**1. Tạo khóa**

* Chọn hai số nguyên tố lớn, giả sử là **p** và **q** . Các số nguyên tố này phải được giữ bí mật.
* Tính tích các số nguyên tố, **n = p \* q** . Tích này là một phần của khóa công khai cũng như khóa riêng tư.
* Tính [hàm Euler Totient](https://www.geeksforgeeks.org/eulers-totient-function/)**Φ(n)** là **Φ(n)** = **Φ(p \* q) = Φ(p) \* Φ(q) = (p - 1) \* (q - 1).**
* Chọn số mũ mã hóa **e** sao cho
  + 1 < e < Φ(n), và
  + gcd(e, Φ(n)) = 1, nghĩa là e phải nguyên tố cùng nhau với Φ(n).
* Tính số mũ giải mã **d,** sao cho
  + **(d \* e) ≡ 1 mod Φ(n)** , tức là d là [nghịch đảo nhân môđun](https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/) của **e** mod Φ(n). Một số phương pháp phổ biến để tính nghịch đảo nhân là: [Thuật toán Euclid mở rộng](https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/) , [Định lý nhỏ Fermat](https://www.geeksforgeeks.org/fermats-little-theorem/) , v.v.
  + Chúng ta có thể có nhiều giá trị d thỏa mãn **(d \* e) ≡ 1 mod Φ(n)** nhưng không quan trọng chúng ta chọn giá trị nào vì tất cả chúng đều là khóa hợp lệ và sẽ cho ra cùng một thông điệp khi giải mã.

Cuối cùng, Khóa công khai = (n, e) và Khóa riêng tư = (n, d) .

2. Mã hóa

Để mã hóa một thông điệp M , trước tiên nó được chuyển đổi thành dạng số bằng ASCII và các chương trình mã hóa khác. Bây giờ, sử dụng khóa công khai (n, e) để mã hóa thông điệp và lấy văn bản mã hóa bằng công thức:

C = Me mod n , trong đó C là văn bản mã hóa và e và n là các phần của khóa công khai.

3. Giải mã

Để giải mã văn bản mã hóa C , hãy sử dụng khóa riêng (n, d) và lấy dữ liệu gốc bằng công thức:

***M = Ce mod n,*** *trong đó M là thông điệp và d và n là các phần của khóa riêng.*

Vấn đề của RSA

1)Tính nhanh giải mã và mã hóa sẽ được đề cập ở bài sau

2) Sử dụng khóa công khai

Sử dụng khóa công khai nhỏ sẽ làm tăng tốc độ tính toán

Các lựa chọn phổ biến khóa công khai là e = 65537 và e = 3 và e = 17

Bởi vì khi viết dưới dạng nhị phân thì mỗi lựa chọn trên sẽ chỉ có 2 bit 1 ở đầu và cuối khi sử dụng thuật toán multi and square thì tính toán cực kỳ nhanh

Rủi ro nếu chọn khóa công khai nhỏ sẽ dễ dàng xảy ra tình trạng simple attack

Khi bản rõ quá nhỏ và không thêm đệm thì hacker có thể dễ dàng tính ngược lại được bản rõ một cách dễ dàng mà không cần private key bởi vì n quá lớn mod vẫn ra chính kết quả lũy thừa M mũ e

Ví dụ M = 5 (nhỏ và chưa thêm padding)

e = 3

n = 9999991

C = 5^3 mod n = 125

M = căn 3 của 125 = 5 dễ dàng tìm ra M mà không cần private key

Giải pháp  
+ không chọn e = 3 hoặc 17 nếu như không cần thiết. Nên lựa chọn có độ lớn vừa đủ và hiệu quả như e = 65537

+ Luôn sử dụng cơ chế padding OAEP và PKCS#1 v1.5

3) giải mã RSA và khóa d hay secret key

Vấn đề:

+ d nhỏ quá thì dễ dàng bị brute force

+ Ở bước mã hóa dùng multi and square để tăng tốc thì ở bước giải mã dùng định lý dư Trung Hoa để tăng tốc với d lớn (CRT) <https://drx.home.blog/2018/07/25/dinh-ly-so-du-trung-hoa/>. Đây là link đọc qua định lý này

+Việc áp dụng cách tính CRT giúp tốc độ giải mã nhanh hơn gấp khoảng 4 lần so với cách tính trực tiếp M = c^d mod n

+ Thuật toán CRT được dùng khi chúng ta đã biết p và q nên là không dùng định lý ở bước mã hóa vì mã hóa không có p và q để áp dụng nên là mã hóa dùng e nhỏ kết hợp với kỹ thật binary exponentiation ( lũy thừa nhị phân hoặc square-and-Multiply)

4) Quy trình tạo khóa thực tế

Trước khi sử dụng hệ mật mã công khai (public-key cryptosystem), mỗi người tham gia cần tạo một **cặp khóa**, bao gồm khóa công khai (public key) và khóa bí mật (private key).

**1. Chọn hai số nguyên tố lớn p và q**  
Hai số nguyên tố này phải đủ lớn để đảm bảo tính an toàn. Việc chọn p và q là bước quan trọng đầu tiên. Nếu chúng quá nhỏ, kẻ tấn công có thể dễ dàng phân tích n = p × q để tìm ra khóa bí mật.

**2. Chọn e hoặc d rồi tính giá trị còn lại**  
Thông thường, người ta chọn trước giá trị e (public exponent), ví dụ như 65537 vì nó có ít bit 1 nên tính toán nhanh, sau đó tính d (private exponent) sao cho:  
    **e × d ≡ 1 mod φ(n)**  
với φ(n) = (p – 1)(q – 1)

**3. Chọn p và q đủ lớn để n = p × q an toàn**  
Giá trị n = p × q sẽ được công khai, nên nếu p và q quá nhỏ, đối thủ có thể phân tích n thành p và q bằng thuật toán phân tích thừa số (factorization). Vì vậy, p và q phải được chọn từ một tập các số nguyên tố đủ lớn.

**4. Phải dùng phương pháp tìm số nguyên tố hiệu quả**  
Để tìm ra các số nguyên tố lớn, cần sử dụng các thuật toán ngẫu nhiên kết hợp với kiểm tra nguyên tố xác suất (chẳng hạn như thuật toán Miller-Rabin). Phương pháp này phải nhanh và hiệu quả vì RSA yêu cầu sinh số nguyên tố có hàng trăm đến hàng nghìn bit.

**5)Quy trình chọn một số nguyên tố (Procedure for Picking a Prime Number)**

1. **Chọn ngẫu nhiên một số lẻ n**  
   Vì mọi số nguyên tố (trừ 2) đều là số lẻ, nên chỉ chọn số lẻ để tiết kiệm thời gian kiểm tra.
2. **Chọn ngẫu nhiên một số a sao cho a < n**  
   Số a này sẽ được dùng làm tham số trong phép kiểm tra nguyên tố xác suất.
3. **Thực hiện kiểm tra nguyên tố xác suất với tham số a**  
   Dùng một thuật toán xác suất (chẳng hạn như Miller-Rabin) để kiểm tra xem n có phải số nguyên tố không.
   * Nếu n **không vượt qua kiểm tra**, **loại bỏ n** và quay lại bước 1 để thử số mới.
   * Nếu n **vượt qua**, tiếp tục kiểm tra thêm.
4. **Nếu n vượt qua đủ số lần kiểm tra**, thì **chấp nhận n là số nguyên tố**.  
   Nếu không vượt qua đủ số lần kiểm tra, **quay lại bước 2** để chọn a mới và kiểm tra lại.

5) Một số vấn đề thêm

1. **Tấn công Brute-force (Vét cạn)**

* **Mô tả**: Kẻ tấn công thử tất cả các khóa bí mật (private key) có thể cho đến khi tìm được khóa đúng.
* **Đối phó**: Sử dụng khóa đủ lớn để không gian khóa quá rộng, khiến việc tấn công brute-force trở nên bất khả thi về mặt thời gian và tài nguyên.
* **Hạn chế**:
  + Khóa phải đủ nhỏ để đảm bảo hiệu suất mã hóa/giải mã.
  + Hiện nay, tốc độ mã hóa/giải mã của hệ mật khóa công khai (như RSA) vẫn chậm hơn so với mật mã khóa đối xứng, nên nó thường chỉ được dùng cho các ứng dụng quản lý khóa hoặc chữ ký số.

2. **Tấn công Tính Toán Khóa Bí Mật Từ Khóa Công Khai**

* **Mô tả**: Kẻ tấn công cố gắng tính toán khóa bí mật từ khóa công khai bằng cách giải các bài toán toán học đằng sau hệ mật (ví dụ: phân tích số nguyên tố lớn trong RSA).
* **Tình trạng**:
  + Hiện chưa có chứng minh toán học nào khẳng định bài toán này là "khó" một cách tuyệt đối (ví dụ: chưa chứng minh phân tích số nguyên tố là NP-đầy đủ).
  + Nếu có bước đột phá trong toán học (ví dụ: thuật toán phân tích số nguyên tố nhanh), hệ mật như RSA có thể bị phá vỡ.

3. **Tấn công Probable-Message (Tấn công Dựa trên Tin Nhắn Có Thể Đoán Trước)**

* **Mô tả**: Kẻ tấn công đoán nội dung tin nhắn (ví dụ: tin nhắn ngắn hoặc có cấu trúc dễ đoán) và kiểm tra xem bản mã có khớp với kết quả mã hóa của tin nhắn đó không.
* **Đối phó**: Thêm các bit ngẫu nhiên vào tin nhắn trước khi mã hóa (padding), chẳng hạn như sử dụng OAEP trong RSA, để tin nhắn trở nên khó đoán hơn.

Liên Hệ Với RSA

* **Brute-force**: RSA sử dụng khóa đủ lớn (2048 bit hoặc hơn) để chống lại tấn công này.
* **Tính toán khóa bí mật**: Dựa trên độ khó của bài toán phân tích số nguyên tố lớn. Hiện chưa có thuật toán hiệu quả để giải quyết bài toán này.
* **Probable-message attack**: RSA cần các sơ đồ padding như PKCS#1 hoặc OAEP để ngăn chặn tấn công này.

6)

1. **Tấn công Brute-force (Vét cạn)**

* **Mô tả**: Kẻ tấn công thử lần lượt tất cả các khóa bí mật (private key) có thể cho đến khi tìm được khóa đúng.
* **Đối phó**:
  + RSA sử dụng khóa lớn (thường từ 2048 bit trở lên) để không gian khóa đủ rộng, khiến việc brute-force mất hàng triệu năm với công nghệ hiện tại.
  + Ví dụ: Khóa 1024 bit đã lỗi thời do nguy cơ bị tấn công bằng máy tính lượng tử.

2. **Tấn công Toán học (Mathematical Attacks)**

* **Mô tả**: Tập trung vào việc **phân tích tích của hai số nguyên tố** (n=p×q*n*=*p*×*q*) để tìm khóa bí mật.
* **Phương pháp**:
  + Phân tích thừa số nguyên tố (Factorization) bằng các thuật toán như **Sieve of Eratosthenes**, **Pollard's Rho**, hoặc **General Number Field Sieve (GNFS)**.
  + Hiệu quả phụ thuộc vào kích thước khóa: Với n*n* đủ lớn (ví dụ 3072 bit), bài toán được coi là "khó" với công nghệ hiện tại.
* **Lưu ý**:
  + Tất cả các phương pháp toán học đều quy về bài toán phân tích số nguyên tố, chưa có cách nào hiệu quả hơn.

3. **Tấn công Thời gian (Timing Attacks)**

* **Mô tả**: Đo thời gian thực hiện giải mã để suy luận về khóa bí mật (ví dụ: thời gian xử lý bit 1 vs bit 0 khác nhau).
* **Ví dụ**: Nếu máy chủ tốn nhiều thời gian hơn khi giải mã các bit nhất định, kẻ tấn công có thể dò ra khóa.
* **Đối phó**:
  + Sử dụng thuật toán giải mã có thời gian cố định (constant-time decryption).
  + Thêm độ trễ ngẫu nhiên vào quá trình xử lý.

4. **Tấn công Dựa trên Lỗi Phần Cứng (Hardware Fault-based Attack)**

* **Mô tả**:
  + Kẻ tấn công gây ra lỗi phần cứng (ví dụ: nhiễu điện, tia vũ trụ) làm sai lệch kết quả tính toán khi thiết bị tạo chữ ký số.
  + Lợi dụng kết quả sai để tìm khóa bí mật (ví dụ: phân tích chữ ký bị lỗi).
* **Đối phó**:
  + Kiểm tra tính toàn vẹn của chữ ký trước khi xuất kết quả.
  + Sử dụng cơ chế chống nhiễu phần cứng.

5. **Tấn công Bản Mã Được Chọn (Chosen Ciphertext Attacks - CCA)**

* **Mô tả**:
  + Kẻ tấn công có thể gửi nhiều bản mã do họ tự chọn đến hệ thống và phân tích kết quả giải mã để tìm khóa.
  + Lợi dụng tính chất **đồng cấu** (homomorphic) của RSA (ví dụ: (memod  n)×(cemod  n)=(m×c)emod  n(*me*mod*n*)×(*ce*mod*n*)=(*m*×*c*)*e*mod*n*).
* **Đối phó**:
  + Sử dụng sơ đồ padding an toàn như **OAEP** (Optimal Asymmetric Encryption Padding) để ngăn chặn thao tác bản mã.
  + Triển khai cơ chế xác thực bản mã trước khi giải mã.

Tóm tắt điểm mấu chốt

* **RSA an toàn khi**:
  + Khóa đủ lớn (≥ 2048 bit).
  + Triển khai đúng cách (chống timing attacks, CCA).
  + Sử dụng padding an toàn (OAEP/PKCS#1).
* **Nguy cơ**:
  + Máy tính lượng tử có thể phá vỡ RSA bằng thuật toán Shor (đòi hỏi khóa post-quantum trong tương lai).

**Mục tiêu của các kỹ thuật này**

Các kỹ thuật này nhằm **ngăn chặn hacker đo thời gian thực hiện phép lũy thừa (exponentiation)** trong RSA để suy ra thông tin về **khóa bí mật**.

**🛡️ 1. Constant exponentiation time (Thời gian lũy thừa cố định)**

* **Ý tưởng**: Đảm bảo rằng mọi phép lũy thừa (ví dụ: M^d mod n) **đều mất cùng một lượng thời gian**, **bất kể giá trị của khóa d là gì**.
* **Mục đích**: Hacker sẽ không thể phân tích thời gian để suy đoán từng bit của khóa.
* **Nhược điểm**: Làm chậm hiệu suất vì ngay cả các phép tính đơn giản cũng bị kéo dài thời gian cho bằng với phép tính phức tạp nhất.

**🎲 2. Random delay (Độ trễ ngẫu nhiên)**

* **Ý tưởng**: Thêm một **độ trễ ngẫu nhiên** (ví dụ, chờ ngẫu nhiên vài mili-giây) vào thuật toán lũy thừa.
* **Mục tiêu**: Làm **rối loạn thời gian thực thi**, khiến cho hacker không thể phân tích chính xác được thời gian xử lý từng bit.
* **Hiệu quả**: Tốt hơn cách 1 vì không cần cố định toàn bộ thời gian; nhưng vẫn có thể bị tấn công nếu hacker thu thập đủ dữ liệu.

**🔒 3. Blinding (Làm mù phép lũy thừa)**

* **Ý tưởng**: Trước khi giải mã (C^d mod n), ta nhân bản mã C với một **số ngẫu nhiên r**, sau đó mới lũy thừa.
  + Ví dụ: Tính C' = C × r^e mod n, sau đó giải mã C'^d, rồi **bỏ đi phần r** sau khi tính xong.
* **Lợi ích**: Vì mỗi lần đều dùng r khác nhau, hacker sẽ **không biết chính xác dữ liệu đầu vào**, nên **không thể phân tích thời gian xử lý từng bit**.
* **Hiệu quả cao**: Đây là phương pháp rất hiệu quả và thường được sử dụng trong thực tế.

**Một cuộc tấn công vào bộ xử lý đang tạo chữ ký số RSA**

* **Gây ra lỗi trong quá trình tính toán chữ ký** bằng cách **giảm nguồn điện cấp cho bộ xử lý**.
* Các lỗi này khiến phần mềm **tạo ra các chữ ký không hợp lệ**, và **kẻ tấn công có thể phân tích những chữ ký sai đó** để **phục hồi khóa bí mật (private key)**.
* Thuật toán tấn công **gây ra lỗi đơn bit** (single-bit errors) và quan sát kết quả đầu ra.
* Mặc dù đáng được xem xét, nhưng **kiểu tấn công này không được coi là mối đe dọa nghiêm trọng đối với RSA**.
* **Nó yêu cầu kẻ tấn công phải có quyền truy cập vật lý** vào máy mục tiêu và **có khả năng điều khiển trực tiếp nguồn điện cấp cho bộ xử lý**.

Kiểu tấn công CCA(Chosen Ciphertext Attack)

**Kẻ tấn công chọn một số bản mã (ciphertexts)** và sau đó nhận được các bản rõ (plaintexts) tương ứng, được giải mã bằng khóa bí mật của mục tiêu.

→ Do đó, **kẻ tấn công có thể chọn một bản rõ**, mã hóa nó bằng **khóa công khai của mục tiêu**, và sau đó **nhận lại bản rõ** bằng cách cho hệ thống giải mã bản mã đó bằng **khóa bí mật**.

**Kẻ tấn công khai thác các tính chất của RSA** và lựa chọn các khối dữ liệu sao cho, khi được xử lý bằng **khóa bí mật của mục tiêu**, sẽ **lộ ra thông tin cần thiết để phân tích mật mã** (cryptanalysis).

Để chống lại các kiểu tấn công như vậy, công ty **RSA Security Inc.** khuyến nghị nên **biến đổi bản rõ** trước khi mã hóa bằng một kỹ thuật gọi là **đệm mã hóa bất đối xứng tối ưu (OAEP – Optimal Asymmetric Encryption Padding)**.

Những hiểu lầm về **Mã hóa Khóa Công khai (Public Key Cryptography)** thường xuất phát từ việc chưa nắm rõ bản chất và giới hạn của phương pháp này so với **Mã hóa Đối xứng (Symmetric Encryption)**. Dưới đây là giải thích chi tiết cho từng quan niệm sai lầm:

1. **"Mã hóa khóa công khai an toàn hơn trước các cuộc tấn công phân tích mật mã so với mã hóa đối xứng"**

* **Hiểu lầm**: Nhiều người nghĩ rằng mã hóa khóa công khai (ví dụ: RSA, ECC) có khả năng chống lại mọi tấn công toán học tốt hơn so với mã hóa đối xứng (ví dụ: AES, ChaCha20).
* **Thực tế**:
  + Mã hóa khóa công khai dựa trên các bài toán khó (như phân tích số nguyên tố lớn hoặc logarit rời rạc), nhưng chúng **dễ bị tấn công bởi các thuật toán lượng tử** (ví dụ: Shor's Algorithm).
  + Trong khi đó, mã hóa đối xứng (như AES-256) vẫn được coi là an toàn ngay cả trước máy tính lượng tử, với kích thước khóa đủ lớn và thuật toán mạnh.
  + **Tốc độ xử lý**: Mã hóa khóa công khai chậm hơn hàng trăm/thậm chí nghìn lần so với mã hóa đối xứng, nên thực tế chúng thường chỉ dùng để trao đổi khóa (như trong TLS), còn dữ liệu được mã hóa bằng khóa đối xứng.

2. **"Mã hóa khóa công khai khiến mã hóa đối xứng trở nên lỗi thời"**

* **Hiểu lầm**: Một số người tin rằng mã hóa khóa công khai có thể thay thế hoàn toàn mã hóa đối xứng.
* **Thực tế**:
  + **Bổ sung chứ không thay thế**: Cả hai phương pháp có vai trò khác nhau và thường được kết hợp. Ví dụ:
    - **Khóa công khai** dùng để trao đổi khóa bí mật (như trong Diffie-Hellman hoặc RSA-KEM).
    - **Khóa đối xứng** (như AES) dùng để mã hóa dữ liệu hàng loạt do hiệu suất cao.
  + **Ứng dụng thực tế**: Hầu hết giao thức bảo mật (TLS, PGP, Signal) đều dùng kết hợp cả hai: khóa công khai cho xác thực/trao đổi khóa, còn khóa đối xứng mã hóa dữ liệu.

3. **"Phân phối khóa trong mã hóa khóa công khai đơn giản hơn so với mã hóa đối xứng"**

* **Hiểu lầm**: Người ta nghĩ rằng việc dùng khóa công khai (public key) loại bỏ hoàn toàn vấn đề phân phối khóa phức tạp của mã hóa đối xứng.
* **Thực tế**:
  + **Vẫn cần xác thực khóa công khai**: Dù không cần giữ bí mật khóa công khai, bạn phải đảm bảo rằng khóa đó **thực sự thuộc về người/đối tượng mong muốn** (tránh tấn công MITM).
    - Ví dụ: Trong HTTPS, bạn cần CA (Certificate Authority) để xác thực khóa công khai của server.
  + **Mã hóa đối xứng cũng có giải pháp phân phối khóa hiện đại**:
    - Các giao thức như **Diffie-Hellman Ephemeral (DHE)** hoặc **Key Derivation Functions (KDF)** giúp trao đổi khóa đối xứng an toàn mà không cần trung tâm phân phối.