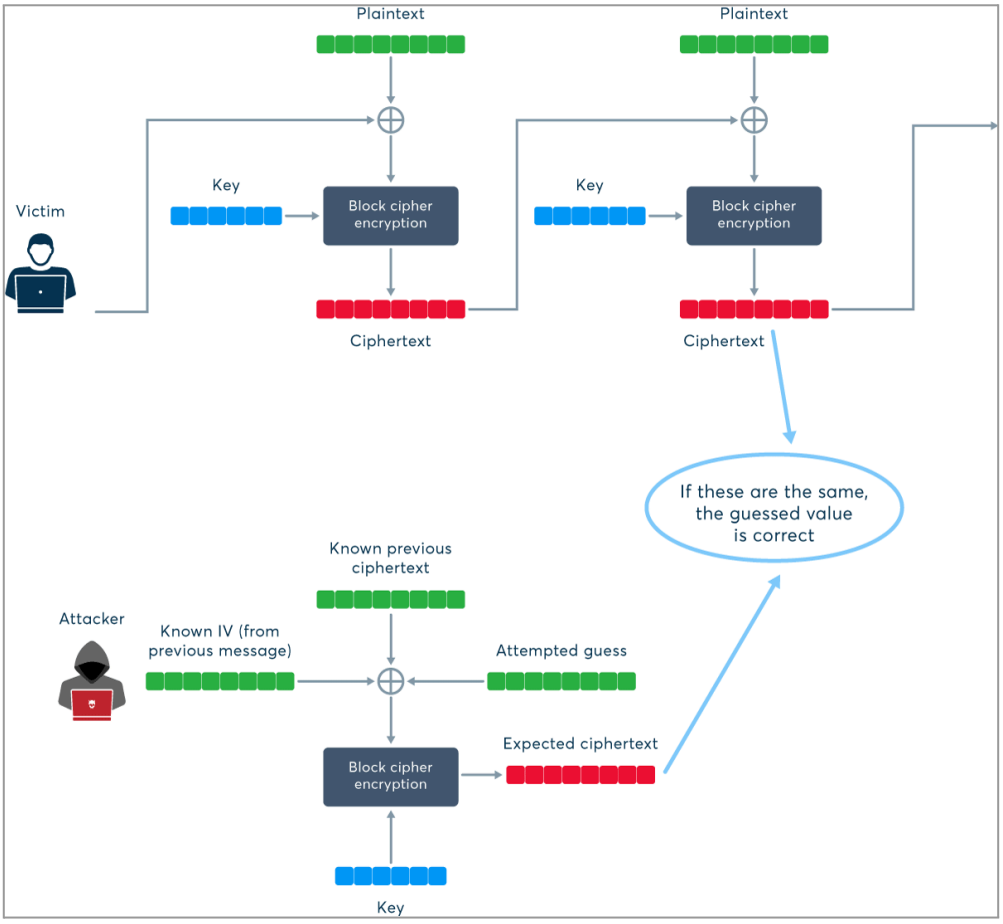
+ Chế độ CBC (Cipher Block Chaining): Là chế độ mà mỗi khối bản rõ được kết hợp (XOR) với khối bản mã trước đó, vì vậy giá trị của mỗi khối phụ thuộc vào tất cả các khối trước đó.

Trong chế độ CBC, khối đầu tiên được kết hợp với vectơ khởi tạo (IV) – một khối dữ liệu ngẫu nhiên làm cho mỗi tin nhắn trở nên duy nhất. Tính bảo mật của bất kỳ mật mã khối nào trong chế độ CBC phụ thuộc hoàn toàn vào tính ngẫu nhiên của các vectơ khởi tạo. Và đây là vấn đề: trong TLS 1.0, các vectơ khởi tạo không được tạo ngẫu nhiên. Thay vì tạo IV mới cho mỗi tin nhắn, giao thức sử dụng khối bản mã cuối cùng từ tin nhắn trước đó làm IV mới. Điều này mở ra một lỗ hổng nghiêm trọng vì bất kỳ ai chặn dữ liệu được mã hóa cũng nhận được các vectơ khởi tạo. Các khối được kết hợp bằng XOR, đây là một hoạt động có thể đảo ngược, vì vậy việc biết các vectơ khởi tạo có thể cho phép kẻ tấn công khám phá thông tin từ các tin nhắn được mã hóa.




*Hình 28. Minh họa chế độ CBC*

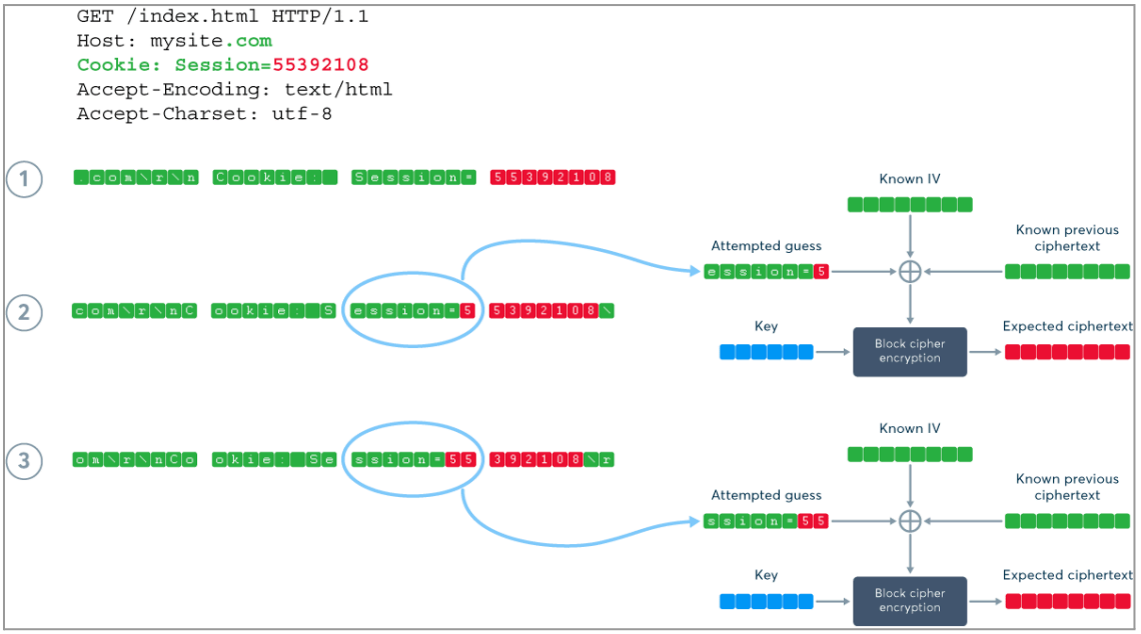
+ Kẻ tấn công có thể tiêm một khối dữ liệu đặc biệt và kiểm tra xem khối được mã hóa kết quả có giống với khối tương ứng trong luồng tin nhắn thực tế hay không. Nếu có, dự đoán là chính xác và kẻ tấn công đã phát hiện ra khối bản rõ. Nếu không, họ có thể thử đi thử lại với các giá trị có khả năng khác nhau. Đây được gọi là một cuộc tấn công chia tách bản ghi.




*Hình 29. Kiểm tra khối dự đoán mà không cần giải mã*

+ Blockwise Chosen Boundary Attack (được công bố bởi hai nhà khoa học an ninh Thái Dương và Juliano Rizzo năm 2011):

Với mục đích để giảm thời gian vét cạn và dựa trên cấu trúc cố định có phần cứng nhắc của HTML. Cụ thể kẻ tấn công cẩn thận tạo các mã HTML có cấu trúc, kiểm soát ranh giới các khối mã để tạo ra một khối dự đoán mà ở đó chỉ có duy nhất 1 byte không xác định (các byte còn lại đều được xác định theo cấu trúc) và thường là các cookie session. Vì vậy, thay vì vét cạn trên một khối kẻ tấn công chỉ cần vét cạn trên 1 byte. Sau khi xác định được byte không xác định, tiến hành dịch chuyển ranh giới để xác định các byte chưa rõ còn lại.




*Hình 30. Minh họa Blockwise Chosen Boundary Attack*

**Cách phòng chống:**

+ Cập nhật phiên bản TLS: Sử dụng phiên bản TLS mới nhất có sẵn, hiện nay là TLS 1.3. TLS 1.3 cung cấp nhiều cải tiến bảo mật so với các phiên bản trước đó và có thể giúp giảm thiểu rủi ro của các cuộc tấn công Beast.

+ Sử dụng cipher suites an toàn: Cấu hình máy chủ để sử dụng các cipher suites mạnh mẽ và an toàn như AES-GCM hoặc ChaCha20-Poly1305. Tránh sử dụng các cipher suites cũ yếu như RC4 hoặc các phiên bản cũ của AES.

+ Tắt hỗ trợ SSL 3.0 và TLS 1.0/1.1: Các phiên bản này đã bị phá vỡ và không an toàn. Thay vào đó, chỉ kích hoạt hỗ trợ cho TLS 1.2 hoặc cao hơn.

+ Giảm thiểu việc sử dụng cookie nhạy cảm trong cùng một truy cập.

Bên cạnh những lỗ hổng đã đề cập phía trên, SSL/TLS từng đối mặt với nhiều lỗ hổng, chiến lược tấn công khác như CRIME (Compression Ratio Info-leak Made Easy) - một phương pháp tấn công cho phép kẻ tấn công xâm nhập thông tin nhạy cảm trong các yêu cầu HTTP được nén và gửi qua kết nối SSL/TLS, FREAK (Factoring RSA Export Keys) - cho phép tin tặc giải mã thông tin được mã hóa bằng SSL/TLS sử dụng các khóa RSA export có độ dài ngắn… Nhìn chung, cách phòng chống tốt nhất vẫn là sử dụng phiên bản TLS mới nhất khi TLS 1.3 cung cấp nhiều cải tiến bảo mật, hạn chế rủi ro so với các phiên bản TLS và SSL cũ.

## 3. KẾT LUẬN, ĐÁNH GIÁ GIAO THỨC SSL/TLS

Thông qua nội dung đã trình bày ở trên, ta đã tìm hiểu chi tiết cách hoạt động của giao thức SSL/TLS. Cụ thể, ở chương 1, ta đã tìm hiểu tổng quan về SSL/TLS, khái quát lịch sử hình thành để có cái nhìn tổng quan về giao thức. Chương 2, ta bắt đầu tìm hiểu sâu về cấu trúc, cách hoạt động của giao thức, giới thiệu về giao thức bắt tay (Handshake). Hiểu được quá trình bắt tay (handshake) để tạo kết nối an toàn giữa client và server và ghi lại (record) để cung cấp tính bảo mật và toàn vẹn dữ liệu cho việc truyền các gói tin. Giao thức Change Cipher Spec đảm bảo rằng cả hai bên của kết nối đang sử dụng cùng một bộ thông số mật mã và đảm bảo rằng dữ liệu được truyền tải an toàn trong suốt quá trình giao tiếp. Giao thức SSL Alert dùng để truyền cảnh báo hoặc thông báo lỗi đến đầu cuối của bên kia. SSL Record Layer xác định khuôn dạng cho tiến hành mã hóa và truyền tin hai chiều giữa hai đối tượng. Tuy vậy, giao thức này vẫn có những lỗ hổng có thể khiến thông tin bị đánh cắp, vì vậy ta cần phải tìm những hướng giải quyết và cách phòng chống phù hợp.

Chức năng cốt lõi của chứng chỉ SSL là bảo vệ giao tiếp giữa máy chủ và máy khách. Khi triển khai SSL, tất cả các bit dữ liệu đều được mã hóa. Dữ liệu được mã hóa và chỉ có thể được giải mã bởi bên nhận dự kiến (trình duyệt hoặc máy chủ) vì không ai khác có chìa khóa để mở nó. Trong quá trình truyền dữ liệu nhạy cảm như ID, mật khẩu, số thẻ tín dụng, và những thông tin tương tự, SSL giúp bảo vệ chúng khỏi các kẻ tấn công và tin tặc có ý đồ xấu. Bằng cách chuyển dữ liệu sang dạng không thể giải mã được, SSL làm cho kỹ năng của các tin tặc trở nên vô dụng, tạo ra một rào cản mạnh mẽ trước công nghệ mã hóa tiên tiến của nó.

Nhiệm vụ quan trọng thứ hai của chứng chỉ SSL là cung cấp xác thực cho một trang web. Xác minh danh tính là một trong những khía cạnh quan trọng nhất liên quan đến bảo mật web, giúp chúng ta có thể tránh được những trang web không có tính bảo mật. Hầu hết trình duyệt hiện nay đều cảnh báo (hoặc thậm chí chặn) khi người dùng cố gắng truy cập vào trang web không có bảo mật SSL (những trang web có giao thức HTTP). Vì vậy, chứng chỉ SSL trở nên cực kỳ quan trọng cho một trang web.

Ngày nay, việc bảo vệ thông tin là yếu tố cực kỳ quan trọng ảnh hưởng đến sự tồn tại của một tổ chức, công ty hoặc doanh nghiệp. Sự tiến bộ nhanh chóng của công nghệ đã mang lại nhiều lợi ích cho người dùng, song đồng thời cũng tạo ra nhu cầu ngày càng cao về an toàn và bảo mật. Qua việc tìm hiểu về SSL, ta có thể thấy rằng việc áp dụng giao thức SSL là giải pháp tốt nhất hiện nay để đáp ứng những nhu cầu này, và nó được coi là lớp bảo vệ cuối cùng trong bảo mật thương mại điện tử.

**TÀI LIỆU THAM KHẢO:**

[1] **Bài giảng An toàn và bảo mật hệ thống thông tin** - Học viện Công nghệ Bưu chính Viễn thông - Hoàng Xuân Dậu.
[2] **What is BEAST Attack?** - ReasonLabs
https://cyberpedia.reasonlabs.com/EN/beast%20attack.html
[3] **What is SSL? | SSL definition -** Cloudflare
https://www.cloudflare.com/learning/ssl/what-is-ssl/
[4] **What is SSL/TLS: An In-Depth Guide -** SSL.com
https://www.ssl.com/article/what-is-ssl-tls-an-in-depth-guide/