

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



NGUYỄN THỊ MỸ DUYÊN
NGUYỄN VĂN QUỐC GIA

ĐỒ ÁN MÔN HỌC

PHÂN TÍCH VÀ THỰC NGHIỆM
RSA VÀ CÁC PHIÊN BẢN CHỮ KÝ SỐ DỰA TRÊN RSA

**CRYPTANALYSIS ON ASYMMETRIC CIPHERS:
RSA & RSA BASED SIGNATURES**

MÔN: MẬT MÃ HỌC
NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2026

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**NGUYỄN THỊ MỸ DUYÊN – 24520408
NGUYỄN VĂN QUỐC GIA – 24520415**

ĐỒ ÁN MÔN HỌC

**PHÂN TÍCH VÀ THỰC NGHIỆM
RSA VÀ CÁC PHIÊN BẢN CHỮ KÝ SỐ DỰA TRÊN RSA**

**CRYPTANALYSIS ON ASYMMETRIC CIPHERS:
RSA & RSA BASED SIGNATURES**

**MÔN: MẬT MÃ HỌC
NGÀNH AN TOÀN THÔNG TIN**

**GIẢNG VIÊN HƯỚNG DẪN
TS. Nguyễn Ngọc Tự**

TP. HỒ CHÍ MINH, 2026

LỜI CẢM ƠN

Để hoàn thành đồ án môn học Mật mã học này, chúng em xin gửi lời tri ân sâu sắc đến những người đã luôn hỗ trợ, quan tâm và giúp đỡ chúng em trong suốt quá trình học tập và nghiên cứu.

Trước tiên, chúng em xin gửi lời cảm ơn chân thành nhất đến Thầy Nguyễn Ngọc Tự, giảng viên hướng dẫn môn học. Cảm ơn Thầy đã tận tình truyền đạt những kiến thức nền tảng quý báu về mật mã học, cũng như đưa ra những định hướng nghiên cứu thiết thực, giúp chúng em tiếp cận được với những vấn đề chuyên sâu và thực tế về phân tích mã hóa RSA. Những nhận xét và góp ý của Thầy là kim chỉ nam giúp chúng em hoàn thiện đồ án này một cách chu đáo nhất.

Chúng em cũng xin gửi lời cảm ơn đến Ban Giám hiệu và Quý Thầy Cô Trường Đại học Công nghệ Thông tin (UIT) đã tạo điều kiện môi trường học tập tốt nhất, cung cấp đầy đủ tài liệu và cơ sở vật chất để chúng em có thể thực hiện đề tài nghiên cứu này.

Mặc dù đã nỗ lực hết mình để tìm hiểu và hoàn thiện đồ án, nhưng do giới hạn về mặt thời gian và kiến thức, báo cáo khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được những đóng góp ý kiến để đề tài được hoàn thiện hơn nữa.

Chúng em xin chân thành cảm ơn!

Nhóm tác giả

TÓM TẮT

Trong bối cảnh an toàn thông tin hiện đại, hệ mật mã RSA và các chữ ký số dựa trên RSA đóng vai trò xương sống trong việc bảo vệ dữ liệu và xác thực giao tiếp trên Internet, đặc biệt là trong các giao thức như TLS/SSL. Tuy nhiên, sự phát triển của các kỹ thuật thám mã và năng lực tính toán ngày càng tăng đặt ra nhiều thách thức đối với tính bảo mật của hệ mật mã này. Đề án "Cryptanalysis on Asymmetric Ciphers: RSA RSA-Based Signatures" tập trung nghiên cứu sâu về cơ sở lý thuyết toán học và đặc biệt là phân tích các điểm yếu trong quá trình triển khai thực tế của RSA.

Nội dung đề án bao gồm việc khảo sát hệ thống các lớp tấn công phổ biến vào RSA, từ các tấn công toán học thuần túy đến các tấn công kênh kề (Side-channel attacks) và tấn công dựa trên lỗi triển khai (Padding Oracle, Fault attacks). Nhóm thực hiện đã tiến hành phân tích chi tiết và xây dựng các kịch bản PoC (Proof of Concept) cho các tấn công nguy hiểm như Bleichenbacher, Marvin Attack, ... để minh chứng cho các lỗ hổng lý thuyết.

Bên cạnh đó, đề án đi sâu vào phân tích các rủi ro bảo mật khi triển khai RSA trong các ứng dụng thực tế như giao thức TLS (phiên bản 1.2 và 1.3), cơ chế xác thực JWT, và quy trình Code Signing. Kết quả nghiên cứu chỉ ra rằng phần lớn các lỗ hổng nghiêm trọng không xuất phát từ thuật toán toán học của RSA, mà bắt nguồn từ những sai sót trong cấu hình (padding yếu, RNG kém chất lượng) và lập trình (không đảm bảo constant-time, rò rỉ oracle).

Từ những phân tích trên, đề án đề xuất các biện pháp phòng chống và khắc phục cụ thể, bao gồm việc chuyển đổi sang các chuẩn padding an toàn, áp dụng các biện pháp bảo vệ phần cứng, và tuân thủ các checklist triển khai an toàn để đảm bảo tính bảo mật lâu dài cho hệ thống.

MỤC LỤC

Lời cảm ơn	i
Tóm tắt	ii
Mục lục	iii
Danh mục các bảng	vi
Danh mục các hình vẽ và đồ thị	vii
Danh mục từ viết tắt	viii
Chương 1. Giới thiệu	1
Chương 2. Lý thuyết cơ bản về RSA và chữ ký số RSA	2
2.1 Lý thuyết cơ bản về RSA	2
2.1.1 Tạo key	2
2.1.2 Mã hóa	3
2.1.3 Giải mã	3
2.1.3.1 Proof of Correctness	3
2.2 Chữ ký số RSA	4
2.3 RSA optimizations	5
2.3.1 Nhược điểm thời gian của RSA	5
2.3.2 RSA-CRT	5
2.4 Bài toán trong RSA	6
Chương 3. Các lớp tấn công phổ biến	8
3.1 Math	8
3.1.1 Factoring	8
3.1.2 Key generation weakness	9
3.1.2.1 Wiener on small private exponent attack	9
3.1.2.2 Håstad's attack on low exponents and common modulus scenarios	10
3.2 Padding/ CCA (Chosen ciphertext attack)	12

MỤC LỤC

3.2.1	Bleichenbacher on PKCS#1 v1.5 padding oracle attack	12
3.2.1.1	Tổng quan ngắn	12
3.2.1.2	PKCS#1 v1.5	13
3.2.1.3	Một số tính chất toán học dùng trong tấn công	13
3.2.1.4	Ý tưởng tấn công	14
3.2.1.5	Thuật toán tấn công	14
3.2.1.6	Áp dụng Bleichenbacher's Attack trong chữ kí số	16
3.2.1.7	Ý nghĩa thực tiễn	16
3.2.1.8	Giải pháp	17
3.2.2	Marvin's Attack	17
3.2.2.1	Tổng quan	17
3.2.2.2	Bản chất của Marvin's Attack	17
3.2.2.3	Vì sao Marvin's Attack đặc biệt nguy hiểm?	18
3.2.2.4	Ý nghĩa đối với triển khai RSA hiện đại	18
3.2.2.5	Kết luận cho Padding Oracle	18
3.2.3	Manger Attack	19
3.2.3.1	Padding OAEP	19
3.2.3.2	Ý tưởng tấn công	19
3.2.3.3	Kết luận	20
3.3	Fault attack	20
3.3.1	RSA-CRT Fault Attack	20
3.3.1.1	Nhắc lại RSA-CRT	20
3.3.1.2	Ý tưởng tấn công	21
3.3.1.3	Giải pháp	22
3.4	Side-channels Attack	22
3.4.1	Timing Attack on RSA	22
3.4.1.1	Ý tưởng tấn công	22
3.4.1.2	Giải pháp	23
Chương 4.	Proof of Concept (PoC)	24
Chương 5.	Phân tích điểm yếu trong các triển khai thực tế	25
5.1	Transport Layer Security - TLS	25
5.1.1	TLS Handshake	25

MỤC LỤC

5.1.2	Những weaknesses về RSA trong TLS 1.2:	26
5.1.2.1	Không forward secrecy	26
5.1.2.2	Kịch bản Bleichenbacher Attack khi hệ thống hỗ trợ TLS 1.2, không hỗ trợ TLS 1.3	26
5.2	JWT / token signing (RS256)	27
5.3	Code signing & package ecosystems	28
5.3.1	Key exposure	28
5.3.2	Legacy signature format	29
5.4	Smartcards / HSMs / TPMs	29
5.5	Random Number Generator - RNG	30
5.5.1	RNG và phân loại	30
5.5.2	Sử dụng RNG trong RSA	31
5.6	Implementation bugs & side channels	31
5.7	Các kịch bản triển khai thực hiện	33
5.8	Bảng tổng hợp giải pháp triển khai	34
5.9	Checklist lưu ý khi triển khai RSA	37
Chương 6. Kết luận		38
Tài liệu tham khảo		39

DANH MỤC CÁC BẢNG

4.1	Tổng hợp các PoC attack đã triển khai	24
5.1	Các kịch bản triển khai tấn công thực tế	33
5.2	Tổng hợp giải pháp khi triển khai	36

DANH MỤC CÁC HÌNH VẼ VÀ ĐỒ THỊ

5.1	Quy trình TLS Handshake	25
-----	-----------------------------------	----

DANH MỤC TỪ VIẾT TẮT

Viết tắt	Tên đầy đủ (Tiếng Anh/Việt)
API	Application Programming Interface (Giao diện lập trình ứng dụng)
CI/CD	Continuous Integration/Continuous Delivery
CRT	Chinese Remainder Theorem (Định lý thặng dư Trung Hoa)
DPA	Differential Power Analysis (Phân tích điện năng vi sai)
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
HMAC	Keyed-Hash Message Authentication Code
HSM	Hardware Security Module (Mô-đun bảo mật phần cứng)
JWT	JSON Web Token
OAEP	Optimal Asymmetric Encryption Padding
PoC	Proof of Concept (Bằng chứng khái niệm)
PRNG	Pseudo Random Number Generator (Bộ sinh số giả ngẫu nhiên)
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator (Bộ sinh số ngẫu nhiên)
RSA	Rivest–Shamir–Adleman
SPA	Simple Power Analysis (Phân tích điện năng đơn giản)
TLS	Transport Layer Security (Bảo mật tầng giao vận)
TPM	Trusted Platform Module
TRNG	True Random Number Generator (Bộ sinh số ngẫu nhiên thực)

Chương 1. GIỚI THIỆU

Hiện nay, RSA vẫn là một trong những thuật toán khóa công khai được sử dụng rộng rãi nhất, đặc biệt trong các giao thức TLS/SSL, trao đổi khóa và chữ ký số. Nền tảng an toàn của mã hóa RSA dựa trên độ khó tính toán của bài toán phân tích các số nguyên lớn thành các thừa số nguyên tố của chúng. Đối với máy tính cổ điển, thuật toán hiệu quả nhất hiện nay được biết đến cho bài toán này là General Number Field Sieve (GNFS).

Tuy nhiên, thuật toán này vẫn không khả thi đối với các khóa có độ dài 2048 bit hoặc lớn hơn. Do đó, nếu được triển khai đúng cách, RSA vẫn được xem là an toàn ở thời điểm hiện tại. Mặc dù vậy, các mối đe dọa đối với RSA đang **gia tăng**. Một số vấn đề phát sinh từ các lỗi trong quá trình **triển khai** hoặc từ các **bộ sinh số ngẫu nhiên yếu**, dẫn đến việc nhiều khóa RSA bị xâm phạm do dùng chung các thừa số nguyên tố. Ngoài ra, các tấn công kênh kề (**side-channel attacks**) như phân tích thời gian, phân tích mức tiêu thụ điện năng và phân tích bức xạ điện từ có thể làm rò rỉ khóa bí mật trong các môi trường phần cứng.

Hơn nữa, sự xuất hiện của **máy tính lượng tử** đặt ra một mối đe dọa lớn đối với RSA. **Thuật toán Shor** cho thấy rằng nếu tồn tại một máy tính lượng tử đủ lớn và ổn định, việc phân tích các số nguyên lớn — và do đó phá vỡ RSA — có thể được thực hiện trong một khoảng thời gian khả thi trên thực tế.

Do đó, việc hiểu rõ **các khái niệm nền tảng** và **các lý thuyết đằng sau các tấn công lên RSA** giúp các tổ chức đánh giá rủi ro và thận trọng hơn khi triển khai RSA trong các dự án của mình.

Chương 2. LÝ THUYẾT CƠ BẢN VỀ RSA VÀ CHỮ KÝ SỐ RSA

2.1 Lý thuyết cơ bản về RSA

Mã hóa RSA sử dụng hai khóa: một khóa công khai và khóa bí mật (tương ứng với khóa công khai, tức một công khai thì chỉ có một khóa bí mật tương ứng). Khóa công khai được sử dụng cho mã hóa và không cần phải giữ bí mật, trong khi khóa bí mật được dùng để giải mã và cần phải giữ bí mật. Tức là ai cũng có thể mã hóa thông tin nhưng chỉ có người sở hữu khóa bí mật tương ứng mới có thể giải mã những đoạn mã đó.

2.1.1 Tạo key

Theo truyền thống trong lý thuyết mật mã, trong văn bản chúng tôi sử dụng Alice và Bob để nói về hai người muốn nói chuyện với nhau qua Internet, còn Eve là “woman-in-the-middle”, tức là người có thể nghe lén cuộc trò chuyện của Alice và Bob.

Đầu tiên, Alice tạo khóa với các bước như sau:

- Chọn hai số nguyên tố lớn p và q
- Tính $N = p \cdot q$
- Tính $\varphi(N) = (p - 1) \cdot (q - 1)$
- Chọn số nguyên dương e thỏa:

$$\diamond e \in (1, \varphi(N))$$

$$\diamond \gcd(e, \varphi(N)) = 1$$

- Tính d thỏa $d \cdot e \equiv 1 \pmod{\varphi(N)}$, tức d là nghịch đảo module $\varphi(N)$ của e

Khóa công khai và khóa bí mật lần lượt là:

- Khóa công khai: (N, e)
- Khóa bí mật: (N, d)

Sau khi tạo khóa, Alice gửi khóa công khai (N, e) đến Bob (chú ý rằng Eve có thể đọc được).

2.1.2 Mã hóa

Bob muốn mã hóa message M :

- Encode M thành số nguyên m
- Sử dụng khóa công khai (N, e) để tính ciphertext như sau:

$$ct \equiv m^e \pmod{N}$$

Lúc này, message M được mã hóa thành ct . Bob gửi ct cho Alice.

2.1.3 Giải mã

Alice nhận ciphertext ct và tiến hành giải mã nó như sau:

- Tính $m \equiv ct^d \pmod{N}$
- Decode m về message M

2.1.3.1 Proof of Correctness

Ở đây, ta sẽ chứng minh tại sao với các điều kiện trong phần **mã hóa** thì:

$$ct \equiv m^e \pmod{N} \text{ suy ra } m \equiv ct^d \pmod{N}$$

Có:

$$d \cdot e \equiv 1 \pmod{(p-1) \cdot (q-1)}$$

$$\Rightarrow \begin{cases} d \cdot e \equiv 1 \pmod{p-1} \\ d \cdot e \equiv 1 \pmod{q-1} \end{cases}$$

Xét $d \cdot e \equiv 1 \pmod{p-1}$:

$$\Rightarrow d \cdot e = 1 + k \cdot (p-1)$$

$$\Rightarrow ct^d = (m^e)^d = m^{e \cdot d} = m^{1+k \cdot (p-1)} = m \cdot (m^{p-1})^k \equiv m \pmod{p}$$

vì định lý **Fermat** bé

$$a^{p-1} \equiv 1 \pmod{p} \text{ với } \gcd(a, p) = 1$$

Tương tự ta cũng có được:

$$ct^d \equiv m \pmod{q}$$

Theo định lý CRT(Thặng dư Trung Hoa) ta suy ra:

$$ct^d \equiv m \pmod{(p \cdot q = N)}$$

2.2 Chữ ký số RSA

Ngoài mục đích mã hóa, RSA còn được ứng dụng cho chữ ký số. Cách hoạt động của nó về ý tưởng cũng giống như mã hóa thông tin nhưng bây giờ thứ tự sử dụng private key và public key là ngược lại.

Giả sử Alice muốn gửi cho Bob một đoạn văn bản kèm theo chữ ký của mình. Trong trường hợp này, Alice giữ cho mình một private key (N, d) , còn Bob giữ public key (N, e) tương ứng.

- Alice tính giá trị băm của toàn bộ văn bản muốn gửi đi, gọi giá trị băm này là *hash*
- Chữ ký số của văn bản Alice muốn gửi đi được tính như sau:

$$sig \equiv hash^d \pmod{N}$$

- Alice gửi đi văn bản muốn gửi kèm theo chữ ký số được tính ở trên
- Khi Bob nhận được, Bob tính:

$$hash \equiv sig^e \pmod{N}$$

sau đó tính giá trị băm của văn bản nhận được, nếu giá trị này trùng với *hash* thì người gửi biết private key của Alice và văn bản không bị thay đổi từ lúc gửi đến khi Bob nhận.

2.3 RSA optimizations

2.3.1 Nhược điểm thời gian của RSA

Ta xét độ phức tạp thời gian của việc tính $a^b \bmod N$ với a, b, N là các số nguyên lớn:

- Để tính lũy thừa a^b , ta thường sử dụng thuật toán **square-and-multiply** với độ phức tạp $O(\log(b))$, chú ý độ phức tạp này chưa tính độ phức tạp thời gian của việc nhân hai số nguyên.
- Tính $a \cdot a \bmod N$ có nhiều thuật toán, nhưng độ phức tạp hiệu quả nhất hiện tại là tầm $O(n \log(n))$ với n là số bits của N , chú ý khi N là số nguyên lớn, dẫn đến số bits n của N là lớn

Suy ra: độ phức tạp thời gian toàn bộ của việc tính a^b là $O(\log(b) \cdot n \log(n))$ với n là số bits biểu diễn của a .

Ta thường chọn $e = 65537 = 2^{16} + 1$ vì khi đó:

- e vẫn thỏa tính chất $1 < e < N$ và $\gcd(e, \varphi(N)) = 1$ (chú ý 65537 là số nguyên tố nên điều ta cần chỉ là chọn p, q sao cho $65537 \nmid p-1$ và $65537 \nmid q-1$)
- $\log(65537) \approx 5$, rất nhỏ

Suy ra, việc mã hóa(tức tính $m^e \bmod N$) sẽ rất nhanh, đó cũng là lý do ta thường chọn $e = 65537$.

Tuy nhiên, khi xét đến giải mã, ta cần quan tâm những điều sau:

- d thường không được chọn trước và thường phụ thuộc vào $e = 65537$ và $\varphi(N) = (p-1) \cdot (q-1)$ ($d \equiv e^{-1} \bmod \varphi(N)$)
- Vì lý do trên do trên, d thường rất lớn, $d \approx \varphi(N) = p \cdot q - p - q + 1 \approx p \cdot q = N$

Vì thế, thời gian giải mã trong mã hóa RSA thường khá lớn.

2.3.2 RSA-CRT

RSA-CRT áp dụng the Chinese Remainder Theorem (CRT) để tối ưu thời gian trong việc giải mã, cụ thể như sau:

- Tạo key:

$$\diamond \text{ Tính } d_p \equiv e^{-1} \bmod p-1$$

◇ Tính $d_q \equiv e^{-1} \pmod{q-1}$

- Mã hóa: giống như RSA bình thường, $ct \equiv m^e \pmod{N}$

- Giải mã:

◇ Tính $m_p \equiv ct^{d_p} \pmod{p}$

◇ Tính $m_q \equiv ct^{d_q} \pmod{q}$

◇ Giải hệ phương trình đồng dư sau bằng CRT:

$$\begin{cases} m \equiv m_p \pmod{p} \\ m \equiv m_q \pmod{q} \end{cases}$$

◇ Nghiệm $m \pmod{(p \cdot q = N)}$ của hệ phương trình trên chính là plaintext

Phân tích độ phức tạp thời gian của RSA-CRT, RSA-CRT chia việc giải mã thành 2 lần tính $a^b \pmod{M}$ nhưng trong mỗi phép tính, modulo M nhỏ hơn khá nhiều so với tính trong một phép tính ($d_p \approx p \approx \sqrt{N}$).

Tuy nhiên, việc triển khai RSA-CRT cần có nhiều lưu ý vì có thể gây ra nhiều nhược điểm bảo mật mà kẻ tấn công có thể khai thác, điều này sẽ được nói rõ hơn trong phần **Fault attacks on RSA-CRT**.

2.4 Bài toán trong RSA

Bài toán cốt lõi trong RSA là:

Giải phương trình đồng dư ẩn x :

$$x^e \equiv ct \pmod{N}$$

với N là tích của hai số nguyên tố và $\gcd(e, \varphi(N)) = 1$

Để phá mã **toàn bộ** RSA thì cần phải giải được bài toán trên trong một thời gian thực tế.

Hiện nay, cách tối ưu nhất để giải bài toán trên là phân tích thừa số nguyên tố của số nguyên lớn (trong trường hợp RSA thì là N), nếu ta phân tích được thừa số nguyên tố của N thì ta sẽ có được p và q , lúc này tính được $\varphi(N) = (p-1) \cdot (q-1)$, từ đó tính được Private Key:

$$d \equiv e^{-1} \pmod{\varphi(N)}.$$

CHƯƠNG 2. LÝ THUYẾT CƠ BẢN VỀ RSA VÀ CHỮ KÝ SỐ RSA

Khi đó, nghiệm x cần tìm là:

$$x = ct^d \mod N.$$

Vì vậy: Việc giữ bí mật p và q là tiên quyết trong mã hóa RSA, việc này cũng quan trọng như việc giữ bí mật d . Rất nhiều cuộc tấn công mã hóa RSA đều nhắm vào việc tìm được p và q .

Chương 3. CÁC LỚP TẤN CÔNG PHỔ BIẾN

3.1 Math

Lớp tấn công này tấn công vào mặt toán học của RSA nếu như key có dạng đặc biệt hoặc không được triển khai đúng.

3.1.1 Factoring

Hiện nay có nhiều thuật toán dùng để factoring N ra đời, một trong những ý tưởng được sử dụng nhiều là:

Tìm hai số nguyên a và b thỏa mãn $a^2 \equiv b^2 \pmod{N}$ và $a \not\equiv \pm b \pmod{N}$

Khi tìm được hai số nguyên thỏa mãn như thế:

$$a^2 - b^2 = kN$$

$$(a - b)(a + b) = kN$$

Do p là một trong hai ước nguyên tố của $N \rightarrow (a - b)$ chia hết cho p hoặc $(a + b)$ chia hết cho p

Chú ý điều kiện $a \not\equiv \pm b \pmod{N}$ nên thừa số chia hết cho p đó sẽ không chia hết cho q

$$\rightarrow p = \gcd(a - b, N) \text{ hoặc } p = \gcd(a + b, N)$$

\rightarrow Factoring N thành công

Một trong những thuật toán sử dụng ý tưởng trên là thuật toán **GNFS**, là thuật toán nhanh nhất ở hiện tại dùng để factoring số nguyên lớn N (dạng tổng quát) với độ phức tạp:

$$L_N \left[\frac{1}{3}, \left(\frac{64}{9} \right)^{1/3} \right] = \exp \left(\left((64/9)^{1/3} + o(1) \right) (\ln N)^{1/3} (\ln \ln N)^{2/3} \right)$$

Với thuật toán này, số nguyên N 512 bits có thể bị factoring chỉ trong vài giờ, số nguyên N 1024 bits thì bị factoring trong nhiều năm. Hiện nay, nhiều hệ thống sử dụng RSA 2048 bits. Tuy nhiên, với khả năng về sự xuất hiện máy tính lượng tử trong nhiều năm tới thì khuyến nghị cần phải sử dụng RSA 3072 bits để an toàn.

CHƯƠNG 3. CÁC LỚP TẤN CÔNG PHỔ BIẾN

Tuy nhiên, nếu như hai thừa số nguyên tố p và q có dạng đặc biệt thì có thể factoring $N = pq$ nhanh hơn rất nhiều so với **GNFS**. Ví dụ, nếu như p là số nguyên tố thỏa $p - 1$ là một smooth number (là số có tất cả các ước nguyên tố của nó đều nhỏ) thì thuật toán **Pollard** $p - 1$ sẽ factoring N rất nhanh so với **GNFS**.

Vì thế, khi sinh số nguyên tố ngẫu nhiên cần kiểm tra để tránh các trường hợp đặc biệt này.

Thuật toán Shor là một thuật toán lượng tử giúp phân tích nhân tử một số nguyên ở dạng $N = p \cdot q$, với p và q là các số nguyên tố. Trong lý thuyết thì nếu **đủ qubit** thì bất kì RSA nào cũng có thể bị phá bằng thuật toán Shor này.

3.1.2 Key generation weakness

3.1.2.1 Wiener on small private exponent attack

Nhắc lại công thức RSA

RSA có:

- $(n = p \cdot q)$
- $(\varphi(n) = (p - 1)(q - 1))$
- $(e \cdot d \equiv 1 \pmod{\varphi(n)})$

Nghĩa là: $e \cdot d = 1 + k\varphi(n)$, với k là số nguyên dương.

Khi d quá nhỏ

Nếu d nhỏ, tức là:

$$d < \frac{n^{0.25}}{3}$$

thì **Wiener (1990)** chứng minh rằng có thể **tính lại** d từ (e, n) bằng **phân số liên tục (continued fractions)**.

Ý tưởng là:

- Do $(e \cdot d - k\varphi(n) = 1)$, nên:

$$\frac{e}{n} \approx \frac{k}{d}$$

- Từ đó, kẻ tấn công có thể dùng **phân số liên tục (continued fraction)** để tìm xấp xỉ của e/n , thử các cặp (k_i, d_i) để tìm ra d thật sự.

Hậu quả

Nếu tìm được d , thì:

- Kẻ tấn công **giải mã được mọi ciphertext**:

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

- Hoặc **ký giả mạo** các thông điệp RSA-signature hợp lệ.

Nguyên nhân thực tế có thể dẫn đến “*d* nhỏ”

- Hệ thống cố ý chọn *d* nhỏ để **tăng tốc giải mã** (vì giải mã $= c^d \bmod n$).
- Hoặc chọn *e* quá lớn \rightarrow khiến *d* nhỏ do $e \cdot d \equiv 1 \pmod{\varphi(n)}$.

Biện pháp phòng tránh

- Không bao giờ chọn $d < n^{0.25}$.
- Thực tế, hầu hết các hệ thống dùng:
 - ◇ $e = 65537$
 - ◇ *d* ngẫu nhiên đủ lớn (vì được sinh tự động từ hàm inverse mod).
- Hoặc dùng **RSA-CRT, RSASSA-PSS** để cải thiện tốc độ mà vẫn an toàn.

3.1.2.2 Håstad’s attack on low exponents and common modulus scenarios

Bối cảnh

RSA mã hóa một thông điệp *m* thành:

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

với *e* là **public exponent** (thường nhỏ, như 3 hoặc 5) và $n = pq$ là modulus.

Håstad’s Broadcast Attack tấn công vào điểm yếu sau:

Nếu *e* đủ nhỏ làm cho $m^e < n$ thì khi đó việc $\bmod n$ trong $c = m^e \bmod n$ không còn tác dụng, suy ra $m = \sqrt[e]{c}$.

Håstad’s Broadcast Attack

Giả định:

- Cùng một thông điệp *m* được gửi cho *e* **người nhận khác nhau**

CHƯƠNG 3. CÁC LỚP TÂN CÔNG PHỔ BIẾN

- Mỗi người có **modulus khác nhau** n_1, n_2, \dots, n_e
- Cùng exponent nhỏ e (ví dụ $e = 3$)
- Không có padding ngẫu nhiên (số nguyên m ở mỗi lần mã hóa là giống nhau)

Ta thu được:

$$c_i = m^e \pmod{n_i}$$

Nếu các n_i **pairwise coprime** (nguyên tố cùng nhau từng đôi một, chú ý điều này rất dễ xảy ra trong RSA khi n_i chỉ có hai ước nguyên tố lớn ngẫu nhiên), ta có thể dùng **Chinese Remainder Theorem (CRT)** để tính được C với:

$$C = m^e \pmod{N}, \quad \text{với } N = n_1 n_2 \dots n_e$$

Do $m^e < N$ vì $m < n_i \forall i \in [1, e]$, ta có thể lấy:

$$m = \sqrt[e]{C}$$

(nghĩa là căn bậc e trên số nguyên, **không modulo**).

Kết quả: attacker có thể khôi phục plaintext mà **không cần khóa bí mật**.

Common Modulus Attack

- Hai người dùng có **cùng modulus** n (do bị cấu hình sai hoặc chia sẻ cùng HSM).
- Nhưng dùng **khóa công khai khác nhau**: e_1, e_2
- Cùng mã hóa **chung một thông điệp** m .

Ta có:

$$c_1 = m^{e_1} \pmod{n}$$

$$c_2 = m^{e_2} \pmod{n}$$

Nếu $\gcd(e_1, e_2) = 1$ (chú ý điều này thường xảy ra nếu e_1 và e_2 khác nhau vì ta thường chọn e là số nguyên tố), ta có thể dùng **Extended Euclidean Algorithm** để tìm (a, b) sao cho:

$$ae_1 + be_2 = 1$$

Khi đó:

$$c_1^a \cdot c_2^b \equiv m \pmod{n}$$

Ta có thể tính:

$$m = (c_1^a \cdot c_2^b) \mod n$$

Nếu $b < 0$, giả sử $b = -k$ ($k > 0$), ta tính:

$$c_2^b \equiv (c_2^{-1})^k \mod n$$

(với $c_2^{-1} \mod n$ là nghịch đảo modulo n của c_2).

Kết quả: attacker khôi phục m mà không cần giải RSA.

Biện pháp phòng tránh

- **Không dùng RSA raw**, luôn thêm padding ngẫu nhiên như **OAEP**:
 - ◇ Khi đó m được tăng lên gần với n , hạn chế hoàn toàn khả năng $m^e < n$.
 - ◇ Đồng thời plaintext giống nhau nhưng số nguyên m mỗi lần mã hóa là khác nhau.
- Không dùng chung modulus giữa nhiều người.
- Dùng **exponent đủ lớn**.

3.2 Padding/ CCA (Chosen ciphertext attack)

Lớp tấn công này tấn công vào một số padding scheme trên RSA và việc cài đặt lỗi dẫn đến để lộ oracle để attacker có thể thực hiện Chosen Ciphertext Attack.

3.2.1 Bleichenbacher on PKCS#1 v1.5 padding oracle attack

3.2.1.1 Tổng quan ngắn

Bleichenbacher (1998) là cuộc tấn công **padding-oracle** nổi tiếng nhắm vào chuẩn **PKCS#1 v1.5** cho RSA (dùng trong nhiều giao thức cũ như SSL/TLS).

Ý tưởng chính: nếu bạn có **oracle** trả lời cho bạn biết “sau khi giải được (RSA-decrypt), padding có hợp lệ theo PKCS#1 v1.5 hay không” (yes/no), thì bằng cách gửi nhiều ciphertext tùy chỉnh bạn có **lần lượt rút hẹp khoảng giá trị** của plaintext gốc và cuối cùng phục hồi toàn bộ thông điệp — tất cả mà không cần biết khóa riêng.

3.2.1.2 PKCS#1 v1.5

Khi RSA dùng cho mã hóa theo PKCS#1 v1.5, plaintext (M) trước khi mã hóa có dạng:

$$EM = 0 \times 00 \ || \ 0 \times 02 \ || \ PS \ || \ 0 \times 00 \ || \ D$$

- 0×02 chỉ ra chế độ mã hóa (khác với 0×01 cho signatures),
- PS (padding string) là một chuỗi padding ngẫu nhiên (ít nhất 8 byte, không chứa byte 0×00),
- D là dữ liệu/khóa (message) thực sự.

Một plaintext/ciphertext được gọi là **PKCS conforming** khi nó thỏa đủ 3 yêu cầu sau:

- 2 byte đầu tiên của plaintext tương ứng lần lượt là 0×00 và 0×02 .
- Byte thứ 3 đến byte 10 không được là byte 0×00 .
- Tồn tại ít nhất một byte nằm sau byte thứ 10 là byte 0×00 .

3.2.1.3 Một số tính chất toán học dùng trong tấn công

Gọi k là số bytes biểu diễn của n . Đầu tiên ta sẽ nói về một số tính chất toán học sẽ được dùng trong tấn công này:

Tính chất 1

Với s bất kì thì: Nếu $c = m^e \pmod{n}$ thì plaintext tương ứng với ciphertext $c' = c \cdot s^e \pmod{n}$ là $m' = m \cdot s \pmod{n}$.

Thật vậy:

$$(c')^d \equiv (c \cdot s^e)^d \equiv c^d \cdot s^{ed} = m \cdot s \pmod{n}$$

Tính chất 2

Nếu plaintext m thỏa PKCS conforming thì:

$$2B \leq m \pmod{n} \leq 3B - 1 \quad \text{với } B = 2^{8(k-2)}$$

Chú ý: việc đặt $B = 2^{8(k-2)}$ sẽ được sử dụng xuyên suốt trong phần này.

3.2.1.4 Ý tưởng tấn công

Ý tưởng của Bleichenbacher on PKCS#1 v1.5 padding oracle attack như sau: Lần lượt tìm s thỏa $c \cdot s^e$ là một ciphertext PKCS conforming

$$\rightarrow 2B \leq ms \pmod{n} \leq 3B - 1$$

$$\rightarrow \text{Tồn tại số nguyên } r \text{ thỏa: } 2B \leq ms - rn \leq 3B - 1$$

$$\rightarrow \frac{2B + rn}{s} \leq m \leq \frac{3B - 1 + rn}{s}$$

Suy ra, ta giới hạn lại m chỉ còn thuộc một khoảng nhất định.

Ứng với mỗi s tìm được, ta sẽ tiếp tục giới hạn lại khoảng có thể có của m (kết hợp với các khoảng ta đã tìm được trước đó) cho đến khi khoảng có thể có của m chỉ còn là $[a, a]$, khi đó ta tìm được $m = a$.

Tuy nhiên, để **tối ưu số lần query gửi đến oracle**, ta cần một cách chọn s hiệu quả để khả năng bắt gặp s thỏa $c \cdot s^e$ là một ciphertext PKCS conforming là cao nhất.

3.2.1.5 Thuật toán tấn công

Gọi c_0 là ciphertext mà ta cần tấn công, m_0 là plaintext mà ta cần tìm. Gọi M_i là biến lưu tập hợp của những khoảng có thể chứa m_0 sau khi tìm được s_i thỏa mãn.

Bước 1: Tìm s_i thỏa mãn $c_0(s_i)^e$ là một ciphertext thỏa PKCS conforming Để tối ưu số lần query gửi đến oracle, tấn công này chia thành 3 trường hợp, mỗi trường hợp có một chiến thuật tìm s_i khác nhau.

a. Nếu $i = 1$

Tìm số nguyên nhỏ nhất $s_1 \geq \frac{n}{3B}$ thỏa ciphertext $c_0(s_1)^e$ là PKCS conforming.

Giải thích lí do:

Nếu $s_1 < \frac{n}{3B} \rightarrow m_0 \cdot s_1 < 3B \cdot \frac{n}{3B} = n$ (chú ý m_0 là plaintext PKCS conforming nên $2B \leq m_0 < 3B$).

\rightarrow Khi đó: việc \pmod{n} không có tác dụng.

$$\rightarrow 2B \leq m_0 \cdot s_1 < 3B (*)$$

Tuy nhiên điều này là không thể vì với $s_1 \geq 2$ và $2B \leq m_0 < 3B$ dẫn đến $m_0 \cdot s_1 \geq 4B > 3B$, ngược lại với (*).

CHƯƠNG 3. CÁC LỚP TẤN CÔNG PHỔ BIẾN

Suy ra, nếu $s_1 < \frac{n}{3B}$ thì $c_0(s_1)^e$ không thể là ciphertext PKCS conforming, vì thế $\frac{n}{3B}$ là mốc đầu tiên ta dùng để bắt đầu tìm s_1 .

b. Nếu $i > 1$ và M_i chứa nhiều hơn 1 khoảng

Tìm s_i nhỏ nhất $> s_{i-1}$ thỏa ciphertext $c_0 \cdot s_i^e$ là PKCS conforming.

Ta lần lượt tìm s_i tăng dần để tránh chọn lại những s thỏa mãn ta đã chọn trước đó.

c. Nếu $i > 1$ và M_i chỉ chứa một khoảng $[a, b]$

Lúc này, khi biết m_0 chỉ có thể nằm trong khoảng $[a, b]$, ta có chiến thuật chọn s_i khác để khả năng ciphertext $c_0 \cdot s_i^e$ thỏa PKCS conforming là cao.

Ta cần tìm s_i và r_i thỏa mãn $2B \leq m_0 s_i - r_i n \leq 3B - 1$, chú ý bây giờ ta đã biết được $a \leq m_0 \leq b$. Cố định r_i , từ hai bất đẳng thức trên ta suy ra được $\frac{2B+r_i n}{b} \leq s_i \leq \frac{3B+r_i n}{a}$ (thường thì khoảng này khá nhỏ).

Để $s_i \geq s_{i-1} + 1$ thì điều kiện đủ là $s_{\min}(r) \geq s_{i-1} + 1$. Suy ra: $\frac{2B+r_i n}{b} \geq s_{i-1} + 1$.

→ Biến đổi bất đẳng thức trên, ta được $r_i \geq \frac{b(s_{i-1}+1)-2B}{n}$.

Như vậy, nếu $i > 1$ và M_i chỉ chứa một khoảng $[a, b]$ thì:

Tìm s_i và r_i thỏa mãn:

- $r_i \geq \frac{b(s_{i-1}+1)-2B}{n}$
- $\frac{2B+r_i n}{b} \leq s_i \leq \frac{3B+r_i n}{a}$ (thường thì khoảng này khá nhỏ)

Bước 2: Thu hẹp M_i Sau khi s_i được tìm thấy, tập M_i được tính như sau:

$$M_i \leftarrow \bigcup_{(a,b,r)} \left\{ \left[\max\left(a, \left\lceil \frac{2B+rn}{s_i} \right\rceil\right), \min\left(b, \left\lfloor \frac{3B-1+rn}{s_i} \right\rfloor\right) \right] \right\}$$

với mọi $([a, b] \in M_{i-1})$ và mọi r thỏa:

$$\left\lceil \frac{as_i - 3B + 1}{n} \right\rceil \leq r \leq \left\lfloor \frac{bs_i - 2B}{n} \right\rfloor.$$

Giải thích ngắn: Với mỗi khoảng $[a, b]$ trong M_{i-1} và mỗi giá trị nguyên r trong đoạn trên, ta xét khoảng nguyên

$$\left[\left\lceil \frac{2B+rn}{s_i} \right\rceil, \left\lfloor \frac{3B-1+rn}{s_i} \right\rfloor \right]$$

và lấy giao của nó với $[a, b]$. Tập hợp tất cả các giao không rỗng thu được chính là M_i .

Bước 3: Tính kết quả

- Nếu M_i chỉ còn chứa một khoảng $[a, a]$ thì m_0 cần tìm chính là a .
- Ngược lại, quay về bước 1.

3.2.1.6 Áp dụng Bleichenbacher's Attack trong chữ ký số

Bây giờ ta nói về việc phân tích áp dụng **Bleichenbacher's Attack** vào việc chữ ký số RSA như sau:

Attacker cần ký một message có hash là h . Tức vấn đề bây giờ đặt ra là: liệu attacker có thể tìm được h^d nếu attacker có một oracle cho phép nhận biết một ciphertext có là PKCS conforming hay không.

Câu trả lời là có. Để tìm h^d , attacker thực hiện như sau:

- Tìm s_0 thỏa hs_0^e là một ciphertext PKCS conforming.
- Đặt $c_0 = hs_0^e \pmod{n}$.
- Thực hiện Bleichenbacher's Attack như trên \rightarrow attacker tìm được $c_0^d = (hs_0^e)^d = h^d \cdot s_0^{ed} = h^d \cdot s_0 \pmod{n}$.
- Attacker tính $h^d = c_0^d \cdot s_0^{-1} \pmod{n}$ với $s_0^{-1} \pmod{n}$ là nghịch đảo module n của s_0 .

Trong ngữ cảnh chữ ký số thì khó có khả năng tồn tại oracle cho phép nhận biết một ciphertext có là PKCS conforming hay không. Vì thế nên nhìn chung việc sử dụng chữ ký số RSA-PKCS#1 v1.5 không nguy hiểm như RSA-PKCS#1 v1.5 key exchange.

3.2.1.7 Ý nghĩa thực tiễn

Bleichenbacher chỉ ra một bài học quan trọng:

RSA không an toàn nếu padding không được thiết kế để chống chosen-ciphertext attack.

Chuẩn PKCS#1 v1.5:

- Không có cơ chế ngẫu nhiên đủ mạnh để che đi cấu trúc plaintext.

- Không đảm bảo an toàn trong mô hình CCA.
- Và không thể sửa triệt để chỉ bằng vá lỗi triển khai.

3.2.1.8 Giải pháp

1. **Ngưng sử dụng PKCS#1 v1.5 cho mã hóa:** Dùng **RSA-OAEP** (được chứng minh an toàn trong mô hình CCA).
2. **Đối với chữ ký:** Dùng **RSA-PSS** thay vì v1.5.
3. **Triển khai an toàn:**
 - Constant-time xử lý.
 - Không phân biệt lỗi.
 - Không để lộ bất kỳ oracle nào (kể cả gián tiếp).

3.2.2 Marvin's Attack

3.2.2.1 Tổng quan

Marvin's Attack (được công bố và khai thác thực tế nhiều năm sau Bleichenbacher) cho thấy rằng:

Ngay cả khi hệ thống không trả về oracle rõ ràng, RSA với PKCS#1 v1.5 vẫn có thể bị phá thông qua side-channel timing.

Nói cách khác:

- Bleichenbacher cần **explicit oracle** (error message).
- Marvin's Attack tạo ra **implicit oracle** từ **thời gian phản hồi**.

3.2.2.2 Bản chất của Marvin's Attack

Trong nhiều triển khai RSA:

- Padding đúng \rightarrow hệ thống tiếp tục xử lý (sinh key, handshake, HMAC, ...).
- Padding sai \rightarrow hệ thống dừng sớm.

Sự khác biệt này dẫn đến:

- **Chênh lệch thời gian xử lý có thể đo được.**
- Dù không có bất kỳ thông báo lỗi nào trả về cho attacker.

Marvin's Attack biến:

Thời gian phản hồi --> Padding đúng / sai

và từ đó:

- Xây dựng lại **padding oracle**.
- Áp dụng lại toàn bộ logic của Bleichenbacher.

3.2.2.3 Vì sao Marvin's Attack đặc biệt nguy hiểm?

- Phá vỡ giả định “đã giấu error là an toàn”.
- Tấn công được cả những hệ thống:
 - ◇ Chỉ trả về một message chung chung.
 - ◇ Không log lỗi.
 - ◇ Không phân biệt trạng thái ở mức giao thức.

Điều này cho thấy:

Side-channel là một phần của oracle, không phải ngoại lệ.

3.2.2.4 Ý nghĩa đối với triển khai RSA hiện đại

Marvin's Attack khẳng định rằng:

- PKCS#1 v1.5 **không thể dùng an toàn**, kể cả khi “triển khai cẩn thận”.
- Các vá kiểu: Delay ngẫu nhiên, Che lỗi thủ công, Log ẩn... đều **không đủ mạnh** trước attacker có khả năng đo thời gian chính xác.

3.2.2.5 Kết luận cho Padding Oracle

- Error hiding không đồng nghĩa với an toàn.
- RSA PKCS#1 v1.5 không thể bảo vệ khỏi side-channel ở tầng triển khai.

- Constant-time không phải là tùy chọn, mà là bắt buộc.

Giải pháp:

- Chuyển sang RSA-OAEP (chống CCA theo thiết kế).
- Triển khai giải mã constant-time tuyệt đối.
- Không sử dụng RSA để mã hóa session key trong giao thức mới.
- Ưu tiên (EC)DHE trong TLS hiện đại.

3.2.3 Manger Attack

Tấn công này nhắm vào RSA-OAEP nếu không được cấu hình đúng và để lộ oracle.

3.2.3.1 Padding OAEP

Cấu trúc của padding OAEP có thể được tóm tắt như sau:

$EM = 0x00 \ || \ maskedSeed \ || \ maskedDB$

- $DB = llHash||PS||0x01||M$ với M là Message.
- $seed$ là chuỗi random.
- $maskedDB = DB \oplus MGF1(seed)$.
- $maskedSeed = seed \oplus MGF1(DB)$.

Manger Attack chỉ nhắm vào một điểm trong cấu trúc này chính là EM phải được bắt đầu bằng byte $0x00$.

3.2.3.2 Ý tưởng tấn công

Gọi k là số bytes biểu diễn của modulus.

Điều kiện để thực hiện tấn công là một oracle cho phép nhận ciphertext C và trả lời byte đầu tiên của plaintext tương ứng có phải là $0x00$ hay không. (điều này xảy ra khi cấu hình việc verify không đúng: phân biệt lỗi padding trả về).

Rõ hơn, oracle phải cho attacker biết liệu plaintext y (sau RSA decrypt) là: $y < B$ hay $y \geq B$ với $B = 2^{(8 \cdot (k-1))}$.

Nhắc lại: với m là plaintext tương ứng của ciphertext c thì plaintext tương ứng với ciphertext $s^e c$ là ms .

Ý tưởng của Manger attack để tìm plaintext m tương ứng với ciphertext c :

- Giả sử đã biết được $m \in [a, b]$.
- Với mỗi s , cho biết $ms < B$ hoặc $ms \geq B$:
 - ◇ Nếu $ms < B$, suy ra $m < B/s$.
 - ◇ Nếu $ms \geq B$, suy ra $m \geq B/s$.
- Cứ như thế, attacker giới hạn lại giá trị có thể có của m cho đến khi $a == b$, khi đó tìm được $m = a$.

3.2.3.3 Kết luận

Nếu implementation RSAES-OAEP leak thông tin phân biệt trong quá trình giải mã, một attacker có thể tận dụng oracle dạng “plaintext nhỏ hơn B hay không” để dần phục hồi plaintext gốc m thông qua **Manger attack** chỉ trong khoảng vài nghìn truy vấn.

3.3 Fault attack

Lớp tấn công này nhắm vào việc cố ý gây ra lỗi trên các thiết bị phần cứng để rò rỉ thông tin bí mật. Trong RSA, có một tấn công điển hình cho lớp tấn công này là RSA-CRT Fault Attack.

3.3.1 RSA-CRT Fault Attack

3.3.1.1 Nhắc lại RSA-CRT

Như đã nói ở phần trên, RSA-CRT thường được áp dụng để giảm thời gian giải mã hoặc ký so với RSA mặc định:

- **Key generation:**
 - ◇ Compute $d_p \equiv e^{-1} \pmod{p-1}$
 - ◇ Compute $d_q \equiv e^{-1} \pmod{q-1}$
- **Signature Generation:**
 - ◇ Compute $S_p \equiv h^{d_p} \pmod{p}$

- ◇ Compute $S_q \equiv h^{d_q} \pmod{q}$
- ◇ Solve the following system of congruences using CRT:

$$\begin{cases} S \equiv S_p \pmod{p} \\ S \equiv S_q \pmod{q} \end{cases}$$

- ◇ The solution $S \pmod{p \cdot q = N}$ is the Signature.

3.3.1.2 Ý tưởng tấn công

Có nhiều loại Fault Attack tấn công vào các hệ thống nhúng như là smart cards, một trong số đó là Fault Attack của Boneh-DeMillo-Lipton:

- Nếu như có Attacker có thể gây lỗi xảy ra ở chỉ một trong hai lần việc tính $S_p \equiv h^{d_p} \pmod{p}$ hoặc $S_q \equiv h^{d_q} \pmod{q}$, giả sử lỗi xảy ra ở S_p , kết quả tính được là $S'_p \neq S_p \pmod{n}$.
- Khi đó, sử dụng CRT để giải hệ phương trình:

$$\begin{cases} S' \equiv S'_p \pmod{p} \\ S' \equiv S_q \pmod{q} \end{cases}$$

- Signature tính được gọi là S' , nhận xét:
 - ◇ $S' \neq S \pmod{p}$
 - ◇ $S' \equiv S \pmod{q}$
- Suy ra, $S' - S$ chia hết cho q và không chia hết cho p .

Suy ra: $\gcd(S' - S, N) = q$, như vậy ta có thể phân tích thừa số nguyên tố của N .

Trong thực tế, Attacker có nhiều cách để gây lỗi trên các hệ thống nhúng như giả thuyết trên:

- Variations in Suply Voltage
- Variations in the external clock
- Temperature Variation

- White light
- ...

3.3.1.3 Giải pháp

Một giải pháp đơn giản nhưng hiệu quả để chống lại Boneh-DeMillo-Liton Fault Attack là **Verify After Sign**: Sau khi tính được chữ ký S , kiểm tra $S^e \equiv h \pmod{N}$, nếu đúng thì mới trả về S .

Thực tế có nhiều loại Fault Attack tấn công trên RSA-CRT và muốn an toàn thì cần kết hợp nhiều giải pháp lại với nhau chứ không chỉ thực hiện mỗi **Verify After Sign**.

3.4 Side-channels Attack

Lớp tấn công này khai thác các thông tin rò rỉ vật lý hoặc hành vi phụ của hệ thống trong quá trình thực thi.

3.4.1 Timing Attack on RSA

3.4.1.1 Ý tưởng tấn công

Timing Attack tấn công vào thuật toán **Square and Multiply** (dùng để tính lũy thừa module $a^b \pmod{N}$).

Mã giả của thuật toán **Square and Multiply** dùng để tính $h^d \pmod{n}$:

```
Square_and_Multiply(h, d, n){
    res = 1
    Duyệt các bit x của d, trọng số thấp đến cao:
        if x == 1:
            res = res * h
            h = h * h
    return res
}
```

→ Nhận xét: nếu bit đang xét của d là 1 thì trong lần lặp đó thực hiện 2 phép nhân, ngược lại nếu bit đang xét của d là 0 thì chỉ thực hiện 1 phép nhân. → Attacker có thể đo thời gian để lần lượt đoán các bits của d từ trọng số thấp đến cao, từ đó đoán được giá trị của d .

3.4.1.2 Giải pháp

a. Constant time

Thay đổi một phần thuật toán **Square-and-Multiply** sao cho dù bit đang xét là 1 hay là 0 thì nó vẫn thực hiện số phép tính như nhau và dùng mask để chọn kết quả mong muốn trả về.

Ví dụ:

```
Square_and_Multiply_ConstantTime(h, d, n):
    res = 1
    Duyệt các bit x của d, trọng số thấp đến cao:
        tmp1 = res
        tmp2 = res * h
        h = h * h
        res = tmp2 * (1 - x) + tmp1
    return res
```

→ Nhận xét trong mã giả trên, các phép tính được thực hiện là như nhau dù cho bit ‘x’ đang xét là 0 hay là 1 và cũng không sử dụng if else để chọn giá trị trả về → Về mặt lý thuyết thì mã giả trên đã tính được $h^d \pmod n$ constant time.

b. RSA-Blinding

Thay vì tính $m = c^d \pmod n$ một cách trực tiếp, ta áp dụng **RSA-Blinding** như sau:

- Tính $c' = c \cdot r^e \pmod n$ với r là một giá trị bất kì thỏa $\gcd(r, n) = 1$.
- Tính $m' = (c')^d = (c \cdot r^e)^d = c^d \cdot r = m \cdot r \pmod n$.
- Tính $m = m' \cdot r^{-1} \pmod n$.

→ Bây giờ, thời gian tính $m = c^d \pmod n$ phụ thuộc vào r , một giá trị random, khi đó Attacker khó có thể đoán được các bit của d từ việc đo thời gian.

Chương 4. PROOF OF CONCEPT (POC)

Phần này chúng tôi triển khai một số tấn công chính của các nhóm lỗi trên trên môi trường mô phỏng lỗi. Từ đó rút ra các nhận xét trên các tấn công này.

No.	Attack	Link PoC
1	Bleichenbacher Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/bleichenbacher_attack
2	Marvin Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/marvin_attack
3	Common Modulus Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/common_modulus_attack
4	Hastad Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/hastad_attack
5	Wiener Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/wiener_attack
6	RSA-CRT Fault Attack	https://github.com/QuocGia12/NT219-Project/tree/main/poc/RSA_CRT_fault_attack

Bảng 4.1. Tổng hợp các PoC attack đã triển khai

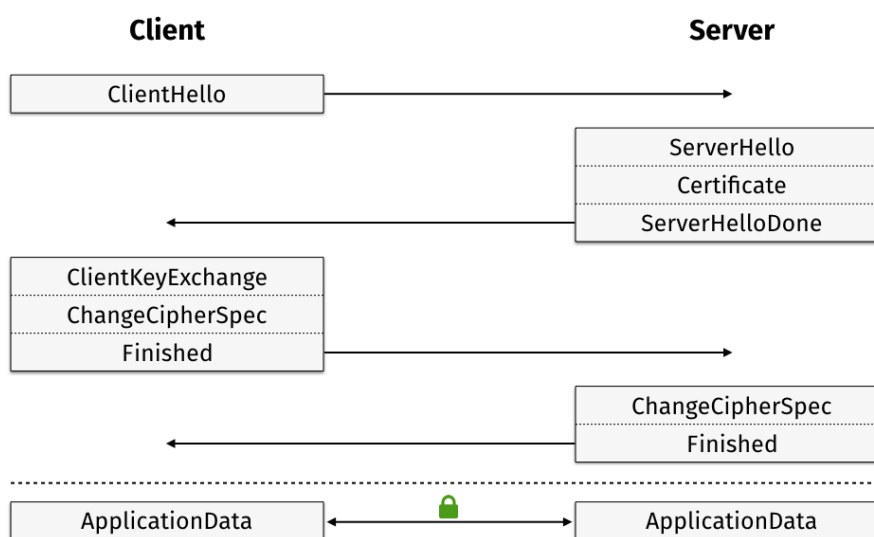
Chương 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

Chương này sẽ đi sâu phân tích các điểm yếu của RSA khi được áp dụng trong các môi trường thực tế, từ các giao thức truyền thông phổ biến như TLS, cơ chế xác thực JWT, đến các hệ sinh thái phần mềm (Code Signing) và thiết bị phần cứng (HSM/Smartcards). Đồng thời, chúng tôi cũng xem xét các nguy cơ đến từ những thành phần hỗ trợ như bộ sinh số ngẫu nhiên (RNG) và các lỗi lập trình dẫn đến tấn công kênh kề (Side-channel attacks). Mục tiêu của chương là làm rõ luận điểm: tính an toàn của RSA không chỉ phụ thuộc vào độ dài khóa, mà phụ thuộc tối mật thiết vào việc tuân thủ các chuẩn mực triển khai an toàn.

5.1 Transport Layer Security - TLS

5.1.1 TLS Handshake

TLS Handshake là bước thiết lập quan trọng cho một kết nối an toàn giữa client và server. Trong quá trình này, cả hai bên đồng ý về phiên bản TLS, cipher suite, xác thực server và tạo ra session key để mã hóa dữ liệu trong phiên đó.



Hình 5.1. Quy trình TLS Handshake

Các bước cơ bản trong quá trình TLS Handshake:

- Client gửi message **ClientHello** đến server, message này gồm các phiên bản TLS mà Client hỗ trợ, danh sách các loại mã hóa mà Client hỗ trợ, cùng với một số ngẫu nhiên **ClientRandom**.

CHƯƠNG 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

- Server gửi lại message ServerHello, message này chứa phiên bản TLS, loại mã hóa mà Server chọn trong danh sách gửi từ Client. Đồng thời cũng gồm Certificate của Server.
 - ◊ Certificate của Server dùng để xác thực Server cũng như chứa Public Key, Public Key này dùng để xác thực dữ liệu mà Server ký, từ đó Attacker không thể giả mạo server cũng như không thể thực hiện Man-in-the-Middle Attack
- Trao đổi khóa: Server và Client gửi cho nhau các tham số sử dụng trong loại mã hóa đã chọn, từ đó cả hai tính được PreMasterSecret. Sử dụng PreMasterSecret cùng với ClientRandom và ServerRandom, Client và Server tính MasterSecret, tức là Session Key. Toàn bộ dữ liệu trong phiên này sẽ được mã hóa bởi Session Key đó.

Chú ý một điều quan trọng là TLS 1.3 không còn sử dụng trao đổi khóa RSA nữa mà thay vào đó chỉ sử dụng ECDHE.

5.1.2 Những weaknesses về RSA trong TLS 1.2:

5.1.2.1 Không forward secrecy

Chú ý trong TLS 1.2, các tham số (n, e, d) sử dụng trong mã hóa RSA cũng chính là các tham số (n, e, d) trong Certificate của Server, vì thế nếu như Server không thay đổi Certificate thì các tham số (n, e, d) mà Server sử dụng trong các phiên khác nhau là như nhau. Dẫn đến nếu như Attacker lấy được d thì Attacker có thể giải mã được toàn bộ các phiên sử dụng trao đổi khóa RSA. → Không có tính forward secrecy

→ Vì thế trong TLS 1.3 đã loại bỏ hoàn toàn trao đổi khóa RSA, thay vào đó chỉ sử dụng ECDHE. Từ đó nếu Attacker có thể tấn công một phiên thì vẫn không giải mã được các phiên khác, vì thế đảm bảo tính forward secrecy

5.1.2.2 Kịch bản Bleichenbacher Attack khi hệ thống hỗ trợ TLS 1.2, không hỗ trợ TLS 1.3

- Attacker thực hiện Man-in-the-Middle Attack, thay đổi message ClientHello của Client sao cho danh sách Cipher Suite chỉ gồm RSA, ép Server phải chọn Cipher Suite là RSA.
- Sau đó, Attacker chặn gói tin ClientKeyExchange từ Client, xem Server là một oracle để thực hiện Bleichenbacher's Attack. Nếu như Server để lộ sự khác biệt thời gian

CHƯƠNG 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

về kết quả padding thì Attacker có thể lấy được PreMasterSecret, từ đó tính được Session Key, giải mã được toàn bộ gói tin gửi trong phiên đó.

Chú ý kịch bản tấn công này không thể sử dụng trong TLS 1.3 vì:

- Trong TLS 1.3, không hỗ trợ trao đổi khóa RSA
- Trong TLS 1.3, Server ký toàn bộ transcript trong quá trình bắt tay, sau đó gửi cho Client để Client xác nhận lại. Vì thế Attacker không thể thay đổi bất cứ điều gì trong quá trình bắt tay.

Tuy nhiên, mặc dù hệ thống có hỗ trợ TLS 1.3, nhưng nếu cấu hình máy chủ vẫn cho phép các phiên bản cũ hơn như TLS 1.2, attacker có thể thực hiện downgrade attack, đặc biệt khi cơ chế chống downgrade không được bật hoặc triển khai không đúng. Vì vậy, trong các hệ thống không yêu cầu tương thích ngược, khuyến nghị chỉ cho phép TLS 1.3 và vô hiệu hóa TLS 1.2 để giảm thiểu rủi ro bảo mật.

5.2 JWT / token signing (RS256)

JWT - JSON Web Token là một chuỗi dùng để xác thực Authentication và Authorization của một user.

JWT = <base64-encoded header>.<base64-encoded
payload>.<base64-encoded signature>

- header: chứa thuật toán ký, ví dụ "alg": "..."
- payload: thông tin dùng để xác thực Authentication và Authorization của user
- signature: chữ ký của header + payload, bảo vệ tính toàn vẹn của header và payload

Thuật toán ký sử dụng có thể chia làm 2 phần:

- HMAC → issuer và verifier cần chia sẻ với nhau trước secret key. Ví dụ: HS256 (HMAC with SHA-256)
- Chữ ký số → ký bằng private key, verify bằng public key. Ví dụ: RS256 (RSA with SHA-256)

Nếu như Server cấu hình việc Verify **không xác định rõ thuật toán verify mà tin vào alg của JWT từ client** thì có thể dẫn đến Token forgery:

CHƯƠNG 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

Kịch bản 1: Attacker tạo JWT giả mạo, gán "alg": "None", JWT = <base64-encoded header>.<base64-encoded payload>. → Nếu như Server vẫn tin vào "alg": "None" mà verify hợp lệ thì attacker có thể tạo bất kì JWT hợp lệ.
→ cần từ chối "alg": "None"

Kịch bản 2: Server sử dụng "alg" = "RS256" để ký nhưng khi verify lại không kiểm tra chặt điều kiện "alg" = "RS256", tin JWT của client.

Pseudo-code:

```
def verify(token):  
    header = parse_header(token)  
    key = load_key() # RSA public key  
  
    if header["alg"] == "RS256":  
        return rsa_verify(token, key)  
    elif header["alg"] == "HS256":  
        return hmac_verify(token, key)
```

Chú ý khi Server sử dụng "alg" = "RS256" thì Server verify bằng public key.

→ Attacker tạo một JWT giả mạo theo ý muốn rồi gán alg = HS256, ký bằng HS256 với secret key sử dụng là public key.

→ Server verify bằng **HS256** hợp lệ.

→ Yêu cầu: cần phải xác định rõ alg khi verify.

5.3 Code signing & package ecosystems

5.3.1 Key exposure

- **Phân tích:** Môi trường CI/CD (DevOps) thường là mắt xích yếu nhất. Nếu private key được lưu dưới dạng file (ví dụ: .pem) trong code repo hoặc biến môi trường không được mã hóa, kẻ tấn công xâm nhập được vào build server sẽ copy được key này.
- **Tác động:** Kẻ tấn công có thể tự ký (self-sign) các token hợp lệ (Forged Tokens) với bất kỳ quyền hạn nào (Admin privilege escalation). Đây là kịch bản "Golden Token".

- **Khắc phục:** Không bao giờ lưu private key trên đĩa cứng của server ứng dụng. Sử dụng **HSM (Hardware Security Module)** hoặc các dịch vụ quản lý khóa đám mây (như AWS KMS, Azure Key Vault) để thực hiện thao tác ký. Private key không bao giờ rời khỏi thiết bị bảo mật.

5.3.2 Legacy signature format

Legacy signature format là các lược đồ hoặc định dạng chữ ký lỗi thời cho phép một message có thể có nhiều chữ ký hợp lệ khác nhau, dù được ký bằng cùng một private key. Bên cạnh đó, từ một chữ ký hợp lệ cũng có thể tính được một chữ ký hợp lệ khác tương ứng với message đó một cách dễ dàng.

→ **Signature malleability:** message không thay đổi, chữ ký thay đổi nhưng khi verify vẫn hợp lệ.

Trong thực tế, nhiều hệ thống cần gán một hành động với một ID để ngăn chặn Attacker thực hiện Replay Attack. Trước khi thực hiện một hành động, kiểm tra ID đó tồn tại chưa rồi mới thực hiện. Nếu như hệ thống sử dụng chữ ký số làm ID, Attacker có thể tính một chữ ký hợp lệ khác tương ứng với hành động đó (Signature malleability) rồi thực hiện Replay Attack.

→ Cần tách biệt chữ ký với ID:

- Signature: kiểm tra tính toàn vẹn và xác thực nguồn gốc của message.
- ID: định danh hành động (thường dùng một số random Nonce).
- Bên cạnh đó, signature cần ký luôn cả ID.

Để chống **Signature malleability**, chỉ xác thực chữ ký ở một format nhất định, không verify hợp lệ những chữ ký ở format khác.

5.4 Smartcards / HSMs / TPMs

Khi RSA được đẩy xuống phần cứng, các vector tấn công chuyển từ phần mềm sang vật lý và API.

- **Side-channel & Fault attacks:**

- ◊ **Phân tích:**

- * **Power Analysis (DPA/SPA):** Khi chip thực hiện tính toán RSA, lượng điện năng tiêu thụ thay đổi tùy thuộc vào việc nó đang xử lý bit 0 hay bit 1 của khóa bí mật. Kẻ tấn công đo đặc dao động điện năng này để tái tạo lại chuỗi bit của private key.
- * **Fault Injection (RSA-CRT):** Kẻ tấn công sử dụng tia laser hoặc dao động điện áp để gây lỗi trong quá trình tính toán CRT (Chinese Remainder Theorem). Chỉ cần 1 lỗi tính toán duy nhất cũng đủ để lộ hoàn toàn một thừa số nguyên tố (p hoặc q).
- ◇ **Khắc phục:** Sử dụng smartcard đạt chuẩn FIPS 140-2 Level 3+ có lớp bảo vệ vật lý (mesh shield). Về thuật toán, luôn thực hiện verify chữ ký ngay trên chip trước khi xuất kết quả ra ngoài. Nếu verify sai, không trả về kết quả lỗi (để tránh lộ manh mối) mà hủy phiên làm việc.
- **API misuse (PKCS#11 attacks):**
 - ◇ **Phân tích:** HSM thường giao tiếp qua chuẩn PKCS#11. Nếu API không được phân quyền chặt (ACL), một ứng dụng bị xâm nhập có thể yêu cầu HSM giải mã bất cứ thứ gì.
 - ◇ **Bleichenbacher Oracle via HSM:** Nếu HSM trả về các mã lỗi khác nhau cho "sai khóa" và "sai padding" khi giải mã RSA, kẻ tấn công có thể lợi dụng HSM như một Oracle để giải mã tin nhắn mà không cần trích xuất private key.
 - ◇ **Khắc phục:** Áp dụng **Key Wrapping** (chỉ cho phép dùng key này để wrap key khác, không cho phép decrypt dữ liệu thô). Giới hạn quyền sử dụng key (Usage Flags) chỉ cho phép Sign hoặc Decrypt cụ thể, không được bật cả hai cho cùng một key.

5.5 Random Number Generator - RNG

5.5.1 RNG và phân loại

RNG là công cụ dùng để tạo số ngẫu nhiên không thể dự đoán.

Phân loại:

- **TRNG (True Random Number Generator):** Tạo số ngẫu nhiên thực sự dựa vào hiện tượng vật lý không thể dự đoán. Dùng để tạo **entropy** (hay **seed**) dùng trong PRNG

- ◇ Avalanche diodes (Zener breakdown noise), reverse biased
- ◇ Atmospheric noise (via attached radio-receiver)
- ◇ Thermal noise in resistor (amplified)
- ◇ Radioactive decay etc.

- **PRNG (Pseudo Random Number Generator):** Thuật toán tạo ra một số ngẫu nhiên từ seed.

→ Một RNG tốt, được dùng trong mật mã cần TRNG sinh seed có entropy cao và PRNG tốt (không thể tính input từ output, không có mối liên hệ giữa input và output).

5.5.2 Sử dụng RNG trong RSA

Trong RSA, RNG được dùng để tạo số nguyên tố p và q ngẫu nhiên.

Nếu như, RNG sinh số không đủ ngẫu nhiên, thừa số nguyên tố p có thể bị trùng trong hai modulo $n1$ và $n2$ khác nhau, từ đó tính được $p = \gcd(n1, n2)$

Lý do RNG không sinh số ngẫu nhiên tốt đa số là do thiết bị không sử dụng TRNG hoặc sử dụng TRNG lấy nguồn entropy kém. Bên cạnh đó, cũng có thể do việc không sử dụng PRNG theo chuẩn NIST mà thay vào đó lại sử dụng PRNG tự chế.

→ Yêu cầu: cần sử dụng RNG chất lượng cao và sử dụng PRNG theo chuẩn của NIST như:

- Hash_DRBG (SHA-256, SHA-512)
- HMAC_DRBG
- AES-CTR-DRBG
- ChaCha20-DRBG (libsodium, Linux /dev/urandom)

Nếu được thì sử dụng HSM / TPM khi có thể.

5.6 Implementation bugs & side channels

Đây là lớp rủi ro liên quan đến việc lập trình thuật toán RSA trong các thư viện (OpenSSL, Bouncy Castle, v.v.) hoặc custom code.

- **Timing leaks in modular exponentiation:**

- ◇ **Phân tích:** Phép tính mũ $m^d \pmod n$ thường dùng thuật toán "Square-and-Multiply". Nếu bit của d là 1, CPU thực hiện thêm phép nhân; nếu là 0 thì không. Sự chênh lệch thời gian này (dù chỉ vài micro giây) qua mạng LAN hoặc internet đủ để kẻ tấn công thống kê và tìm ra d .

- ◇ **Khắc phục:**

- * Bắt buộc sử dụng cài đặt **Constant-time** (thời gian thực thi không phụ thuộc vào dữ liệu đầu vào).
- * Sử dụng kỹ thuật **Blinding**: Nhân dữ liệu đầu vào với một số ngẫu nhiên trước khi tính toán, sau đó loại bỏ nó, làm cho kẻ tấn công không thể đoán được trạng thái bên trong.

- **CRT recombination faults (The Bellcore Attack):**

- ◇ **Công thức:** RSA-CRT tính chữ ký S bằng cách tính $S_p = M^d \pmod p$ và $S_q = M^d \pmod q$. Nếu lỗi xảy ra khi tính S_p (tạo ra S'_p) nhưng S_q đúng, chữ ký sai S' sẽ được tạo ra.

- ◇ **Khai thác:** Kẻ tấn công tính ước chung lớn nhất:

$$\gcd(S - S', N) = q$$

- ◇ **Kết quả:** Lập tức tìm ra thừa số nguyên tố q , từ đó suy ra p và phá vỡ hoàn toàn RSA.
- ◇ **Khắc phục:** Luôn luôn kiểm tra lại kết quả tính toán: $S^e \equiv M \pmod N$ trước khi trả về S .

- **Padding oracle in TLS stacks (Bleichenbacher/ROBOT):**

- ◇ **Phân tích:** Trong RSA PKCS#1 v1.5 (dùng cho trao đổi khóa TLS cũ), cấu trúc bản rõ phải bắt đầu bằng 00 02. Nếu server phản hồi nhanh hơn hoặc trả về thông báo lỗi khác nhau khi giải mã một gói tin không đúng định dạng 00 02, nó tạo ra một "Padding Oracle".
- ◇ **Tác động:** Kẻ tấn công gửi hàng triệu gói tin biến thể, dựa vào phản hồi của server để giải mã Pre-Master Secret, từ đó giải mã toàn bộ phiên TLS.

CHƯƠNG 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

- ◇ **Khắc phục:** Loại bỏ hoàn toàn RSA Key Exchange trong cấu hình TLS server (chuyển sang dùng (EC)DHE cho Key Exchange và chỉ dùng RSA để ký). Nâng cấp lên TLS 1.3 (đã loại bỏ PKCS#1 v1.5 padding cho mã hóa).

5.7 Các kịch bản triển khai thực hiện

Chúng em có thực hiện một vài kịch bản triển khai tấn công RSA trên các hệ thống mô phỏng thực tế.

Tên	Mô tả	Link script
Wiener Attack on JWT	Mô phỏng hệ thống OAuth2 / OpenID Connect sử dụng d nhỏ để ký JWT, từ đó attacker thực hiện Wiener Attack để khôi phục d và giả mạo JWT.	https://github.com/QuocGia12/NT219-Project/tree/main/poc_implement_weekness/wiener%20attack%20on%20JWT
JWT Algorithm Confusion	Mô phỏng cách một Hacker có thể vượt qua cơ chế xác thực RSA (bất đối xứng) bằng cách ép Server sử dụng thuật toán HMAC (đối xứng) với Public Key đóng vai trò là mật khẩu (Secret Key).	https://github.com/QuocGia12/NT219-Project/tree/main/poc_implement_weekness/JWT_Confusion
RSA Signature Malleability & Replay Attack	Minh họa lỗ hổng Signature Malleability trong việc triển khai thuật toán RSA và cách nó dẫn đến lỗ hổng nghiêm trọng Replay Attack.	https://github.com/QuocGia12/NT219-Project/tree/main/poc_implement_weekness/signature_malleability%20and%20replay_attack

Bảng 5.1. Các kịch bản triển khai tấn công thực tế

5.8 Bảng tổng hợp giải pháp triển khai

Triển khai	Weakness	Giải pháp
1. TLS (Web Server)	<p>a. Không forward secrecy (TLS 1.2): Nếu Attacker lấy được d, sẽ giải mã được toàn bộ các phiên quá khứ do Server dùng chung tham số (n, e, d) cho mã hóa.</p> <p>b. Kịch bản Bleichenbacher Attack (TLS 1.2): Attacker ép Server chọn RSA Cipher Suite, dùng Server làm Oracle để lấy PreMasterSecret.</p>	<p>- Sử dụng TLS 1.3: Đã loại bỏ hoàn toàn trao đổi khóa RSA, chỉ sử dụng ECDHE (đảm bảo tính forward secrecy).</p> <p>- Trong TLS 1.3, Server ký toàn bộ transcript, Attacker không thể thay đổi quá trình bắt tay.</p> <p>- Khuyến nghị chỉ cho phép TLS 1.3 và vô hiệu hóa TLS 1.2 để giảm thiểu rủi ro bảo mật.</p>
2. JWT / token signing (RS256)	<p>- "alg": "None"</p> <p>- Verify "alg": "HS256" trong khi thuật toán ký là RS256</p>	<p>- Cần từ chối "alg": "None"</p> <p>- Phải xác định rõ alg khi verify</p>

Triển khai	Weakness	Giải pháp
3. Code signing & package ecosystems	<p>Weak signing tools / key exposure: Private key (file .pem/biến môi trường) lưu trên code repo hoặc build server. Attacker copy được sẽ tự ký "Golden Token" (leo quyền Admin).</p> <p>Legacy signature format: là các lược đồ hoặc định dạng chữ ký lỗi thời cho phép một message có thể có nhiều chữ ký hợp lệ khác nhau, dù được ký bằng cùng một private key. Dẫn đến:</p> <ul style="list-style-type: none"> - Signature Malleability: Có nhiều chữ ký được verify hợp lệ cho cùng một message → Message không thay đổi, chữ ký thay đổi nhưng vẫn hợp lệ. - Replay Attack: Signature Malleability + Sử dụng Signature làm ID định danh action. 	<ul style="list-style-type: none"> - Không lưu private key trên đĩa cứng server ứng dụng. - Sử dụng HSM hoặc dịch vụ quản lý khóa đám mây (AWS KMS, Azure Key Vault). - Private key không bao giờ rời khỏi thiết bị bảo mật. - Không sử dụng Signature làm ID định danh hành động (sử dụng Nonce thay thế), tách biệt vai trò của Signature và ID. - Hệ thống cần chỉ chấp nhận chữ ký ở dạng canonical, và từ chối các chữ ký có biểu diễn khác nhưng vẫn verify được.

CHƯƠNG 5. PHÂN TÍCH ĐIỂM YẾU TRONG CÁC TRIỂN KHAI THỰC TẾ

Triển khai	Weakness	Giải pháp
4. Smartcards / HSMs / TPMs	<p>a. Side-channel (Power Analysis): Đo dao động điện năng để tái tạo private key.</p> <p>b. Fault Injection (RSA-CRT): Dùng laser/điện áp gây lỗi tính toán \rightarrow lộ thừa số nguyên tố (p hoặc q).</p> <p>c. API misuse: Dùng HSM làm Oracle giải mã hoặc dùng sai quyền hạn key.</p>	<ul style="list-style-type: none"> - Dùng smartcard chuẩn FIPS 140-2 Level 3+ (có mesh shield). - Verify on-chip: Kiểm tra chữ ký trên chip trước khi xuất, nếu sai thì hủy phiên. - Key Wrapping: Chỉ wrap key, không decrypt dữ liệu thô. - Giới hạn Usage Flags: Chỉ Sign hoặc Decrypt, không bật cả hai.
5. Random Number Generator (RNG)	<p>RNG kém: Không dùng TRNG hoặc dùng PRNG tự chế \rightarrow sinh số không đủ ngẫu nhiên \rightarrow trùng thừa số $p \rightarrow$ tính được private key bằng GCD.</p>	<ul style="list-style-type: none"> - Sử dụng TRNG để sinh entropy (seed). - Sử dụng PRNG theo chuẩn NIST (Hash_DRBG, HMAC_DRBG, AES-CTR_DRBG, ChaCha20-DRBG). - Sử dụng HSM/TPM khi có thể.
6. Implementation bugs & side channels	<p>a. Timing leaks: Chênh lệch thời gian tính toán (Square-and-Multiply) làm lộ d.</p> <p>b. CRT recombination faults (Bellcore Attack): Lỗi khi tính S_p dẫn đến lộ q.</p> <p>c. Padding oracle (Bleichenbacher/ROBOT): Server phản hồi lỗi khác nhau với định dạng 00 02 sai \rightarrow Attacker giải mã được Pre-Master Secret.</p>	<ul style="list-style-type: none"> - Cài đặt Constant-time (thời gian không phụ thuộc dữ liệu). - Dùng kỹ thuật Blinding. - Luôn kiểm tra lại kết quả $S^e \equiv M \pmod{N}$ trước khi trả về S. - Loại bỏ RSA Key Exchange (chuyển sang (EC)DHE). - Nâng cấp lên TLS 1.3.

Bảng 5.2. Tổng hợp giải pháp khi triển khai

5.9 Checklist lưu ý khi triển khai RSA

- **Key size:** sử dụng n 3072 bits để thời gian an toàn lâu nhất có thể. Bên cạnh đó, sử dụng $e = 65537$, khi đó d được tính theo e sẽ tự động lớn, tránh các tấn công về key nhỏ.

- **Padding scheme:** Luôn phải sử dụng padding trong RSA:

- ◊ Key exchange \rightarrow RSA-OAEP
- ◊ Chữ ký số \rightarrow RSA-PSS

Tuyệt đối không sử dụng padding scheme PKCS#1 v1.5 trong RSA key exchange.

- **Side channel Protections:**

- ◊ **Constant time:** thời gian xử lý và trả về giống nhau với mọi input.
- ◊ **Không được phân biệt lỗi trả về**, mọi lỗi xảy ra đều trả về thông tin như nhau \rightarrow không tạo thành oracle.
- ◊ **Luôn sử dụng blinding:** từ đó, giả các thông tin attacker có thể thu được qua side-channel.
- ◊ **Verify Signature trước khi trả về** khi sử dụng **RSA-CRT**.

- **RNG:** dùng OS CSPRNG, không tự seed và đảm bảo entropy đủ.

- Đối với các hệ thống có giá trị cao (CA, TLS private key, firmware signing), **nên sử dụng HSM** để bảo vệ private key và chống side-channel/fault attacks.

Chương 6. KẾT LUẬN

Trong đồ án này, chúng em đã tiến hành khảo sát và phân tích toàn diện thuật toán RSA và các cơ chế chữ ký số dựa trên RSA, không chỉ ở khía cạnh lý thuyết toán học mà quan trọng hơn là ở các điểm yếu phát sinh trong quá trình triển khai và sử dụng thực tế. Thông qua việc nghiên cứu nhiều lớp tấn công khác nhau như factoring attack, padding oracle attack, key generation weakness, cũng như các tấn công ở mức triển khai (side-channel, fault attack), đồ án cho thấy rằng tính an toàn của RSA không chỉ phụ thuộc vào độ khó của bài toán phân tích thừa số, mà phụ thuộc rất lớn vào cách hệ thống thiết kế, cài đặt và vận hành RSA trong môi trường thực tế.

Các tấn công như Bleichenbacher, Marvin, hay RSA-CRT fault attack minh họa rõ ràng rằng RSA thường **không bị phá vỡ trực tiếp về mặt toán học**, mà bị khai thác thông qua rò rỉ thông tin phụ như lỗi padding, sai khác thời gian xử lý, hoặc lỗi phần cứng. Điều này nhấn mạnh rằng việc lựa chọn kích thước khóa lớn là **chưa đủ** nếu không đi kèm với padding an toàn, xử lý constant-time, nguồn sinh số ngẫu nhiên chất lượng cao và các biện pháp bảo vệ ở tầng triển khai.

Bên cạnh đó, phân tích các kịch bản triển khai hiện đại như TLS, JWT/RS256, code signing và HSM cho thấy nhiều cơ chế cũ (đặc biệt là PKCS#1 v1.5 và RSA key exchange trong TLS 1.2) tiềm ẩn rủi ro nghiêm trọng và cần được **loại bỏ hoặc thay thế một cách có chủ đích**. Việc chuyển sang các chuẩn hiện đại như RSA-OAEP, RSA-PSS, (EC)DHE và TLS 1.3 là yêu cầu tất yếu để đảm bảo an toàn dài hạn.

Tóm lại, RSA vẫn có thể được sử dụng an toàn **chỉ khi** đi kèm với các chuẩn thiết kế hiện đại và triển khai cẩn trọng. Chúng em có thể khẳng định một nguyên lý cốt lõi trong mật mã học ứng dụng: **phần lớn các sự cố an ninh không xuất phát từ việc thuật toán bị phá, mà từ cách thuật toán được triển khai và sử dụng trong thực tế**. Việc hiểu rõ các điểm yếu này là nền tảng quan trọng để xây dựng, đánh giá và vận hành các hệ thống mật mã an toàn.

TÀI LIỆU THAM KHẢO

- [1] *RSA cryptosystem*. https://en.wikipedia.org/wiki/RSA_cryptosystem. Accessed 2025.
- [2] Dan Boneh. “Twenty Years of Attacks on the RSA Cryptosystem”. In: (1999).
- [3] Daniel Bleichenbacher. “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1”. In: (1998).
- [4] *Everlasting ROBOT: the Marvin Attack*. <https://people.redhat.com/~hkario/marvin/marvin-attack-paper.pdf>.
- [5] Q. Kim. “Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures”. In: (2007).
- [6] Paul Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: (1996).
- [7] James Manger. “A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0”. In: (2001).
- [8] *Digital signature*. https://en.wikipedia.org/wiki/Digital_signature. Accessed 2025.