BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

MẬT MÃ ỨNG DỤNG TRONG AN TOÀN THÔNG TIN

(CHUẨN MÃ HÓA RSA)

Chuyên ngành hẹp: An toàn ứng dụng

Ngành: An toàn thông tin

Hà Nội, tháng 9 năm 2021

BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

MẬT MÃ ỨNG DỤNG TRONG AN TOÀN THÔNG TIN

(CHUẨN MÃ HÓA RSA)

Chuyên ngành hẹp: An toàn ứng dụng

Ngành: An toàn thông tin

Giảng viên hướng dẫn:

- Lục Như Quỳnh

Nhóm sinh viên thực hiện:

- Nhóm 4: 12 thành viên

Hà Nội, tháng 9 năm 2021

MỤC LỤC

PHẦN MỞ ĐẦU

Ngày nay công nghệ thông tin đang phát triển, giúp ích rất nhiều trong đời sống chúng ta. Nhờ có nó mà mọi người có thể liên lạc với nhau một cách dễ dàng hơn bằng nhiều cách như gọi điện, call video hay nhắn tin, … Bên cạnh đó, thì luôn có nhưng kẻ nhòm ngó để có thể lấy cắp được thông tin nhằm mục đích xấu cho bản thân. Vậy nên đã có rất nhiều người chọn cách mã hóa tin nhắn để tăng tính bảo mật. Nhưng nói đến tin nhắn thì sẽ tồn tại người gửi và người nhận, vậy làm thế nào có thể gửi cho ai đó một tin nhắn được mã hóa mà không có cô hội trước đó? Đây cũng là nội dung mà nhóm chúng em đã tìm hiểu, chính là chuẩn mã hóa RSA.

Trong quá trình tìm hiểu về RSA, tuy thời gian không nhiều nhưng với sự hướng dẫn và giúp đỡ của giáo viên, nhóm em đã hoàn thành. Do thời gian hạn chế nên phạm vi nghiên cứu và vẫn còn một số vấn đề chưa được giải quyết triệt để. Nhóm em mong nhận được sự đóng góp của giáo viên và các bạn, nhóm em xin chân thành cảm ơn thầy giáo và các bạn !

CHƯƠNG I. TỔNG QUAN

1.1 Mật mã khóa công khai

Năm 1976, Whitfield Diffie và Martin Hellman đã đề xuất khái niệm mật mã hóa khóa bất đối xứng, và đưa ra phương pháp trao đổi khóa công khai. Trước khi khái niệm mật mã học bất đối xứng ra đời, để đảm bảo tính bí mật khi trao đổi thông tin, các thuật toán mã hóa thông tin được xây dựng trên mật mã học đối xứng. Trong loại mật mã này, giữa các bên trao đổi thông tin cần phải cần gặp gỡ hoặc trao đổi qua một kênh bảo mật một khóa bí mật. Điều này đã hạn chế ứng dụng của mật mã học vào việc đảm bảo tính riêng tư, bí mật trong cuộc sống, và sự thật thì cho tới những năm đó, mật mã học chủ yếu chỉ được ứng dụng trong lĩnh vực quân sự, an ninh quốc phòng.

Mật mã hóa khóa công khai (bất đối xứng) cho phép người sử dụng trao đổi các thông tin mật mà không cần phải trao đổi các khóa chung bí mật trước đó. Điều này được thực hiện bằng cách sử dụng một cặp khóa có quan hệ toán học với nhau là khóa công khai và khóa cá nhân (hay khóa bí mật). Trong mật mã hóa khóa công khai, khóa cá nhân phải được giữ bí mật trong khi khóa công khai được phổ biến công khai. Trong 2 khóa, một dùng để mã hóa và khóa còn lại dùng để giải mã. Điều quan trọng đối với hệ thống là không thể tìm ra khóa bí mật nếu chỉ biết khóa công khai.

Quan hệ giữa khóa công khai và khóa bí mật có tính chất một chiều, có nghĩa là từ việc sở hữu khóa bí mật ta dễ dàng tìm ra được khóa công khai, tuy nhiên cho dù biết được khóa công khai thì việc tìm ra khóa bí mật ra rất khó (khó theo nghĩa khối lượng và thời gian tính toán được yêu cầu là rất lớn, lên tới hàng trăm năm). Cơ sở toán học của tính chất một chiều này là nhờ vào các hàm toán học một chiều (one-way function). Một ví dụ về hàm dạng này là bài toán phân tích thừa số nguyên tố : giả sử ta có một vài số nguyên tố lớn, từ các số này ta tìm tích của chúng thì đơn giản ; ngược lại từ tích số, việc phân tích ra các thừa số nguyên tố này là khó. Nếu các số này là lớn, cỡ 1024 hoặc 2024 bit thì việc phân tích thừa số nguyên tố này có thể đòi hỏi máy tính năng lực lớn nhất bây giờ chạy mất vài chục năm. Thời gian này là quá lớn đối với nhu cầu cần giữa bí mật của một khóa bí mật trong vài năm.

1.2 Lịch sử ra đời RSA

Thuật toán được Ron Rivest, Adi Shamir và Len Adleman mô tả lần đầu tiên vào năm 1977 tại Học viện Công nghệ Massachusetts (MIT). Tên của thuật toán lấy từ 3 chữ cái đầu của tên 3 tác giả. Trước đó, vào năm 1973, Clifford Cocks, một nhà toán học người Anh làm việc tại GCHQ, đã mô tả một thuật toán tương tự. Với khả năng tính toán tại thời điểm đó thì thuật toán này không khả thi và chưa bao giờ được thực nghiệm. Tuy nhiên, phát minh này chỉ được công bố vào năm 1997 vì được xếp vào loại tuyệt mật.

Thuật toán RSA được MIT đăng ký bằng sáng chế tại Hoa Kỳ vào năm 1983 (Số đăng ký 4,405,829). Bằng sáng chế này hết hạn vào ngày 21 tháng 9 năm 2000. Tuy nhiên, do thuật toán đã được công bố trước khi có đăng ký bảo hộ nên sự bảo hộ hầu như không có giá trị bên ngoài Hoa Kỳ. Ngoài ra, nếu như công trình của Clifford Cocks đã được công bố trước đó thì bằng sáng chế RSA đã không thể được đăng ký.

1.3 Tiêu chuẩn về RSA

1.3.1 PKCS#1. Ver1.0-2.2. RSA Cryptography Standard

PKCS (Tiêu chuẩn mật mã khóa công khai - Public-Key Cryptography Standard) - do Phòng thí nghiệm RSA (Mỹ) ban hành - là một tập hợp các tiêu chuẩn để hoàn thiện hệ mật mã khóa công khai, được phát triển vào năm 1991

Trong đó, PKCS #1 là một trong những tiêu chuẩn được sử dụng nhiều nhất (thực tế) cho việc sử dụng RSA trong thực tế

Một nâng cấp lớn cho PKCS # 1, từ phiên bản 1.0 đến phiên bản 2.0 là giới thiệu các chế độ bổ sung với đối số bảo mật mạnh hơn và RSA đa nguyên tố.

Phiên bản 2.2 cập nhật thêm danh sách các thuật toán băm như SHA-224,SHA-512/224 và SHA-512/256.

1.3.2 TCVN 7635:2007. Tiêu chuẩn mật mã – Chữ ký số

TCVN 7635:2007 do Tiểu Ban kỹ thuật tiêu chuẩn TCVN/JTC 1/SC 27 "*Các kỹ thuật mật mã*" biên soạn trên cơ sở dự thảo đề nghị của Ban cơ yếu Chính phủ, Tổng cục Tiêu chuẩn Đo lường Chất lượng đề nghị, Bộ Khoa học và Công nghệ công bố.

Tiêu chuẩn này áp dụng cho các chữ ký số sử dụng trong hoạt động giao dịch điện tử của mọi tổ chức, công dân Việt Nam và tổ chức, công dân nước ngoài có quan hệ kinh tế - xã hội với tổ chức, công dân Việt Nam.

Tiêu chuẩn này quy định 3 thành phần cần thiết cho lược đồ chữ ký số:

- Thành phần thứ nhất là thuật toán chữ ký số RSA-PSS
- Thành phần thứ hai là thuật toán hàm băm SHA-256
- Thành phần cuối cùng là thuật toán số giả ngẫu nhiên dùng AES-128

1.4 Hoạt động của thuật toán RSA

Thuật toán RSA có hai khóa: khóa công khai (hay khóa công cộng) và khóa bí mật (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để mã hóa. Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

Ta có thể mô phỏng trực quan một hệ mật mã khoá công khai như sau : Bob muốn gửi cho Alice một thông tin mật mà Bob muốn duy nhất Alice có thể đọc được. Để làm được điều này, Alice gửi cho Bob một chiếc hộp có khóa đã mở sẵn và giữ lại chìa khóa. Bob nhận chiếc hộp, cho vào đó một tờ giấy viết thư bình thường và khóa lại (như loại khoá thông thường chỉ cần sập chốt lại, sau khi sập chốt khóa ngay cả Bob cũng không thể mở lại được-không đọc lại hay sửa thông tin trong thư được nữa). Sau đó Bob gửi chiếc hộp lại cho Alice. Alice mở hộp với chìa khóa của mình và đọc thông tin trong thư. Trong ví dụ này, chiếc hộp với khóa mở đóng vai trò khóa công khai, chiếc chìa khóa chính là khóa bí mật.

1.5 Vấn đề an toàn của RSA

Độ an toàn của hệ thống RSA dựa trên hai vấn đề của toán học: bài toán phân tích ra thừa số nguyên tố các số nguyên lớn và bài toán RSA. Nếu hai bài toán trên là khó (không tìm được thuật toán hiệu quả để giải chúng) thì không thể thực hiện được việc phá mã toàn bộ đối với RSA.

Bài toán RSA là bài toán tính căn bậc e theo mô-đun n (với n là hợp số), tức là tìm số m sao cho $m^e = c \pmod n$, trong đó (n, e) chính là khóa công khai và c là bản mã. Hiện nay phương pháp triển vọng nhất giải bài toán này là phân tích n ra

thừa số nguyên tố. Khi thực hiện được điều này, kẻ tấn công sẽ tìm ra số mũ bí mật d từ khóa công khai và có thể giải mã theo đúng quy trình của thuật toán. Nếu kẻ tấn công tìm được 2 số nguyên tố p và q sao cho: n = pq thì có thể dễ dàng tìm được giá trị (p - 1) . (q - 1) và qua đó xác định d từ e . Chưa có một phương pháp nào được tìm ra trên máy tính để giải bài toán này trong thời gian đa thức (polynomial-time). Tuy nhiên người ta cũng chưa chứng minh được điều ngược lại (sự không tồn tại của thuật toán).

Tại thời điểm năm 2005, số lớn nhất có thể được phân tích ra thừa số nguyên tố có độ dài 663 bit với phương pháp phân tán trong khi khóa của RSA hiện nay thường có độ dài từ 1024 tới 2048 bit. Một số chuyên gia cho rằng khóa 1024 bit có thể sớm bị phá vỡ (cũng có nhiều người phản đối việc này). Với khóa 4096 bit thì hầu như không có khả năng bị phá vỡ trong tương lai gần. Do đó, người ta thường cho rằng RSA đảm bảo an toàn với điều kiện n được chọn đủ lớn. Nếu n có độ dài 256 bit hoặc ngắn hơn, nó có thể bị phân tích trong vài giờ với máy tính cá nhân dùng các phần mềm có sẵn. Nếu n có độ dài 512 bit, nó có thể bị phân tích bởi vài trăm máy tính tại thời điểm năm 1999. Một thiết bị lý thuyết có tên là TWIRL do Shamir và Tromer mô tả năm 2003 đã đặt ra câu hỏi về độ an toàn của khóa 1024 bit. Vì vậy hiện nay người ta khuyến cáo sử dụng khóa có độ dài tối thiểu 2048 bit.

Năm 1993, Peter Shor công bố thuật toán Shor chỉ ra rằng: máy tính lượng tử (trên lý thuyết) có thể giải bài toán phân tích ra thừa số trong thời gian đa thức. Tuy nhiên, máy tính lượng tử vẫn chưa thể phát triển được tới mức độ này trong nhiều năm nữa.

1.6 Kết luận chương tổng quan

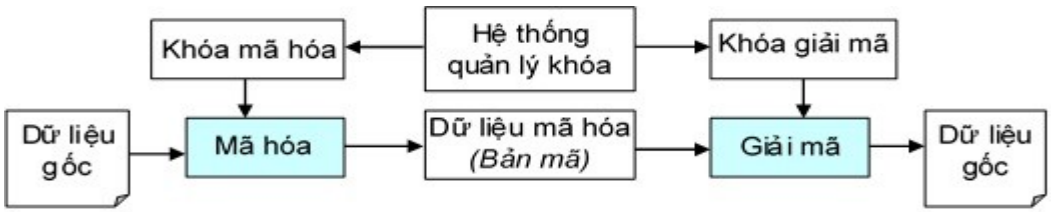
RSA là hệ mật khóa công khai được sử dụng phổ biến hàng đầu hiện nay. Trong phạm vi chương này, ta đã nhắc lại những nét khái quát về hệ mật đã nêu. Qua đây có thể thấy rằng để đảm bảo an toàn cho hệ mật thì không thể áp dụng trực tiếp lược đồ cơ bản của RSA vào thực tế, mà cần phải sử dụng các thao tác chuyển đổi bản rõ trước khi mã hóa (và trước khi kí). Có một số lược đồ chuyển đổi bản rõ đã được đề xuất cho RSA, trong số đó, có lược đồ đã được chứng minh là kém an toàn và đã được chỉnh sửa hoặc thay thế bằng lược đồ khác. Như vậy, lược đồ chuyển đổi bản rõ là một nhân tố quan trọng đóng góp vào tính an toàn của RSA khi sử dụng thực tế. Do đó, trong các phần tiếp theo của đồ án này, ta sẽ xem xét một số lược đồ như thế.

CHƯƠNG II. KHÓA RSA

2.1.Một số khái niệm

Mã hóa dữ liệu là sử dụng một phương pháp biến đổi dữ liệu từ dạng bình thường sang một dạng khác, mà một người không có thẩm quyền, không có phương tiện giải mã thì không thể đọc hiểu được.

Giải mã dữ liệu là quá trình ngược lại, là việc sử dụng một phương pháp biến đổi dữ liệu đã được mã hóa về dạng thông tin ban đầu. Có thể mô tả quy trình thực hiện mã hóa dữ liệu và giải mã dữ liệu như sau:



Quy trình giải mã dữ liệu

Sau đây là một số khái niệm và kí hiệu liên quan về vấn đề mã hóa dữ liệu :

- E (Encryption - Mã hóa): Là quá trình chuyển đổi dữ liệu gốc thành dữ liệu được mã hóa sao người khác không thể đọc hiểu.

-D (Decryption - Giải mã): Là quá trình ngược lại của mã hóa, biến đổi dữ liệu đã được mã hóa thành dạng gốc ban đầu.

- M (Message - Thông điệp), bản gốc hay bản rõ (Plaintext): Là tệp dữ liệu chưa được mã hóa hoặc đã được giải mã.

- C (Ciphertext - Bản mã): Tệp dữ liệu đã được mã hóa.

- K (Key - Khóa): Là dãy các bit 0, 1 thường được đưa ra dạng xâu ký tự, số...

- K_E: Là khóa dùng để mã hóa.

- K_D: Là khóa dùng để giải mã.

Theo quy ước, khi mã hóa thì C = E(M) và khi giải mã thì M = D(C) = D(E(M)).

Theo phương pháp truyền thống, người ta thường dùng cùng một khóa để mã hóa và giải mã. Lúc đó khóa phải được giữ bí mật tuyệt đối. Người ta gọi đây là hệ thống mã hóa cổ điển (các tên gọi khác: đối xứng, một khóa, khóa bí mật...). Một phương pháp khác, sử dụng khóa công khai (còn gọi là phương pháp mã hóa bất đối xứng, hay hệ thống hai khóa) trong đó khóa để mã hóa và khóa để giải mã là khác nhau. Các khóa này tạo thành một cặp chuyển đổi ngược nhau và không khóa nào có thể suy ra được từ khóa kia. Quy trình mã hóa khóa công khai gồm có các bước cơ bản như sau:

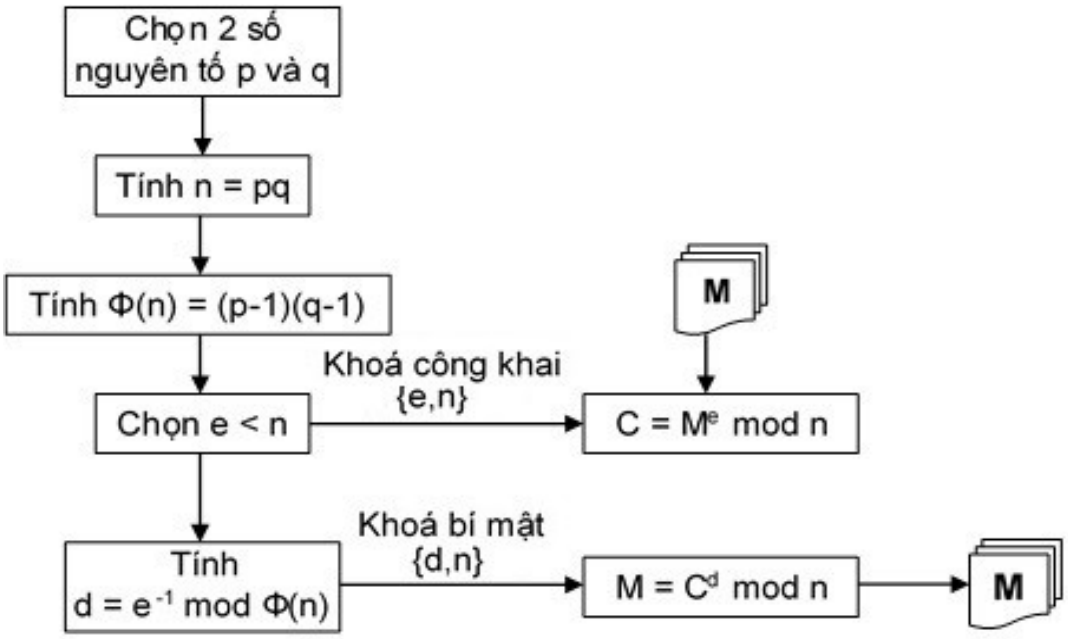
- Mỗi một hệ thống đầu cuối tạo một cặp khóa dùng cho quá trình mã hóa và giải mã mà hệ thống đó sẽ nhận.

- Mỗi hệ thống công bố khóa mã hóa của mình, gọi là khóa công khai, khóa còn lại gọi là khóa bí mật (khóa riêng) và phải được giữ an toàn.

- Nếu Bên gửi muốn gửi thông điệp cho Bên nhận, Bên gửi mã hóa thông điệp sử dụng khóa công khai của Bên nhận.

- Khi Bên nhận nhận thông điệp, Bên nhận giải mã bằng khóa riêng của Bên nhận.

Sơ đồ dưới đây minh họa các bước trong thuật toán RSA.



Sơ đồ biểu diễn thuật toán mã hóa RSA

Thuật toán RSA được thiết kế dựa trên độ khó của bài toán phân tích ra thừa số nguyên tố trên tập số nguyên Z_n.

Cho số nguyên dương n = p * q, với p, q là 2 số nguyên tố rất lớn (ít nhất 100 ký số). Khi biết *n*, muốn tìm *p, q* thì phải giải bài toán phân tích ra thừa số nguyên tố, công việc này đòi hỏi phải thực hiện một số lượng các phép tính vô cùng lớn.

2.2. Tạo khóa

Giả sử Alice và Bob cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Internet). Với thuật toán RSA, Alice đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau:

1. Chọn 2 số nguyên tố lớn p và q với p q ≠ , lựa chọn ngẫu nhiên và độc lập.

2. Tính: n p q

3. Tính giá trị hàm số Ơle của n : φ(n) =(p-1).(q-1)

4. Chọn một số tự nhiên e sao cho 1< e < φ(n) và là số nguyên tố cùng nhau với φ (n).

5. Tính: d sao cho d.e ≡1 modφ(n).

Một số lưu ý:

- Các số nguyên tố thường được chọn bằng phương pháp thử xác suất.

- Các bước 4 và 5 có thể được thực hiện bằng giải thuật Euclid mở rộng
- Đối với bước 3, PKCS #1 v2.1 sử dụng λ = LCM (p-1,q-1) thay cho φ(n) = (p-1)(q-1) , trong đó LCM (Least Common Multiple) là hàm tìm bội số chung nhỏ nhất.

Khi đó, khóa công khai là bộ (n e,) , với e được gọi là số mũ công khai, hay số mũ mã hóa; còn khóa bí mật là bộ (n d,) , với d được gọi là số mũ bí mật, hay số mũ giải mã. Một dạng khác của khóa bí mật là bộ bao gồm:

- p và q là hai số nguyên tố chọn ban đầu
- d mod (p-1) và d mod (q-1) thường được gọi là dP và dQ
- $q^{-1} \bmod p$ thường được gọi là qInv
- $e.dP \equiv 1 \bmod (p-1)$, $e.dQ \equiv 1 \bmod (q-1)$

Dạng khóa bí mật này cho phép thực hiện giải mã và ký nhanh hơn với việc sử dụng định lý số dư Trung Quốc. Ở dạng này, tất cả thành phần của khóa bí mật phải được giữ bí mật. Alice gửi khóa công khai cho Bob, và giữ bí mật khóa cá nhân của mình. Ở đây, p và q giữ vai trò rất quan trọng. Chúng là các phân tố của n và cho phép tính d khi biết e . Nếu không sử dụng dạng sau của khóa bí mật (dạng CRT) thì p và q sẽ được xóa ngay sau khi thực hiện xong quá trình tạo khóa.

2.3.Mã hóa

Giả sử Bob muốn gửi thông điệp M cho Alice. Nói chung, thông điệp M là một số nguyên m thỏa mãn 0 ≤ < m n . Lúc này Bob có m và biết n cũng như e do Alice gửi. Bob sẽ tính c là bản mã hóa của m theo công thức:

$$c = m^e \bmod n$$

Hàm trên có thể tính dễ dàng sử dụng phương pháp tính hàm mũ (theo môđun) bằng thuật toán bình phương và nhân. Cuối cùng Bob gửi c cho Alice.

2.4.Giải mã

Alice nhận c từ Bob và biết khóa bí mật d . Alice có thể tìm được m từ c theo công thức sau:

$$m = c^d \bmod n$$

Tính đúng đắn của công thức giải mã được giải thích như sau:

$$c^d = (m^e)^d = m^{ed} \bmod n$$

- Do $ed \equiv 1 \bmod (p-1)$ và $ed \equiv 1 \bmod (q-1)$ theo định lí fermat nhỏ nên:

$$m^{ed} \equiv m(mod\, p) \text{ và } m^{ed} \equiv m(mod\, q)$$

- Do p và q là hai số nguyên tố cùng nhau nên áp dụng định lý số dư Trung Quốc ta được:

$$m^{ed} \equiv m(mod\, pq) \text{ tức là } m^{ed} \equiv m(mod\, n)$$

2.5. Yêu cầu đối với khóa RSA

2.5.1. Các yêu cầu chung

- Cặp khóa RSA dùng để ký thì không được dùng cho mục đích khác (chẳng hạn dùng lại để mã thông điệp).

-Hai số nguyên tố *p*, *q* và số mũ bí mật *d* cần phải được giữ bí mật tránh việc bị truy cập bất hợp pháp, làm lộ hoặc sửa đổi. Modulo *n* và số mũ công khai *e* phải được công bố công khai.

-Mỗi người sử dụng cần có Modulo *n* riêng

-Độ dài của Modulo *n* (*nlen*) không được nhỏ hơn 2048 bit và nên được thay đổi theo thời gian như sau.

Năm	Độ an toàn	nlen tối thiểu
Năm < 2020	112	2048
	128	3072

Trong đó, độ mạnh về an toàn (*security_strength*) là một số nguyên biểu thị lượng tính toán cần thiết đề phá hệ mã.

Vì các phương pháp phá hệ mã thường xuyên được hoàn thiện nên cần phải định kỳ 3 đến 5 năm một lần xem xét lại *nlen* tối thiểu (có thể tham khảo chi tiết yêu cầu này trong tài liệu *NIST Special Publication 800-57: Recommendation for Key Management - Part1: General, January 2016*).

- Phiên bản áp dụng: Áp dụng phiên bản 2.1 của tiêu chuẩn RSA Cryptography Standard PKCS #1 v2.1.

- Áp dụng lược đồ RSAES-OAEP để mã hóa và RSASSA-PSS để ký.

2.5.2. Yêu cầu chi tiết cho khóa RSA

1) Số mũ công khai *e* cần phải được chọn với các ràng buộc sau:
   a) Số mũ công khai *e* cần được chọn trước khi tạo số mũ bí mật *d*;
   b) Số mũ công khai *e* cần phải là số nguyên dương lẻ sao cho

$$65{,}537 \le e < 2^{nlen-2security\_strength}$$

Với *nlen* là độ dài của modulo *n* theo bit.

Chú ý rằng *e* có thể là giá trị bất kỳ mà thỏa mãn ràng buộc 1(b); *p* và *q* sẽ được chọn (trong mục 2) sao cho *e* là nguyên tố cùng nhau với cả (*p* - 1) và (*q* - 1).

2) Hai số nguyên tố *p* và q được tạo ngẫu nhiên và giữ bí mật cần phải được chọn với các ràng buộc sau:
   a) (*p* - 1) và (*q* - 1) cần phải là nguyên tố cùng nhau với số mũ công khai *e*;
   b) Mỗi một trong bốn số (*p* + 1), (*p* - 1) và (*q* + 1), (*q* - 1) cần phải có các nhân tử nguyên tố lớn hơn $2^{security\text{-}strength+20}$;
   c) Nhân tử nguyên tố bí mật *p, q* cần phải được chọn ngẫu nhiên từ các số nguyên tố thỏa mãn $(\sqrt{2})(2^{(nlen/2)-1}) \leq q < p \leq (2^{(nlen/2)} - 1)$;
   d) $|p - q| > 2^{(nlen/2-100)}$.
3) Số mũ bí mật *d* cần phải được lựa chọn sau khi tạo *p* và *q* với các ràng buộc:
   a) Số mũ *d* cần phải lớn hơn $2^{(nlen/2)}$
   b) $d = e^{-1} mod(LCM((p - 1), (q - 1)))$
   c) (Chi tiết về hàm tạo các tham số RSA có thể tham khảo trong tài liệu *FIPS 186-4: Digital Signature Standard)*.

2.6. Các vấn đề trong thực tế

2.6.1. Quá trình tạo khóa

Việc tìm ra 2 số nguyên tố đủ lớn p và q thường được thực hiện bằng cách thử xác suất các số ngẫu nhiên có độ lớn phù hợp (dùng phép kiểm tra nguyên tố cho phép loại bỏ hầu hết các hợp số).

Hai số p và q còn cần được chọn không quá gần nhau để phòng trường hợp phân tích n bằng phương pháp phân tích Fermat. Ngoài ra, nếu $(p - 1)$ và $(q - 1)$ có thừa số nguyên tố nhỏ thì n cũng có thể dễ dàng bị phân tích và vì thế p và q cũng cần được thử để tránh khả năng này.

Bên cạnh đó, cần tránh sử dụng các phương pháp tìm số ngẫu nhiên mà kẻ tấn công có thể lợi dụng để biết thêm thông tin về việc lựa chọn (cần dùng các bộ tạo số ngẫu nhiên tốt). Yêu cầu ở đây là các số được lựa chọn cần đồng thời ngẫu nhiên và không dự đoán được. Đây là các yêu cầu khác nhau: một số có thể được lựa chọn ngẫu nhiên (không có kiểu mẫu trong kết quả) nhưng nếu có thể dự đoán được dù chỉ một phần thì an ninh của thuật toán cũng không được đảm bảo. Một ví dụ là bảng các số ngẫu nhiên do tập đoàn Rand xuất bản vào những năm 1950 có thể rất thực sự ngẫu nhiên nhưng kẻ tấn công cũng có bảng này. Nếu kẻ tấn công 6 đoán được một nửa chữ số của p hay q thì chúng có thể dễ dàng tìm ra nửa còn lại (theo nghiên cứu của Donald Coppersmith vào năm 1997)

Một điểm nữa cần nhấn mạnh là khóa bí mật d phải đủ lớn. Năm 1990, Wiener chỉ ra rằng nếu giá trị của p nằm trong khoảng q và 2q (khá phổ biến) và 1/4 d n < / 3 thì có thể tìm ra được d từ n và e .

Mặc dù e đã từng có giá trị là 3 nhưng hiện nay các số mũ nhỏ không còn được sử dụng do có thể tạo nên những lỗ hổng. Giá trị thường dùng hiện nay là 65537 vì được xem là đủ lớn và cũng không quá lớn ảnh hưởng tới việc thực hiện hàm mũ.

Tốc độ RSA có tốc độ thực hiện chậm hơn đáng kể so với DES và các thuật toán mã hóa đối xứng khác. Trên thực tế, Bob sử dụng một thuật toán mã hóa đối xứng nào đó để mã hóa văn bản cần gửi và chỉ sử dụng RSA để mã hóa khóa để giải mã (thông thường khóa ngắn hơn nhiều so với văn bản). Phương thức này cũng tạo ra những vấn đề an ninh mới. Một ví dụ là cần phải tạo ra khóa đối xứng thật sự ngẫu nhiên. Nếu không, kẻ tấn công sẽ bỏ qua RSA và tập trung vào việc đoán khóa đối xứng.

2.6.2. Phân phối khóa

Cũng giống như các thuật toán mã hóa khác, cách thức phân phối khóa công khai là một trong những yếu tố quyết định đối với độ an toàn của RSA. Quá trình phân phối khóa cần chống lại được tấn công đứng giữa (man-in-the-middle attack). Giả sử Eve có thể gửi cho Bob một khóa bất kỳ và khiến Bob tin rằng đó là khóa (công khai) của Alice. Đồng thời Eve có khả năng đọc được thông tin trao đổi giữa Bob và Alice. Khi đó, Eve sẽ gửi cho Bob khóa công khai của chính mình (mà Bob nghĩ rằng đó là khóa của Alice). Sau đó, Eve đọc tất cả văn bản mã hóa do Bob gửi, giải mã với khóa bí mật của mình, giữ một bản sao đồng thời mã hóa bằng khóa công khai của Alice và gửi cho Alice. Về nguyên tắc, cả Bob và Alice đều không phát hiện ra sự can thiệp của người thứ ba. Các phương pháp chống lại dạng tấn công này thường dựa trên các chứng thực khóa công khai (digital certificate) hoặc các thành phần của hạ tầng khóa công khai (public key infrastructure - PKI).

2.6.3. Tấn công dựa trên thời gian

Vào năm 1995, Paul Kocher mô tả một dạng tấn công mới lên RSA: nếu kẻ tấn công nắm đủ thông tin về phần cứng thực hiện mã hóa và xác định được thời gian giải mã đối với một số bản mã lựa chọn thì có thể nhanh chóng tìm ra khóa d . Dạng tấn công này có thể áp dụng đối với hệ thống chữ ký điện tử sử dụng RSA.

Năm 2003, Dan Boneh và David Brumley chứng minh một dạng tấn công thực tế hơn: phân tích thừa số RSA dùng mạng máy tính (Máy chủ web dùng SSL). Tấn công đã khai thác thông tin rò rỉ của việc tối ưu hóa định lý số dư Trung quốc mà nhiều ứng dụng đã thực hiện.

Để chống lại tấn công dựa trên thời gian là đảm bảo quá trình giải mã luôn diễn ra trong thời gian không đổi bất kể văn bản mã. Tuy nhiên, cách này có thể làm giảm hiệu suất tính toán. Thay vào đó, hầu hết các ứng dụng RSA sử dụng một kỹ thuật gọi là che mắt. Kỹ thuật này dựa trên tính nhân của RSA: thay vì tính $c^d \bmod n$, Alice đầu tiên chọn một số ngẫu nhiên r và tính $(r^e c)^d \bmod n$.

Kết quả của phép tính này là $rm \pmod n$ và tác động của r sẽ được loại bỏ bằng cách nhân kết quả với nghịch đảo của r . Đối với mỗi văn bản mã, người ta chọn một giá trị của r . Vì vậy, thời gian giải mã sẽ không còn phụ thuộc vào giá trị của văn bản mã.

CHƯƠNG III. CÁC PHÉP BIẾN ĐỔI CƠ SỞ

- Mật mã nguyên thủy là các phép toán cơ bản trên đó các lược đồ mật mã có thể được xây dựng. Chúng được dành cho triển khai trong phần cứng hoặc dưới dạng mô-đun phần mềm, và không nhằm cung cấp bảo mật ngoài một chương trình.

- Bốn loại primitive được chỉ định trình bày dưới đây, được tổ chức theo các cặp: mã hóa và giải mã; và chữ ký và xác minh.

3.1. Chuyển đổi dữ liệu nguyên thủy

- Có 2 kiểu chuyển đổi dữ liệu nguyên thủy được trình bày trong tài liệu này là
  - I2OSP - Integer-to-Octet-String primitive
  - OS2IP - Octet-String-to-Integer primitive
- Một chuỗi octet là một chuỗi các octet có thứ tự (byte tám bit). Chuỗi được lập chỉ mục từ đầu tiên (theo quy ước, ngoài cùng bên trái) đến cuối cùng (ngoài cùng bên phải). Đối với mục đích chuyển đổi sang và từ số nguyên, octet đầu tiên được coi là quan trọng nhất trong các nguyên thủy chuyển đổi sau.

3.1.1 Hàm cơ sở chuyển đổi từ dạng số nguyên sang một chuỗi octet I2OSP (x, xLen)

- I2OSP: Integer To Octet String Primitive:X = I2OSP(x, xLen), x≥0
  - Chức năng: Chuyển đổi đổi số nguyên không âm x thành một chuỗi octet có độ dài xLen
  - Đầu vào: x Số nguyên không âm cần chuyển đổi
  - Đầu ra: X Chuỗi octet tương ứng có độ dài XLen
  - Các bước biến đổi.

    Bước 1: Kiểm tra x nếu x >256^xLen, thông báo"số nguyên quá lớn" và dừng

    Bước 2: Viết số nguyên x dưới dạng biểu diễn chữ số xLen duy nhất của nó trong cơ số 256:

$$x = x_{xLen-1} \cdot 256^{xLen-1} + x_{xLen-2} \cdot 256^{xLen-2} + \ldots + x_1 \cdot 256 + x_0$$

Trong đó 0 <= xi <256 (lưu ý rằng một hoặc nhiều chữ số đứng đầu sẽ bằng 0 nếu x nhỏ hơn 256 ^ (xLen-1))

Bước 3: Lấy octet Xi có giá trị nguyên x_(xLen-i) với 1 <= i <= xLen. Xuất chuỗi octet.

- Ví dụ:

$X = \text{I2OSP}(581.579.775,\ 6)$

$x = 581.579.775$

$= 00_h \cdot 256^5 + 00_h \cdot 256^4 + 22_h \cdot 256^3 + AA_h \cdot 256^2 + 33_h \cdot 256 + FF_h$

$X = 000022AA33FF$

3.1.2. Hàm cơ sở chuyển đổi từ dạng chuỗi octet về dạng số nguyên OS2IP(X)

- OS2IP: Octet String To Integer Primitive: **x = OS2IP(X)**
  - Chức năng: Chuyển chuỗi octet thành một số nguyên không âm
  - Đầu vào: X Chuỗi octet cần chuyển đổi
  - Đầu ra: x Số nguyên không âm tương ứng.
  - Các bước biến đổi.

    Bước 1: Gọi X_1 X_2 ... X_xLen là các octet của X từ đầu tiên đến cuối cùng và đặt x_ (xLen-i) là giá trị nguyên của octet X_i cho 1 <= i <= xLen

    Bước 2: Cho x = x_ (xLen-1) 256 ^ (xLen-1) + x_ (xLen-2) 256 ^ (xLen-2) + ... + x_1 256 + x_0.

    Bước3: Đầu ra x.

- Ví dụ:

$X = 22AA33FF$

$x = 22_h \cdot 256^3 + AA_h \cdot 256^2 + 33_h \cdot 256 + FF_h$

$= 34 \cdot 256^3 + 170 \cdot 256^2 + 51 \cdot 256 + 255 = 581.579.775$

3.2. Các phép toán cơ sở mã hóa và giải mã nguyên thủy RSAEP và RSADP

- Nguyên tắc mã hóa tạo ra đại diện bản mã từ đại diện thông báo dưới sự kiểm soát của khóa công khai và nguyên thủy giải mã khôi phục đại diện thông báo từ đại diện bản mã dưới sự kiểm soát của khóa cá nhân tương ứng.

- Một cặp mã hóa và giải mã nguyên thủy được sử dụng trong các lược đồ mã hóa được định nghĩa ở đây: RSAEP / RSADP. RSAEP và RSADP liên quan đến cùng một phép toán, với các khóa khác nhau làm đầu vào.
- Các kiểu nguyên thủy được xác định ở đây giống như IFEP-RSA / IFDP-RSA trong IEEE Std 1363-2000 (ngoại trừ hỗ trợ cho RSA đa nguyên tố đã được thêm vào) và tương thích với PKCS # 1 v1.5.
- Phép toán chính trong mỗi nguyên thủy là lũy thừa.

3.2.1. Phép mã hóa: c = RSAEP ((n, e), m)

- Đầu vào (n, e): là khóa công khai của RSA

  m: là bản rõ dạng số nguyên từ 0 đến n-1
- Đầu ra: c: là bản mã dạng số nguyên từ 0 đến n-1
- Các bước

  Bước 1: Kiểm tra m nếu bản rõ m không nằm trong 0 đến n-1 thì báo lỗi "bản rõ m nằm ngoài phạm vi" và dừng mã hóa.

  Bước 2: Thực hiện c= m^e mod n

  Bước 3: Output c

3.2.2. Phép giải mã: m = RSADP (K_S, c)

- Đầu vào: K_S: là khóa bí mật của RSA

  c: là bản mã dạng số nguyên từ 0 đến n-1
- Đầu ra: m: là bản rõ dạng số nguyên từ 0 đến n-1
- Các bước:

  Bước 1: Kiểm tra c nếu bản mã c không nằm trong 0 đến n-1 thì báo lỗi "bản mã c nằm ngoài phạm vi" và dừng giải mã.

  Bước 2:

  TH1: K S thuộc cặp (n, d) thì thực hiện giải mã m = c^d mod n

  TH2: K_S thuộc (p, q, dP, dQ, qInv) và (r_i, d_i, t_i)

  i. Lấy m_1= c^dP mod p, m_2= c^dQ mod q

  ii. Nếu u > 2, lấy m_i = c^(d_i) mod r_i với i= 3, 4, …, u

  iii. h = (m_1 - m_2) * qInv mod p

iv. m = m_2 + q*h

v. Nếu u > 2, lấy R= r_1 và for i=3 to u do

1. R = R*r_(i-1)
2. h = (m_i - m) * t_i mod r_i
3. m = m + R*h

Bước 3: Output m.

3.3. Chữ ký và kiểm tra chữ ký

- Một chữ ký nguyên thủ tạo ra đại diện chữ ký từ một thông điệp dưới sự điều khiển của private key và bản gốc xác minh khôi phục đại diện thông điệp từ đại diện chữ ký dưới sự kiểm soát của public key tương ứng. Một cặp chữ ký và kiểm tra nguyên thủy được sử dụng trong các lược đồ chữ ký được xác định trong tài liệu này và được chỉ định ở đây: RSASP1 / RSAVP1.

3.3.1. Phép ký RSASP1(K_S, m)

- Đầu vào: K_S: Là khóa bí mật của RSA

  m: Là bản rõ dạng số nguyên từ 0 đến n-1
- Đầu ra: s: là chữ ký dạng số nguyên từ 0 đến n-1
- Các bước

  Bước 1: Kiểu tra nếu bản rõ m không nằm trong 0 đến n-1 thì báo lỗi "bản rõ m nằm ngoài phạm vi" và dừng.

  Bước 2:

  TH1: K_S thuộc cặp (n, d) thì thực hiện giải mã s = m^d mod n

  TH2: K_S thuộc (p, q, dP, dQ, qInv) và (r_i, d_i, t_i)

  i. Lấy s_1= m^dP mod p, s_2= m^dQ mod q

  ii. Nếu u > 2, lấy s_i = m^(d_i) mod r_i với i= 3, 4, …, u

  iii. h = (s_1 - s_2) * qInv mod p

  iv. s = s_2 + q*h

  v. Nếu u > 2, lấy R= r_1 và for i=3 to u do

  1. R = R*r_(i-1)

2. h = (s_i - s) * t_i mod r_i

3. s = s + R*h

Bước 3: Output s

3.3.2. Phép kiểm tra chữ ký m = RSAVP1((n, e), s)

- Đầu vào: (n, e): là khóa công khai của RSA.

  s: là chữ ký dạng số nguyên từ 0 đến n-1

- Đầu ra: m là bản rõ dạng số nguyên từ 0 đến n – 1.
- Các bước:

  Bước 1: Kiểm tra nếu chữ ký s không nằm trong 0 đến n-1 thì báo lỗi "chữ ký s nằm ngoài phạm vi" và dừng.

  Bước 2: Thực hiện m= s^e mod n

  Bước 3: Output m

CHƯƠNG IV. LƯỢC ĐỒ MÃ HÓA

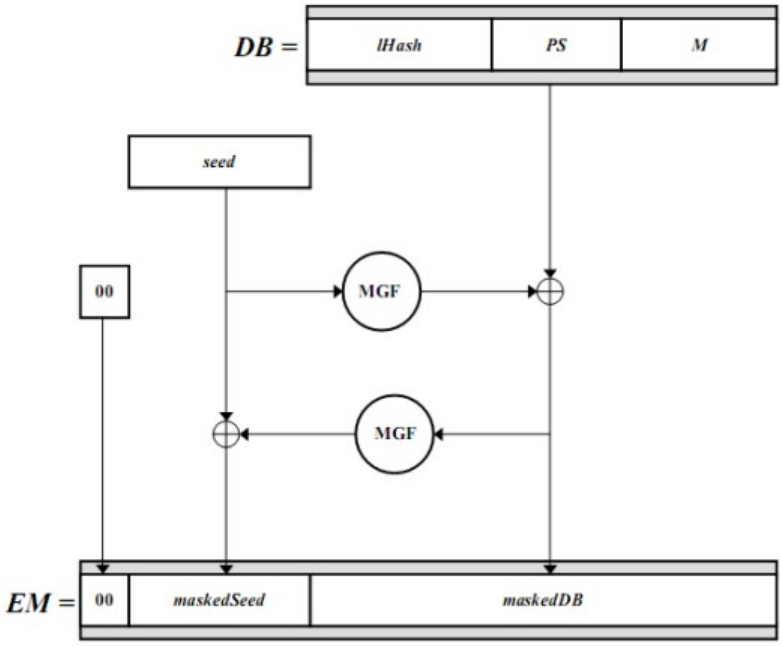
4.1. Lược đồ mã RSA-OAEP

4.1.1. Mô tả chung về lược đồ chuyển đổ bản rõ RSA-OAEP

Lược đồ chuyển đổi bản rõ OAEP (Optimal Asymmetric Encryption Padding) được phát triển bởi Mihir Bellare và Phillip Rogaway vào năm 1994 và được cải tiến bởi Don B. Johnson và Stephen M. Matyas vào năm 1996 [5,7]. Như tên gọi của nó, lược đồ OAEP có thể được áp dụng chung cho các hệ mật bất đối xứng, chứ không riêng gì RSA. Cụ thể, nó có thể được sử dụng cho các hệ mật bất đối xứng khác như ElGamal, Rabin. Đối với hệ mật RSA, kể từ phiên bản 2.0 (PKCS#1 v2.0, RFC2437), lược đồ OAEP được khuyên dùng để thay thế cho lược đồ chuyển đổi bản rõ PKCS#1 v1.5.

Như đã giới thiệu ở phần trên, lược đồ OAEP được đưa vào chuẩn mật mã RSA bắt đầu từ phiên bản 2.0 (PKCS #1 v2.0). Trong phạm vi mục 3.2 này ta sẽ xem xét quy định của PKCS #1 v2.0 về áp dụng lược đồ OAEP trong mã hóa và giải mã (không xem xét việc áp dụng cho kí số).

4.1.2. Lược đồ định dạng dữ liệu EME-OAEP



Hình 1.4. Thuật toán mã hóa EME-OAEP

EM = EME-OAEP-ENCODE(M, L)

- EME = Encoding Method for Encryption
- OAEP = Optimal Asymmetric Encryption Padding
- M = Message, kích thước "bất kì"
- L = Label, có thể là xâu rỗng
- EM = Encoded Message, kích thước bằng k (octet), có tính ngẫu nhiên dù M cố định.
- Lược đồ sử dụng hàm băm Hash() và hàm sinh mặt nạ MGF()
- seed: là một dãy bít ngẫu nhiên có độ dài là k_0

4.1.2.1. Hàm băm

Hàm băm H là hàm tính được “hiệu quả”:

$$H:\{0,1\}^* \rightarrow \{0,1\}^d$$

Nó ánh xạ một xâu độ dài bất kỳ thành một xâu có độ dài cố định gọi là mã băm.

Để an toàn hàm băm H nên thỏa mãn các yêu cầu:

1) *Tính một chiều*: với một mã băm h cho trước, không thể tìm lại được x sao cho:

$$h = H(x)$$

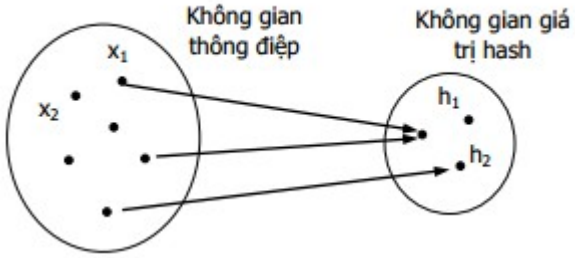
2) *Tính chống trùng yếu*: cho trước một x, không thể tìm y khác x sao cho:

$$H(x) = H(y)$$

3) *Tính chống trùng mạnh*: không thể tìm ra cặp x, y bất kỳ, x khác y sao cho:

$$H(x) = H(y)$$

hay nói cách khác nếu H(x) = H(y) thì có thể chắc chắn rằng x = y



Hình 1: ánh xạ giữa thông điệp và giá trị băm không phải là song ánh

Kích thước của input x là bất kỳ còn kích thước của h là nhỏ, ví dụ giả sử kích thước của x là 512 bit còn kích thước của h là 128 bit. Như vậy trung bình có khoảng 2^{384} giá trị x mà có cùng giá trị h. việc trùng là không thể loại bỏ. Tính chống trùng mạnh của hàm băm là yêu cầu rằng việc tìm ra hai input x như vậy thì phải là rất khó về mặt thời gian tính toán.

4.1.2.2. Hàm sinh mặt nạ MGF

Hàm sinh mặt nạ (MGF: Mask Generation Function) là hàm cho phép sinh một chuỗi octet có độ dài tùy ý từ một chuỗi octet có độ dài bất kì. Có những giới hạn nhất định đối với kích thước của chuỗi đầu vào và chuỗi đầu ra, nhưng thường thì giới hạn đó nằm trong một khoảng rất lớn. Cũng như hàm băm, hàm sinh mặt nạ là hàm tất định, tức là giá trị của chuỗi đầu ra được quyết định hoàn toàn bởi giá trị của chuỗi đầu vào. Chuỗi đầu ra của hàm sinh mặt nạ phải là một chuỗi giả ngẫu nhiên, tức là nếu không biết giá trị đàu vào (mầm, seed) thì không thể phân biệt

được chuỗi đầu ra với một chuỗi ngẫu nhiên thực sự. Chính tính chất ngẫu nhiên của chuỗi đầu ra của hàm sinh mặt nạ quyết định tính chất plaintextawareness của RSAES-OAEP. Ở đây, RSAES-OAEP (RSA Encryption Scheme with OAEP) có nghĩa là mô hình mã hóa bằng RSA có sử dụng lược đồ chuyển đổi bản rõ OAEP. Còn tính chất plaintext-awareness (tạm dịch: phải biết bản rõ) có nghĩa là không thể tạo ra bản mã mà không biết bản rõ [21]. Đối với lược đồ cơ bản của RSA, việc mã hóa được thực hiện bởi công thức

$$c = m^e \mod n$$

còn việc giải mã được thực hiện bởi

$$m = c^d \mod n$$

Trong các công thức này, cả bản rõ m và bản mã c đều là những số nguyên nằm trong khoảng [0;n). Dễ dàng nhận thấy, với một số nguyên c bất kì thuộc khoảng [0;n) , ta luôn có thể "giải mã" nó bằng công thức $m = c^d \mod n$ để tìm ra "bản rõ" m tương ứng. Như vậy, ở đây, một số nguyên c bất kì luôn có thể được coi là bản mã của một bản rõ nào đó, tức là luôn có thể tìm được một bản mã mà không cần biết bản rõ. Vì vậy, lược đồ cơ bản của RSA không có tính chất plaintextawareness.

Nếu hàm sinh mặt nạ đảm bảo được tính ngẫu nhiên cho chuỗi đầu ra thì sẽ giúp lược đồ RSA với OAEP có được tính chất plaintext-awareness. Việc chứng minh tính chất này bằng toán học là rất phức tạp. Ở đây, để hiểu thêm về tính chất đã nêu, ta hãy xét bài toán sau. Cho trước các số nguyên d n, và chuỗi octet pHash. Hãy tìm số nguyên c sao cho kết quả của phép tính $c^d \mod n$ là một số nguyên mà biểu diễn của nó trong hệ cơ số 256 có dạng

$$m = pHash \,||0000...00||01||M$$

Đây chính là bài toán tìm ra một bản mã hợp lệ của lược đồ RSAES-OAEP. Và rõ ràng bài toán này không hề đơn giản. Hàm sinh mặt nạ được khuyên dùng trong PKCS #1 v2.0 gọi là hàm MGF1 và được định nghĩa như sau.

$$MGF1\ (Z, l)$$

Các yếu tố tùy chọn:

- Hash: hàm băm với kích thước đầu ra được kí hiệu là hLen

Đầu vào:

- Z: mầm để sinh mặt nạ, là một chuỗi octet có độ dài bất kì
- l: độ dài mong muốn của mặt nạ, tính bằng octet, tối đa là $2^{32} \cdot$ hLen .

Đầu ra:

- mask: mặt nạ cần sinh có độ dài l; hoặc thông báo lỗi “Mask too long” ("Mặt nạ quá dài").

Thực hiện:

Bước 1: Nếu $l > 2^{32}$. hLen thì đưa ra thông báo lỗi “Mask too long” ("Mặt nạ quá dài") và dừng.

Bước 2: Khởi tạo một chuỗi rỗng T

Bước 3: Cho counter chạy từ 0 đến [l/hLen] – 1

- Sử dụng hàm I2OSP để chuyển đổi counter thành một chuỗi C gồm 4 octet.

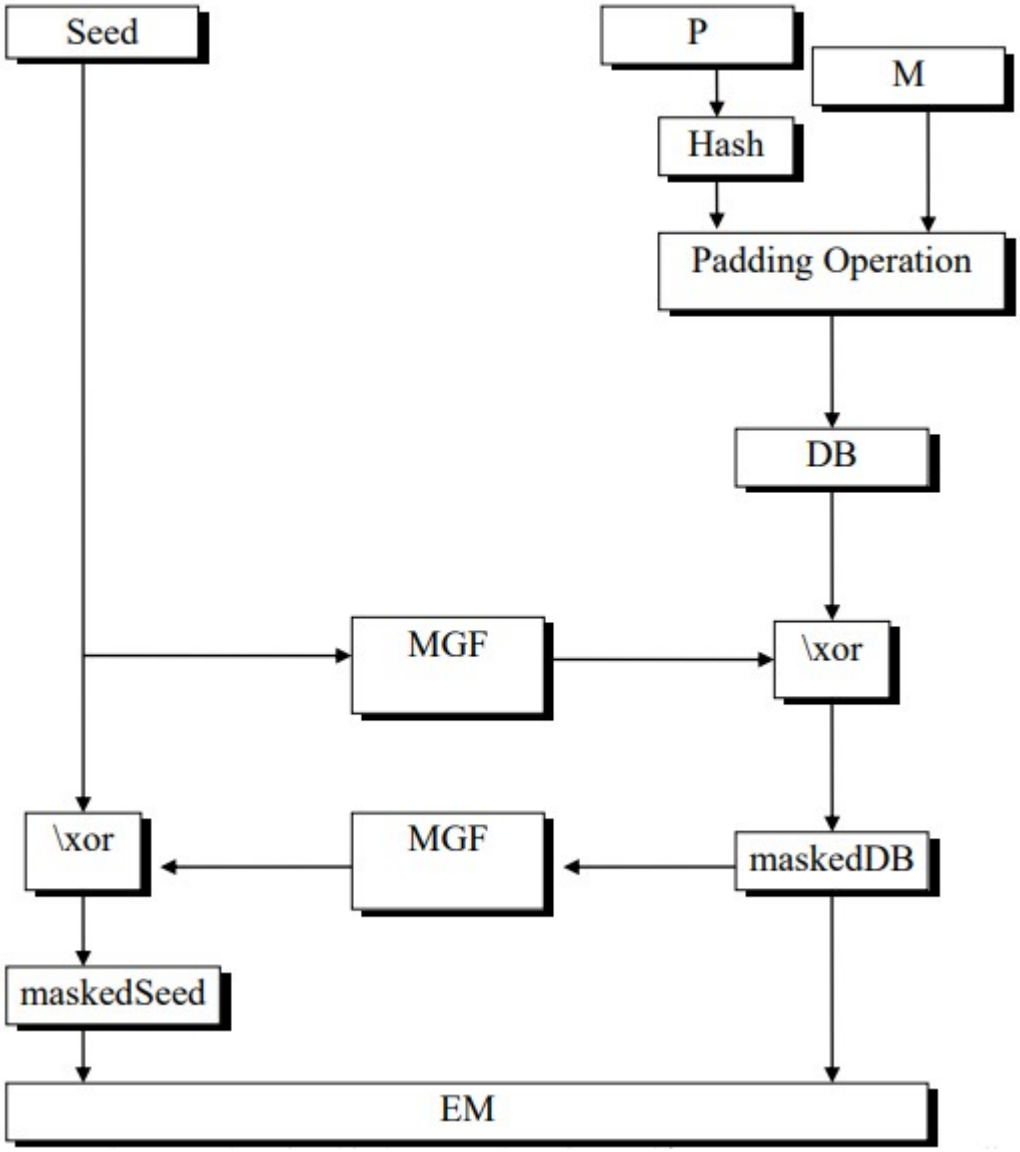
$$C = counter (I2OSP ,4)$$

- Nối C vào mầm Z rồi đem băm bằng hàm băm Hash , nối kết quả băm vào T

$$T = T \,||\, Hash (Z \,||\, C)$$

Bước 4: Xác định mặt nạ cần sinh là l octet đầu tiên của chuỗi T .

4.1.3. Hàm mã hoá RSAES-OAEP



RSAES-OAEP-ENCRYPT((n, e), M,L)

Các lựa chọn: Hash là hàm băm (hLen ký hiệu độ dài theo byte của đầu ra hàm băm)

MGF là hàm sinh mặt nạ

Đầu vào: (n, e) là khoá công khai RSA của người nhận (k ký hiệu độ dài theo byte của RSA modulo n).

M là thông báo được mã hóa, chuỗi byte có độ dài mLen với mLen ≤ k –2hLen-2.

	L là nhãn tùy chọn liên quan đến thông báo, giá trị mặc định của L là chuỗi rỗng nếu không được cung cấp.
Đầu ra:	C bản mã, chuỗi byte có độ dài k.
Các lỗi:	“thông báo quá dài”, “nhãn quá dài”.
Giả thiết:	Khoá công khai RSA (n, e) là hợp lệ.

Khi đó, việc chuyển đổi bản rõ được thực hiện như sau:

Bước 1: Kiểm tra kích thước của P. Nếu kích thước của P lớn hơn so với kích thước tối đa của đầu vào hàm băm Hash (đối với SHA1 thì kích thước tối đa của đầu vào là 2^{61} - 1 octet) thì đưa ra thông báo là "Parameter string too long" ("Tham số P quá dài") và dừng.

Bước 2: Nếu |M| > emLen – 2hLen -1 thì đưa ra thông báo "Message too long" ("Thông điệp quá dài") và dừng. Nhắc lại rằng emLen chính là kích thước khóa, hay nói chính xác hơn thì đó là kích thước của modulus.

Bước 3: Sinh một chuỗi PS gồm

emLen - |M| - 2hLen -1 octet 00.

Ở đây, độ dài của PS có thể là 0.

Bước 4: Tính giá trị băm của tham số P: pHash = Hash (P) . Kết quả băm pHash có độ dài bằng hLen .

Bước 5: Ghép nối các giá trị pHash PS , , octet 01 và M để được khối dữ liệu:

DB = pHash |PS| |01| M

Để ý rằng độ dài của khối DB sẽ là

DB = hLen + (emLen – M -2hLen -1) + 1 + M = emLen – hLen

Bước 6: Sinh một chuỗi ngẫu nhiên Seed gồm hLen octet

Bước 7: Trên cơ sở chuỗi ngẫu nhiên thu được ở bước 6, sinh mặt nạ có độ dài

emLen – hLen:

dbMash = MGF (Seed, emLen - hLen)

Bước 8: Che dấu khối dữ liệu thu được ở bước 5 bằng mặt nạ thu được ở bước 7 (hai khối này có kích thước bằng nhau và bằng emLen - hLen):

$$maskedDB = DB \oplus dbMask$$

Bước 9: Tiếp tục sử dụng kết quả thu được ở bước 8 để sinh một mặt nạ khác có kích thước là hLen , tức là bằng với kích thước của chuỗi ngẫu nhiên Seed có được ở bước 6:

$$seedMask = MGF(maskedDB, hLen)$$

Bước 10: Che dấu chuỗi ngẫu nhiên Seed bằng mặt nạ thu được ở bước 9:

$$maskedSeed = Seed \oplus seedMask$$

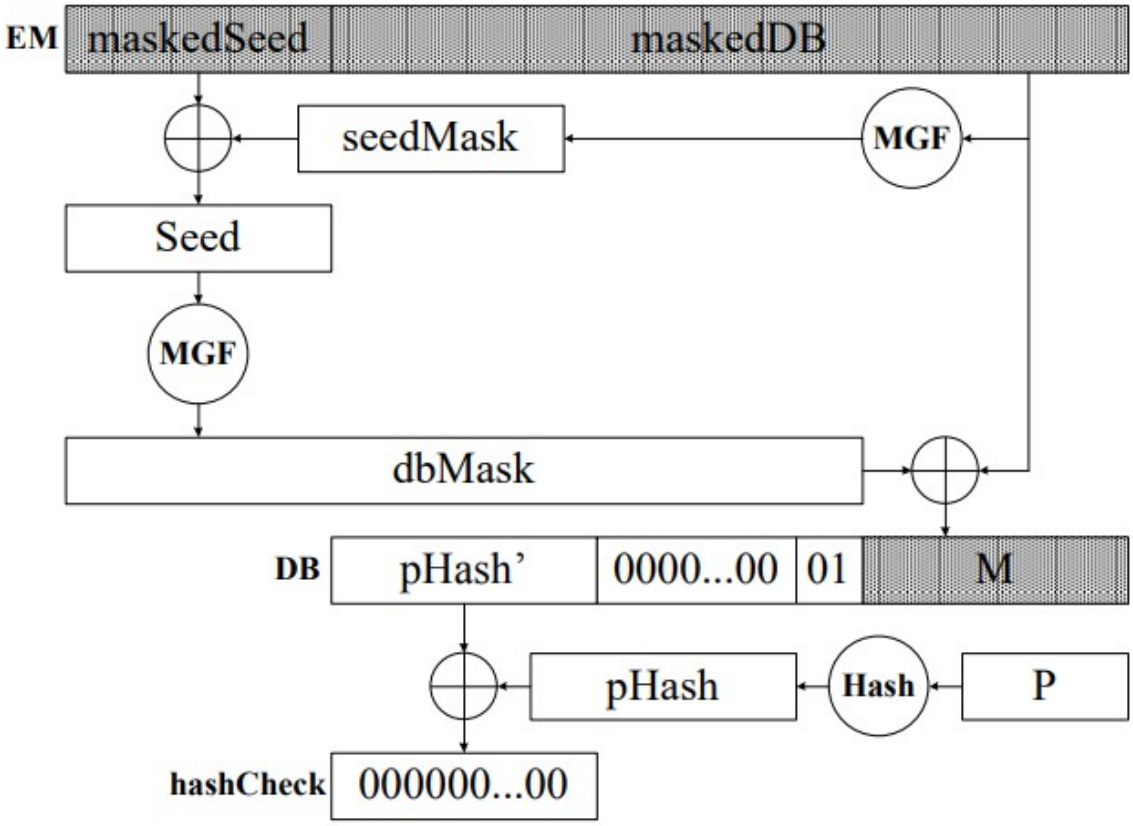
Bước 11: Kết quả cuối cùng của việc chuyển đổi là:

$$EM = maskedSeed \,||\, maskedDB$$

4.1.4. Hàm giải mã RSAES-OAEP

Việc giải chuyển đổi bản rõ theo lược đồ OAEP có thể được xem xét như là áp dụng một hàm đối với thông điệp EM:

Có thể nhận thấy rằng kết quả chuyển đổi (EM) bao gồm hai phần là maskedSeed và masedDB . Trong đó masedDB là kết quả che dấu khối DB, còn maskedSeed là kết quả che dấu chuỗi ngẫu nhiên Seed bằng các mặt nạ tương ứng. Do vậy, việc giải chuyển đổi cho khối EM có thể được trình bày như sau:



RSAES-OAEP-DECRYPT(K, C,L)

Các lựa chọn: Hash là hàm băm (hLen ký hiệu độ dài theo byte của đầu ra hàm băm

MGF là hàm sinh mặt nạ

Đầu vào: K là khoá riêng RSA của người nhận (k ký hiệu độ dài theo byte của RSA-modulo n).

C là bản mã cần được giải mã, chuỗi byte có độ dài k,

với $k \geq 2hLen + 2$.

L là nhãn tùy chọn liên quan đến thông báo, giá trị mặc định của L là chuỗi rỗng nếu không được cung cấp.

Đầu ra: M thông báo, một chuỗi byte có độ dài mLen với

$mLen \leq k - 2hLen - 2$

Lỗi: "Lỗi giải mã"

Bước 1: Kiểm tra kích thước của P. Nếu kích thước của P lớn hơn so với kích thước tối đa của đầu vào hàm băm Hash (đối với SHA1 thì kích thước tối đa của

đầu vào là 2^{61} -1 octet) thì đưa ra thông báo là "Decoding error" ("Có lỗi khi giải chuyển đổi") và dừng.

Bước 2: Nếu EM < 2hLen + 1 thì đưa ra thông báo "Decoding error" ("Có lỗi khi giải chuyển đổi") và dừng.

Bước 3: Xây dựng lại mặt nạ đã được sử dụng để che dấu Seed :

$$seedMask = MGF(maskedDB, hLen)$$

Bước 4: Gỡ mặt nạ để tìm lại Seed :

$$Seed = maskedSeed \oplus seedMask$$

Bước 5: Xây dựng lại mặt nạ đã được sử dụng để che dấu DB :

$$seedDB = MGF(Seed, EM - hLen)$$

Bước 6: Gỡ mặt nạ để tìm lại DB :

$$DB = maskedDB \oplus dbMask$$

Bước 7: Phân tách khối DB theo cấu trúc đã biết. Phần thứ nhất có kích thước hLen octet là giá trị băm pHash' của tham số P. Kế tiếp đó là PS, một dãy octet 00 với độ dài không xác định, nhưng biết rằng dãy này kết thúc khi gặp octet 01 đầu tiên. Và toàn bộ phần còn lại của DB chính là bản rõ M cần tìm.

$$DB = pHash' ||PS||01||M$$

Nếu không tìm thấy octet 01 như trên thì đưa ra thông báo "Decoding error" ("Có lỗi khi giải chuyển đổi") và dừng.

Bước 8: Tính lại giá trị băm pHash của P

$$pHash = Hash(P)$$

và so sánh với giá trị pHash' . Nếu hai giá trị này khác nhau ($pHash \oplus pHash' \neq 0$) thì đưa ra thông báo "Decoding error" ("Có lỗi khi giải chuyển đổi") và dừng.

Bước 9: Nếu trong các bước trước không có lỗi xảy ra thì thông báo kết quả cần tìm là M và dừng thuật toán.

CHƯƠNG V. CƠ SỞ TOÁN HỌC CỦA LƯỢC ĐỒ KÝ SỐ RSASSA-PSS

5.1 Các hàm cơ sở chuyển đổi dữ liệu

5.1.1 Hàm cơ sở chuyển đổi dữ liệu số nguyên sang dạng chuỗi octet

I2OSP *(x, xLen)*

Chức năng: Chuyển số nguyên không âm x thành một chuỗi octet có độ dài *xLen*

Đầu vào: x Số nguyên không âm cần chuyển đổi

Đầu ra: X Chuỗi octet tương ứng có độ dài xLen

Thông báo lỗi: "số nguyên quá lớn"

Các bước:

1. nếu $x \geq 256^{xLen}$, cho ra thông báo lỗi "số nguyên quá lớn" và dừng

2. viết số nguyên *x* duy nhất gồm *xLen* chữ số với cơ số 256:

$x = x_{xLen-1} \cdot 256^{xLen-1} + x_{xLen-2} \cdot 256^{xLen-2} + \ldots + x_1 \cdot 256 + x_0$

với $0 \leq x_i < 256$ (chú ý rằng một hay nhiều chữ số đầu sẽ bằng 0 nếu x nhỏ hơn 256^{xLen-1}).

3. cho octet X_i giá trị nguyên x_{xLen-i} với $1 \leq i \leq xLen$. Cho ra chuỗi octet

$$X = X_1 X_2 \ldots X_{xLen}$$

5.1.2 Hàm cơ sở chuyển đổi từ dạng chuỗi octet về dạng số nguyên

OS2IP (X)

Chức năng: Chuyển chuỗi octet thành một số nguyên không âm

Đầu vào: X Chuỗi octet cần chuyển đổi

Đầu ra: x Số nguyên không âm tương ứng

Các bước:

1. cho $X_1X_2...X_{xLen}$ là các octet của X từ octet đầu tiên với octet cuối cùng, x_{xLen-i} là giá trị nguyên của octet Xi với $1 \le i \le xLen$;

2. cho $x = x_{xLen-1} \cdot 256^{xLen-1} + x_{xLen-2} \cdot 256^{xLen-2} + ... + x_1 \cdot 256 + x_0$;

3. xuất ra x.

2. Các phép toán mật mã cơ sở

2.1. Phép toán cơ sở RSASP

RSASP *(K, m)*

Đầu vào: *K* Khóa bí mật RSA, với K có một trong hai dạng sau:
- cặp (*n, d*);
- bộ năm *(p, q, dP, dQ, qlnv)*;

m Biểu diễn của thông điệp, dưới dạng số nguyên giữa 0 và *n*-1

Đầu ra: s Biểu diễn của chữ ký, là số nguyên giữa 0 và *n*-1

Thông báo lỗi: "biểu diễn thông điệp ở ngoài miền hợp lệ"

Giả thiết: K là một khóa bí mật RSA hợp lệ

Các bước:

1. nếu biểu diễn của thông điệp *m* không nằm giữa 0 và *n*-1, cho ra thông báo lỗi "biểu diễn thông điệp ở ngoài miền hợp lệ" và dừng lại;

2. biểu diễn của chữ ký được tính như sau:

a. nếu dạng thứ nhất (*n, d*) của *K* được sử dụng thì $s = m^d \bmod n$;

b. nếu dạng thứ hai (*p, q, dP, dQ, qlnv*) của K được sử dụng thì tiến hành như sau:

i. Lấy $s_1 = m^{dP} \bmod p$ và $s_2 = m^{dQ} \bmod q$

ii. Đặt $h = (s_1 - s_2) \cdot qlnv \bmod p$

iii. Đặt $s = s_2 + q \cdot h$

c. xuất ra *s.*

2.2. Phép toán cơ sở RSAVP

RSAVP *((n, e), s)*

Đầu vào:	*(n, e)*	Khóa công khai RSA
	s	Biểu diễn của chữ ký, là số nguyên giữa 0 và *n*-1
Đầu ra:	m	Biểu diễn của thông điệp, là số nguyên giữa 0 và *n*-1
Thông báo lỗi:		"biểu diễn chữ ký ở ngoài miền hợp lệ"
Giả thiết:		Khóa công khai RSA (*n, e*) là hợp lệ

Các bước:

1. nếu biểu diễn của chữ ký s không nằm giữa 0 và *n*-1, cho ra "biểu diễn chữ ký ở ngoài miền hợp lệ" và dừng lại;

2. đặt $m = s^e \bmod n$;

3. xuất ra *m.*

3. Lược đồ chữ ký RSA kèm phụ lục theo PSS

Các thao tác tạo chữ ký số áp dụng một thao tác định khuôn dạng vào một thông điệp trước khi nó được chuyển thành một biểu diễn thông điệp ở dạng số nguyên. Phép toán cơ sở RSASP được áp dụng vào biểu diễn thông điệp này để tạo ra chữ ký số. Đảo ngược quá trình này, các thao tác kiểm tra chữ ký áp dụng một phép toán cơ sở RSAVP vào một chữ ký để khôi phục một biểu diễn thông điệp, sau đó nó được chuyển thành thông điệp đã được định dạng ở dạng chuỗi octet.

Thao tác kiểm tra được áp dụng vào thông điệp ban đầu và thông điệp đã được định dạng để xác định xem chúng có tương ứng với nhau hay không.

3.1. Thao tác tạo chữ ký

RSASSA-PSS-SIGN*(K, M)*

Đầu vào:	*K*	Khóa bí mật RSA của người ký
	M	Thông điệp sẽ được ký, là một chuỗi octet
Đầu ra:	S	Chữ ký, chuỗi octet có độ dài k, với k là độ dài của môđun RSA theo octet
Thông báo lỗi:		"văn bản quá dài", "lỗi định dạng"

Các bước:

1. *mã hóa EMSA-PSS*: Áp dụng thao tác EMSA-PSS-ENCODE (được giới thiệu ở phần sau) vào văn bản *M* để tạo ra thông điệp được định dạng *EM* có độ dài $\lceil (modBits\text{-}1)/8 \rceil$ octet sao cho độ dài bit của số nguyên OS2IP *(EM)* nhiều nhất là *modBits*-1, với *modBits* là độ dài theo bit của số n (mođun RSA):

EM = EMSA-PSS-ENCODE *(M*, *modBits-1)*

Chú ý rằng độ dài octet của EM sẽ bằng k - 1 nếu *modBits-1* chia hết cho 8 và bằng k nếu *modBits-1* không chia hết cho 8. Nếu hàm EMSA-PSS-ENCODE cho ra thông báo lỗi "văn bản quá dài" thì RSASSA-PSS-SIGN cũng cho ra thông báo lỗi "văn bản quá dài" và dừng lại. Nếu EMSA-PSS-ENCODE cho ra thông báo "lỗi định dạng" thì RSASSA-PSS-SIGN cũng cho ra thông báo "lỗi định dạng" và dừng lại.

2. *Chữ ký RSA*:

a. chuyển thông điệp đã được định dạng (chuỗi octet) *EM* thành biểu diễn thông điệp ở dạng số nguyên *m*;

m = OS2IP (EM)

b. áp dụng phép toán cơ sở RSASP với K là khóa bí mật RSA và biểu diễn thông điệp *m* để tạo ra biểu diễn chữ ký là số nguyên *s*:

s = RSASP(K, m);

c. chuyển chữ ký s dạng số nguyên thành chữ ký S dạng chuỗi octet có độ dài k:

S = I2OSP (*s, k*)

3. *xuất ra chữ ký S*.

3.2. Thao tác kiểm tra chữ ký

RSASSA-PSS-VERIFY*((n, e), M, S)*

Đầu vào: *(n, e)* Khóa công khai RSA của người ký

M Thông điệp mà chữ ký của nó cần được kiểm tra, là chuỗi octet

S Chữ ký được kiểm tra, chuỗi octet có độ dài k, với k là độ dài theo octet của số n, môđun RSA

Đầu ra: "chữ ký hợp lệ" hoặc "chữ ký không hợp lệ"

Các bước:

1. kiểm tra độ dài: Nếu độ dài của chữ ký *S* không là *k* octet, cho ra thông báo lỗi "chữ ký không hợp lệ" và dừng;

2. kiểm tra chữ ký RS;

a. chuyển chữ ký S thành biểu diễn chữ ký ở dạng số nguyên s;

s = OS2IP *(S)*

b. áp dụng phép toán cơ sở RSAVP với khóa công khai RSA là (*n, e*) và biểu diễn chữ ký s để tạo ra m là số nguyên biểu diễn thông điệp;

m = RSAVP *((n, e), s)*

c. chuyển biểu diễn thông điệp m thành thông điệp đã được định dạng *EM* có độ dài *emLen* =⌈(*modBits*-1/8⌉ octet, với *modBits là độ dài theo bit của số n (mođun RSA):*

EM = I2OSP (m, emLen)

Chú ý rằng *emLen* sẽ bằng *k*-1 nếu *modBits-1* chia hết cho 8 và bằng k nếu *modBits-1* không chia hết cho 8. Nếu I2OSP cho ra thông báo lỗi "số nguyên quá lớn" thì RSASSA-PSS-VERIFY cho ra thông báo lỗi "chữ ký không hợp lệ" và dừng lại.

3. *kiểm tra EMSA-PSS:* Áp dụng thao tác kiểm tra EMSA-PSS-VERIFY (sẽ được mô tả ở phần 5.6 dưới đây) vào thông điệp M và thông điệp đã được định dạng EM để xác định xem chúng có tương ứng với nhau hay không;

Result = EMSA-PSS-VERIFY (*M, EM, modBits-1)*

4. nếu kết quả (Result) là "phù hợp" thì cho ra "chữ ký hợp lệ". Ngược lại sẽ cho ra "chữ ký không hợp lệ".

4. Phương pháp định dạng cho chữ ký kèm phụ lục theo PSS (EMSA-PSS)

4.1. Thao tác định dạng

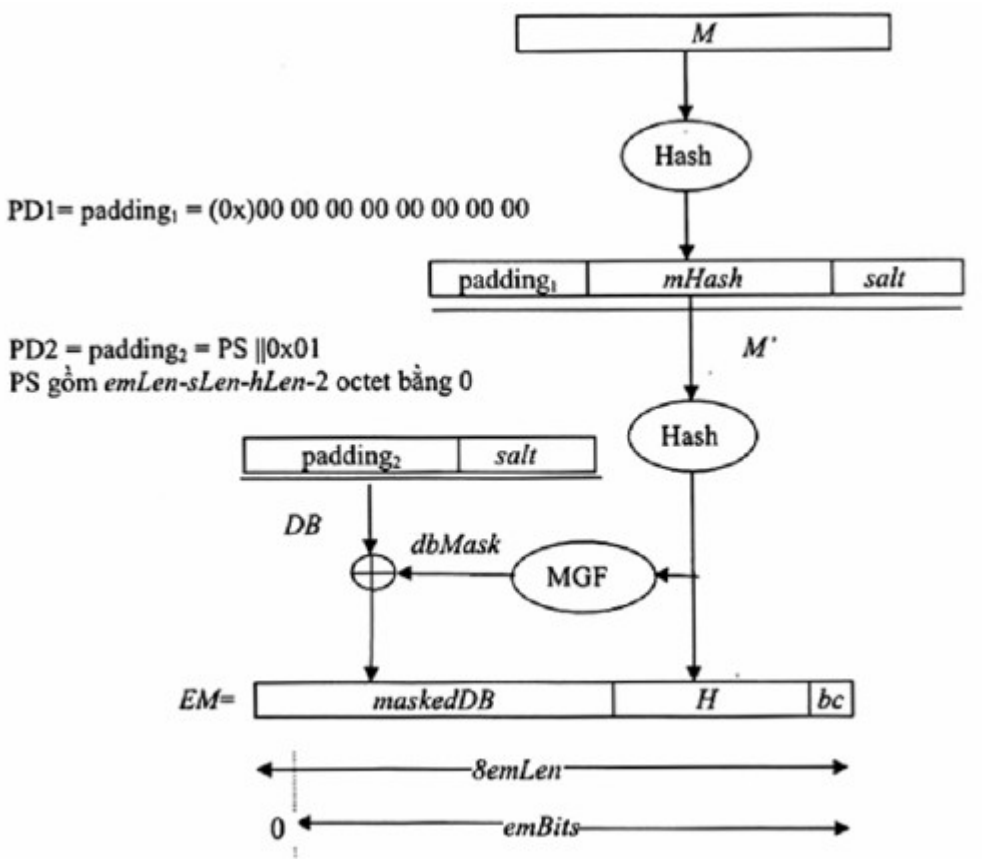
Phương pháp định dạng được tham số hóa bằng cách chọn:

- hàm băm (cố định với khóa RSA đã cho);

- hàm tạo mặt nạ (cố định với khóa RSA đã cho) và;

- độ dài phần phụ thêm (có thể thay đổi với khóa RSA đã cho).

Các hàm băm và hàm tạo mặt nạ được đề xuất sẽ được mô tả trong phần 6 và 5.6.3. Hình 2 minh họa thao tác định dạng.



Hình 2 - Minh họa thao tác định dạng

Công thức để tính *EM:*

((PD2||*r*) ⊕ MGF(*h*(PD1||*h*(M)||*r*))) || *h*(PD1||*h*(*M*)||*r*)) || *0xbc*

EMSA-PSS-ENCODE *(M, emBits)*

Lựa chọn:	*h*	Hàm băm (độ dài đầu ra của nó theo octet là *hLen)*
	MGF	Hàm tạo mặt nạ
	sLen	Đô dài chủ định của phần phụ thêm theo octet
Đầu vào:	*M*	Văn bản để mã hóa, là một chuỗi octet
		emBits độ dài tối đa theo bit của số nguyên OS2IP (*EM*), ít nhất bằng *8hLen + 8sLen + 9*
Đầu ra:	*EM*	Văn bản đã được mã, đó là chuỗi octet có độ dài emLen

$= \lceil emBits/8 \rceil$

Thông báo lỗi: "lỗi định dạng"; "văn bản quá dài"

Các bước:

1. nếu độ dài của *M* lớn hơn giới hạn đầu vào cho hàm băm (2^{64}-1 đối với SHA-256 thì cho ra thông báo lỗi "văn bản quá dài" và dừng;

2. lấy *mHash = h(M)*, đó là một chuỗi octet dài *hLen*;

3. nếu *emLen* < *hLen* + *sLen* + 2, cho ra thông báo "lỗi định dạng" và dừng;

4. tạo ra chuỗi octet ngẫu nhiên salt có độ dài *sLen*; nếu *sLen* = 0 thì salt không có;

5. đặt *M'* = (0x)00 00 00 00 00 00 00 00 || *mHash* ||*salt*;

6. lấy *H = Hash(M')*, đó là một chuỗi octet dài *hLen*;

7. lấy PS là một chuỗi octet bằng 0 dài *emLen* - *hLen* - *sLen* - 2;

8. Lấy *DB = PS* || 0x01 || *salt*; *DB* là một chuỗi octet dài *emLen-hLen* - 1;

9. lấy *maskedDB = DB* ⊕ *dbMask*;

10. đặt *8emLen-emBits* bit đầu tiên bên trái của octet đầu tiên bên trái trong *maskedDB* bằng 0;

11. lấy *EM = maskedDB* ||*H* || *0xbc*;

12. xuất ra *EM*.

4.2. Thao tác kiểm tra

EMSA-PSS-VERIFY

(M, EM, emBits)

Lựa chọn:	*h*	Hàm băm (độ dài đầu ra của nó theo octet là *hLen)*
	MGF	Hàm tạo mặt nạ
	sLen	Đô dài dự kiến của phần thêm theo octet
Đầu vào:	*M*	Thông điệp cần kiểm tra chữ ký, là chuỗi octet

EM Thông điệp đã được định dạng, là chuỗi octet có độ dài $emLen = \lceil emBits/8 \rceil$

emBits Độ dài tối đa theo bit của số nguyên OS2IP (EM), tối thiểu là *8hLen + 8sLen + 9*

Đầu ra: "phù hợp" hoặc "không phù hợp".

Các bước:

1. nếu độ dài của *M* lớn hơn giới hạn đầu vào của hàm băm (2^{64} - 1 octet đối với SHA-256, thì đưa ra thông báo "không phù hợp" và dừng;

2. đặt *mHash = h(M)*, là chuỗi octet có độ dài *hLen*;

3. nếu *emBits* < *8hLen+8sLen+9*, đưa ra thông báo "không phù hợp" và dừng;

4. nếu octet đầu tiên bên phải của EM không chứa giá trị bc, đưa ra thông báo "không phù hợp" và dừng;

5. đặt *maskedDB* là *emLen-hLen-1* octet đầu tiên bên trái của *EM*, và *H* là *hLen* octet tiếp theo;

6. nếu *8emLen-emBits* bit đầu tiên bên trái của octet đầu tiên bên trái trong *maskedDB* không phải tất cả bằng 0, đưa ra thông báo "không phù hợp" và dừng;

7. đặt *dbMask = MGF*(*H, emLen-hLen-1*);

8. đặt *DB = maskedDB* ⊕ *dbMask*;

9. thiết lập *8emLen-emBits* bit đầu tiên bên trái của *DB* bằng 0;

10. nếu *emLen-hLen-sLen-2* octet đầu tiên bên trái của *DB* không phải bằng 0 hoặc nếu octet tại vị trí thử *emLen-hLen-sLen-1* không bằng 0x01, đưa ra thông báo "không phù hợp" và dừng;

11. đặt *salt* bằng *sLen* octet cuối cùng của DB;

12. đặt M' = 00 00 00 00 00 00 00 00 || *mHash* || *salt*

M' là chuỗi octet có độ dài *8+hLen+sLen* với 8 octet bằng 0 khởi đầu;

13. đặt *H' = h(M')*, là chuỗi octet có độ dài *hLen*;

14. nếu *H = H'*, đưa ra thông báo "phù hợp". Ngược lại, đưa ra thông báo "không phù hợp" .

4.3. Hàm tạo mặt nạ MGF dựa vào hàm băm

MGF*(mgfSeed, maskLen)*

Lựa chọn	*h*	Hàm băm (độ dài đầu ra của nó theo octet là *hLen)*
Đầu vào:	*mgfSeed*	Mầm được dùng để tạo mặt nạ, là chuỗi octet
	maskLen	Độ dài chủ ý theo octet của mặt nạ, nhiều nhất là $2^{32}hLen$
Đầu ra:	*mask*	Mặt nạ, là chuỗi octet có độ dài *maskLen*

Thông báo lỗi: "mặt nạ quá dài"

Các bước:

1. nếu *maskLen* > 2^{32} *hLen*, cho ra thông báo lỗi "mặt nạ quá dài" và dừng;

2. lấy *T* là chuỗi octet rỗng;

3. với *counter* chạy từ 0 tới $\lceil maskLen/hLen \rceil - 1$, thực hiện các bước

a. chuyển *counter* thành một chuỗi octet C có độ dài 4 octet;

C = I2OSP (*counter,* 4)

b. nối hàm *mgfSeed* với C, tính hàm băm của chuỗi này. Sau đó nối chuỗi octet *T* với giá trị băm vừa thu được.

T = T || *h*(*mgfSeed* || C)

4. xuất ra *maskLen* octet đầu tiên của T như là chuỗi octet *mask*.

CHƯƠNG VI. BÀI TẬP

Bài 1: Dùng thuật toán Euclide tìm phần tử nghịch đảo:

Thuật toán: Cho b mod m

1. $(A_1, A_2, A_3) = (1, 0, m)$ $(B_1, B_2, B_3) = (0, 1, b)$
2. $Q = A_3 \text{ div } B_3$ $B_3 = A_3 \text{ mod } B_3$
3. $B_1 = A_1 – Q*B_1$ $B_2 = A_2 – Q*B_2$
4. $(A_1 A_2 A_3) = (B_1, B_2, B_3)$
5. If $B_3 = 0$: gcd (b, m) = A_1

   Else $B_3 = 1$: gcd (b, m) = 1

   $b^{-1} \text{ mod } m = B_2$

- $357^{-1} \text{ mod } 1137$

Q	A1	A2	A3	B1	B2	B3
-	1	0	1137	0	1	357
3	0	1	357	1	-3	66
5	1	-3	66	-5	16	27
2	-5	16	27	11	-35	12
1	11	-35	12	-16	**51**	1

Vậy $357^{-1} \text{ mod } 1137 =$ **51**

- $213^{-1} \text{ mod } 1577$

Q	A1	A2	A3	B1	B2	B3
-	1	0	1577	0	1	213
7	0	1	213	1	-7	86

2	1	-7	86	-2	15	41
2	-2	15	41	5	-37	4
10	5	-37	4	-52	**385**	1

Vậy $213^{-1} \bmod 1577 =$ **385**

Bài 2: Tính Jacobi

Tính chất:

- Với m, n là số nguyên lẻ:

$$\left(\frac{n}{m}\right) = \begin{cases} -\left(\frac{n}{m}\right) \text{ nếu } n \equiv 3 \bmod 4 \\ \left(\frac{n}{m}\right) \text{ nếu } m \equiv 1 \bmod 4 \text{ hoặc nếu } n \equiv 1 \bmod 4 \end{cases}$$

- Với n là số nguyên lẻ:

$$\left(\frac{2}{n}\right) = \begin{cases} 1 \text{ nếu } n \equiv \pm 1 \text{ hoặc } \pm 7 \bmod 8 \\ -1 \text{ nếu } n \equiv \pm 3 \text{ hoặc } \pm 5 \bmod 8 \end{cases}$$

- Với n là số nguyên lẻ:

$$m_1 \equiv m_2 \bmod n \text{ -> } \left(\frac{m_1}{n}\right) \equiv \left(\frac{m_2}{n}\right)$$

- Với n là số nguyên lẻ:

$$\left(\frac{m_1 . m_2}{n}\right) = \left(\frac{m_1}{n}\right) . \left(\frac{m_2}{n}\right)$$

- Với t là số lẻ:

$$m = 2^k.t \Rightarrow (\frac{m}{n}) = (\frac{2}{n})^k.(\frac{t}{n})$$

- $(\frac{1}{n}) = 1$
- $(\frac{a}{m.n}) = (\frac{a}{m})(\frac{a}{n})$

Giải

- $\left(\frac{29}{199}\right) = \left(\frac{199}{29}\right) = \left(\frac{25}{29}\right) = \left(\frac{29}{25}\right) = \left(\frac{4}{25}\right) = \left(\frac{2}{5}\right)^2 = 1$
- $\left(\frac{21}{211}\right) = \left(\frac{211}{21}\right) = \left(\frac{1}{21}\right) = 1$
- $\left(\frac{47}{97}\right) = \left(\frac{97}{47}\right) = \left(\frac{3}{47}\right) = -\left(\frac{47}{3}\right) = -\left(\frac{2}{3}\right) = -(-1) = 1$
- $\left(\frac{5}{97}\right) = \left(\frac{97}{5}\right) = \left(\frac{2}{5}\right) = -¿1$

Bài 3: Cho hệ mật RSA với p = 37, q = 41 và số mũ mã hoá e = 211.

- Hãy tính số mũ giải mã d.
- Hãy mã hoá bản tin m = 47 và giải mã bản mã vừa thu được.

Giải:

Trong hệ mật RSA ta có:

n = p.q = 37.41 = 1517

⇨ Φ(n) = Φ(p.q) = Φ(p).Φ(q)

⇨ Φ(1517) = Φ(37). Φ(41) = 36.40 = 1440

⇨ Số mũ giải mã:

$d = e^{-1} \bmod \Phi(n) = 211^{-1} \bmod 1440$

Sử dụng thuật toán Euclide mở rộng tính $211^{-1} \bmod 1440$

Q	A1	A2	A3	B1	B2	B3
-	1	0	1440	0	1	211
6	0	1	211	1	-6	174
1	1	-6	174	-1	7	37
4	-1	7	37	5	-34	26
1	5	-34	26	-6	41	11
2	-6	41	11	17	-116	4
2	17	-116	4	-40	273	3
1	-40	273	3	57	**-389**	1

Số mũ giải mã $d = 211^{-1} \bmod 1440 = -389 \bmod 1440 =$ **1051**

Mã hóa bản tin với m = 47

Trong hệ mật RSA, ta có bản mã: $c = m^e \bmod n = 47^{211} \bmod 1517$

Áp dụng thuật toán nhân bình phương có lặp ta tính:

$c = 47^{211} \bmod 1517$

Đổi hệ số: $211 = 1101\ 0011 = 2^0 + 2^1 + 2^4 + 2^6 + 2^7$

⇨ t = 7

i		0	1	2	3	4	5	6	7
i	K	1	1	0	0	1	0	1	1
	A	47	692	1009	174	1453	1062	713	174
	b	47	667	667	667	1305	1305	544	**602**

⇨ Bản mã: $c = 47^{211} \bmod 1517 = \mathbf{602}$

Giải mã

Để giải mã, ta tìm: $m = c^d \bmod n$

⇨ $m = 602^{1051} \bmod 1517$

Đổi hệ số: $1051 = 100\ 0001\ 1011 = 2^0 + 2^1 + 2^3 + 2^3 + 2^4 + 2^{10}$

⇨ t = 10

i		0	1	2	3	4	5	6	7	8	9	10
i	K	1	1	0	1	1	0	0	0	0	0	1
	A	602	1358	1009	174	1453	1062	713	174	1453	1062	713
	b	602	1370	1370	211	149	149	149	149	149	149	**47**

⇨ Vậy bản rõ ban đầu:

$m = 602^{1051} \bmod 1517 = \mathbf{47}$

Bài 4: Sử dụng hệ mật Elgamal với số nguyên tố p = 211, phần tử sinh α = 39 của Z*211. Giả sử người dùng A chọn khóa bí mật a = 113.

- Hãy tìm khoá công khai của A?
- Giả sử chọn số ngẫu nhiên k = 23, hãy thực hiện mã hoá bản tin x = 34 với khoá công khai của A, và giả mã bản mã vừa thu được.

Thuật toán nhân bình phương có lặp: tính a^k mod n

- Viết k dưới dạng nhị phân => tìm t (bậc cao nhất của k)
- Đặt b =1, nếu k = 0 thì return b
- Đặt A = a
- Nếu k_0 = 1 thì b = a
- Cho i chạy từ 0 đến t
  - A = A^2 mod n
  - K_i = 1 thì b = Ab mod n
- Return b (b cuối cùng là kết quả a^k mod n)

Giải

Tìm khóa công khai (p, α, α^a)

Áp dụng thuật toán nhân bình phương có lặp ta tính: α^a mod p = 39^{113} mod 211

Đổi hệ số: 113 = 111 0001 => t=6

i	0	1	2	3	4	5	6
K_i	1	0	0	0	1	1	1
A	39	44	37	103	59	105	53
b	39	39	39	39	191	10	**108**

Vậy $\alpha^a \mod p = \mathbf{108}$

⇨ Khóa công khai là (211, 39, 108)

Tính bản mã c = (y, 2)

$y = \alpha^k \mod p = 39^{23} \mod 211$

$= (39^4)^5 \cdot 39^3 \mod 211$

$= 37^5 \cdot 39^3 \mod 211$

$= 73 \cdot 28 \mod 211$

$= \mathbf{145}$

$2 = m \cdot (\alpha^a)^k \mod p = 34 \cdot (39^{113})^{23} \mod 211$

$= 34 \cdot (108)^{23} \mod 211$

$= 34 \cdot (108^4)^5 \cdot 108^3 \mod 211$

$= 34 \cdot (105)^5 \cdot 42 \mod 211$

$= 34 \cdot 178 \cdot 42 \mod 211$

$= \mathbf{140}$

⇨ Bản mã c = (y, 2) = (145, 140)

Tính bản rõ m = (y^{p-1-a}) . 2 mod p

Áp dụng thuật toán nhân bình phương có lặp ta tính:

$m = 145^{211-1-113} \cdot 140 \mod 211 = 145^{97} \cdot 140 \mod 211$

Tính $145^{97} \mod 211$

Đổi hệ số: 97 = 110 0001 => t=6

i	0	1	2	3	4	5	6
K_i	1	0	0	0	0	1	1
A	145	136	139	120	52	172	44
b	145	145	145	145	145	42	**160**

Vậy $145^{97} \bmod 211 = 160$

⇨ Bản rõ m = 160 . 140 mod 211 = **34**

TÀI LIỆU THAM KHẢO

[1] TCVN 7635:2007 - Kỹ Thuật Mật Mã - Chữ Ký Số;

[2] RSA PKCS#7 v1.5: March, 1998. Cryptographic Message Syntax Standard , RSA security inc. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v1-5.pdf;

[3] RSA PKCS#1 v2.1: June 14, 2002. RSA Cryptography Standard, RSA security inc. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf;

[4] RSA PKCS#11 v 2.40: Mark, 2014. RSA Cryptographic Token Interface, RSA security inc. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/pkcs-1v2-40.pdf;

[5] RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://www.ietf.org/rfc/rfc5280.txt;

[6] RFC 3125: Electronic Signature Policies https://tools.ietf.org/html/rfc3125;

[7] RFC 3379: Delegated Path Validation and Delegated Path Discovery Protocol Requirements. https://tools.ietf.org/html/rfc3379.