

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN**



**BÁO CÁO BÀI THỰC HÀNH
HỌC PHẦN: MẬT MÃ HỌC CƠ SỞ
MÃ HỌC PHẦN: INT1344**

**BÀI TẬP CÁ NHÂN:
HỆ MẬT KHÓA CÔNG KHAI RSA**

Tên sinh viên: Nguyễn Văn Hùng

Mã sinh viên: B22DCAT136

Nhóm lớp: 02

Giảng viên: TS. Quản Trọng Thế
HỌC KỲ 2 NĂM HỌC 2024-2025

MỤC LỤC

MỤC LỤC	1
DANH MỤC CÁC HÌNH VẼ	4
DANH MỤC CÁC BẢNG BIỂU	4
DANH MỤC CÁC TỪ VIẾT TẮT	5
MỞ ĐẦU	7
CHƯƠNG 1. Tổng quan hệ mật khóa công khai RSA	8
1.1 Giới thiệu chung	8
1.2 Vai trò và ứng dụng của RSA trong mật mã và bảo mật thông tin	9
1.3 Kết chương	10
CHƯƠNG 2. Cơ chế và nguyên lý hoạt động thuật toán RSA	11
2.1 Sinh khóa	11
2.2 Chia sẻ khóa	12
2.3 Mã hóa và giải mã	12
2.4 Kết chương	13
CHƯƠNG 3. Hiệu năng và độ an toàn rsa	14
3.1 Tốc độ mã hóa và giải mã của RSA	14
3.1.1 Các yếu tố ảnh hưởng đến hiệu năng	14
3.1.2 Tối ưu hóa hiệu năng	14
3.2 Độ dài khóa và ảnh hưởng đến tính bảo mật	15
3.2.1 Mối quan hệ giữa độ dài khóa và độ bảo mật	16
3.2.2 Khuyến nghị về độ dài khóa	16
3.3 So sánh RSA với các hệ mã khác	16
3.3.1 RSA và ElGamal	16
3.3.2 RSA và ECC (<i>Elliptic Curve Cryptography</i>)	17
3.3.3 RSA và AES (<i>Advanced Encryption Standard</i>)	17
3.3.4 RSA và các thuật toán mật mã hậu lượng tử	18
3.3.5 Bảng so sánh tổng hợp	19
3.4 Hiệu năng RSA trong các hệ thống thực tế	19
3.5 Kết chương	20

CHƯƠNG 4. lỗ hổng, điểm yếu và kỹ thuật tấn công vào RSA	21
4.1 Giới thiệu về các lỗ hổng của RSA	21
4.2 Tấn công phân tích khóa	21
4.2.1 Tấn công vét cạn (Brute-force attack)	21
4.2.2 Tấn công phân tích thừa số (Factoring attack)	22
4.2.3 Tấn công bằng phân tích các trường hợp đặc biệt	22
4.3 Tấn công kênh phụ (Side-channel attacks)	23
4.3.1 Tấn công thời gian (Timing attack)	23
4.3.2 Tấn công phân tích công suất (Power analysis attack)	23
4.3.3 Tấn công phát xạ điện từ (Electromagnetic analysis attack)	24
4.3.4 Tấn công lỗi (Fault attack)	24
4.4 Tấn công dựa trên lỗ hổng padding	24
4.4.1 Tấn công Bleichenbacher vào PKCS#1 v.5	24
4.4.2 Tấn công Manger vào PKCS#1 v2.0 (OAEP)	25
4.4.3 Tấn công vào Coppersmith	25
4.5 Những sai lầm khi triển khai thực tế RSA	26
4.5.1 Sinh khóa không an toàn	26
4.5.2 Triển khai RSA dễ bị tấn công kênh phụ	26
4.5.3 Sử dụng padding không đúng cách	26
4.5.4 Quản lý khóa kém	27
4.6 Giải pháp phòng chống	27
4.6.1 Sinh khóa an toàn	27
4.6.2 Phòng chống tấn công kênh phụ	28
4.6.3 Sử dụng padding hiện đại	28
4.6.4 Quản lý khóa hiệu quả	28
4.6.5 Triển khai an toàn	29
4.7 Kết chương	29
CHƯƠNG 5. Ứng dụng của RSA trong bảo mật thông tin	30
5.1 Mã hóa dữ liệu	30
5.1.1 Tổng quan về việc mã hóa dữ liệu bằng RSA	30
5.1.2 Mã hóa file	30

5.1.3 Mã hóa hình ảnh	31
5.2 Chữ ký số và bảo đảm tính toàn vẹn dữ liệu	31
5.2.1 Khái niệm và đặc điểm của chữ ký số RSA	31
5.2.2 Cơ sở toán học, quy trình và xác minh chữ ký số RSA	32
5.2.3 Vai trò của hàm băm trong chữ ký số RSA	32
5.2.4 Ứng dụng của chữ ký số RSA	33
5.2.5 So sánh RSA với các hệ thống chữ ký số khác	34
5.3 Quản lý chứng thực	34
5.3.1 Cơ sở hạ tầng khóa công khai (PKI)	34
5.3.2 SSL/TLS và HTTPS	35
5.3.3 Ứng dụng của RSA trong chứng chỉ số	36
5.4 Truyền thông an toàn	36
5.4.1 VPN (Virtual Private Network)	36
5.4.2 Email bảo mật	36
5.4.3 Truyền tin nhắn bảo mật	37
5.5 Tích hợp vào phần mềm và hệ điều hành	37
5.5.1 Xác thực phần mềm và cập nhật an toàn	37
5.5.2 Tích hợp RSA vào hệ điều hành	37
5.5.3 Bảo vệ dữ liệu trong ứng dụng	38
5.5.4 Secure Boot và Trusted Platform Module (TPM)	38
5.6 Kết chương	38
CHƯƠNG 6. DEMO	39
6.1 Tấn công module nhỏ (Small modulus attack)	39
6.2 Tấn công số mũ nhỏ (Small exponent attack - Håstad's broadcast attack)	39
6.3 Tấn công thông tin phụ (Common modulus attack)	40
6.4 Tấn công số p, q gần nhau (Fermat's factorization attack)	41
6.5 Tấn công Wiener trên số mũ riêng tư d quá nhỏ	42
6.6 Ứng dụng xác thực SSH bằng khóa công khai	44
KẾT LUẬN	48
TÀI LIỆU THAM KHẢO	49

DANH MỤC CÁC HÌNH VẼ

Hình 1 . RSA	11
Hình 2 . Tấn công module nhỏ	39
Hình 3 . Tấn công số mũ nhỏ	40
Hình 4 . Tấn công thông tin phụ	41
Hình 5 . Tấn công p, q gần nhau	42
Hình 6 . Tấn công Wiener	44
Hình 7 . Máy ảo Kali Linux	45
Hình 8 . Máy ảo Ubuntu	45
Hình 9 . Đảm bảo cài đặt SSH trên 2 máy ảo	45
Hình 10 . Tạo Secure Shell Keys	46
Hình 11 . Sao chép khóa công khai sang ubuntu	47
Hình 12 . Kiểm tra kết quả SSH	47

DANH MỤC CÁC BẢNG BIỂU

<i>Bảng 1.</i> Thời gian mã hóa và giải mã RSA	14
<i>Bảng 2.</i> Độ dài khóa bảo mật tương đương của RSA	16
<i>Bảng 3.</i> So sánh RSA và ElGamal	17
<i>Bảng 4.</i> So sánh RSA và ECC	17
<i>Bảng 5.</i> So sánh RSA và AES	18
<i>Bảng 6.</i> So sánh RSA và các thuật toán mật mã hậu lượng tử	19
<i>Bảng 7.</i> So sánh tổng hợp	19
<i>Bảng 8.</i> So sánh RSA và DSA	34
<i>Bảng 9.</i> So sánh RSA và ECDSA	34

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Thuật ngữ tiếng Anh/Giải thích	Thuật ngữ tiếng Việt/Giải thích
RSA	Rivest-Shamir-Adleman	Hệ mã hóa RSA
MIT	Massachusetts Institute of Technology	Viện Công nghệ Massachusetts
SSH	Secure Shell	Giao thức Shell An toàn
S/MIME	Secure/Multipurpose Internet Mail Extensions	Tiêu chuẩn Email An toàn
PGP	Pretty Good Privacy	Bảo mật Tốt
ECC	Elliptic Curve Cryptography	Mật mã Đường cong Elliptic
NIST	National Institute of Standards and Technology	Viện Tiêu chuẩn và Công nghệ Quốc gia (Mỹ)
ANSSI	Agence Nationale de la Sécurité des Systèmes d'Information	Cơ quan An ninh Hệ thống Thông tin Quốc gia (Pháp)
BSI	Bundesamt für Sicherheit in der Informationstechnik	Văn phòng Liên bang về An ninh Công nghệ Thông tin (Đức)
ECRYPT-CSA	European Consortium for Research in Cryptography – Cybersecurity Advisory	Liên minh Nghiên cứu Mật mã Châu Âu
HTTPS	Hypertext Transfer Protocol Secure	Giao thức Truyền Siêu văn bản An toàn
PKI	Public Key Infrastructure	Hạ tầng Khóa Công khai
CRT	Chinese Remainder Theorem	Định lý Số dư Trung Hoa
TPM	Trusted Platform Module	Mô-đun Nền tảng Đáng tin cậy
HSM	Hardware Security Module	Mô-đun Bảo mật Phần cứng
AES	Advanced Encryption Standard	Tiêu chuẩn Mã hóa Nâng cao
TLS/SSL	Transport Security/Secure Layer Sockets	Bảo mật Lớp Truyền tải

	Layer	
NTRU	N-th Degree Truncated Polynomial Ring	Hệ mã hóa dựa trên Lattice
SPHINCS+	Stateless Practical Hash-Based Signatures	Chữ ký Hash Không trạng thái
NFS	Number Field Sieve	Sàng Trường Số
ECM	Elliptic Curve Method	Phương pháp Đường cong Elliptic
PKCS	Public-Key Cryptography Standards	Tiêu chuẩn Mật mã Khóa Công khai
LLL	Lenstra–Lenstra–Lovász	Thuật toán LLL
GCD	Greatest Common Divisor	Ước chung Lớn nhất
OAEP	Optimal Asymmetric Encryption Padding	Đệm Mã hóa Bất đối xứng Tối ưu
CSPRNG	Cryptographically Secure Pseudorandom Number Generator	Bộ sinh số Giả ngẫu nhiên An toàn
PSS	Probabilistic Signature Scheme	Lược đồ Chữ ký Xác suất
RSA-KEM	RSA Key Encapsulation Mechanism	Cơ chế Đóng gói Khóa RSA
IEEE TIFS	IEEE Transactions on Information Forensics and Security	Tạp chí An ninh Thông tin và Pháp y Số
DKIM	DomainKeys Identified Mail	Xác thực Email bằng Khóa Miền
ECDSA	Elliptic Curve Digital Signature Algorithm	Thuật toán Chữ ký số Đường cong Elliptic
CA	Certificate Authority	Tổ chức Chứng thực
CRL	Certificate Revocation List	Danh sách Thu hồi Chứng chỉ
PGP/OpenPGP	Pretty Good Privacy/Open PGP	Chuẩn Mã hóa PGP Mở

MỞ ĐẦU

Trong bối cảnh Internet và các hệ thống thông tin ngày càng phát triển, yêu cầu về bảo mật dữ liệu trở nên cấp thiết hơn bao giờ hết. Một trong những thách thức lớn là đảm bảo rằng chỉ có người được ủy quyền mới có thể truy cập hoặc sửa đổi thông tin nhạy cảm. Chính vì vậy, các phương pháp mã hóa khóa công khai (public-key cryptography) ra đời đã mở ra kỷ nguyên mới cho bảo mật thông tin, cho phép trao đổi khóa một cách an toàn ngay cả khi hai bên chưa từng gặp nhau.

RSA (Rivest–Shamir–Adleman), được phát triển vào năm 1977, là thuật toán mã hóa khóa công khai đầu tiên được áp dụng rộng rãi và vẫn giữ vai trò nền tảng trong nhiều giao thức an toàn ngày nay. Báo cáo này nhằm mục đích:

Khám phá nguyên lý toán học: Trình bày cơ sở số học của RSA, bao gồm bài toán phân tích thừa số, thuật toán lũy thừa modulo, ...

Phân tích tính năng và độ an toàn: Đánh giá các phương thức sinh khóa, cơ chế padding, cũng như các tấn công và giải pháp phòng chống (padding oracle, timing attack, Coppersmith attack, v.v.).

Ứng dụng thực tiễn: Thực hiện các demo mã hóa, ký số và minh họa các tình huống khai thác lỗ hổng để hiểu rõ hơn ưu nhược điểm của RSA trong môi trường thực tế.

Qua báo cáo, hi vọng mọi người không chỉ nắm vững lý thuyết mà còn hình dung được cách RSA vận dụng trong các hệ thống bảo mật hiện đại, cũng như nhận diện và khắc phục các điểm yếu tiềm ẩn trong bảo mật thông tin.

CHƯƠNG 1. TỔNG QUAN HỆ MẬT KHÓA CÔNG KHAI RSA

1.1 Giới thiệu chung

Mã hóa đối xứng dù đã phát triển từ cổ điển đến hiện đại nhưng vẫn tồn tại một số điểm yếu như:

- Vấn đề trao đổi khóa giữa người gửi và người nhận: Quá trình trao đổi khóa yêu cầu phải có kênh an toàn để trao đổi, nhưng điều này gây nhiều khó khăn về mặt phí và chậm trễ về mặt thời gian
- Tính bí mật của khóa: Việc khóa bí mật cần trao đổi giữa người gửi và người nhận sẽ dẫn đến trường hợp không có cơ sở quy trách nhiệm nếu khóa bị tiết lộ.
- Vấn đề quản lý khóa: Rất phức tạp khi sử dụng trong môi trường trao đổi tin giữa nhiều người dùng. Với số lượng người dùng là n thì số khóa cần tạo lập là $n(n-1)/2$. Gây khó khăn và không an toàn khi n tăng lớn
- Trên cơ sở mã đối xứng, không thể thiết lập được khái niệm chữ ký điện tử, dẫn đến không có dịch vụ non-repudiation (không thể phủ nhận được) cho các giao dịch thương mại trên mạng.

Để khắc phục điểm yếu, cần thực hiện 1 số phương án sau:

Phương án 1: Người nhận (Bob) giữ bí mật khóa K_2 , còn khóa K_1 thì công khai cho tất cả mọi người biết. Alice muốn gửi dữ liệu cho Bob thì dùng khóa K_1 để mã hóa. Bob dùng K_2 để giải mã. Chỉ có duy nhất Bob mới có thể giải mã được. Ưu điểm của phương án này là không cần phải truyền khóa K_1 trên kênh an toàn.

Phương án 2: Người gửi (Alice) giữ bí mật khóa K_1 còn khóa K_2 thì công khai cho tất cả mọi người biết. Alice muốn gửi dữ liệu cho Bob thì dùng khóa K_1 để mã hóa. Bob dùng K_1 để giải mã. Người dùng khác biết khóa K_2 nên cũng có thể giải mã được. Phương án này không đảm bảo tính bí mật. Vì vậy nếu kết hợp phương án 1 và 2 sẽ khắc phục được các nhược điểm của mã hóa đối xứng.

Vào năm 1976 Whitfield Diffie và Martin Hellman đã tìm ra một phương pháp mã hóa khác mà có thể giải quyết được hai vấn đề trên, đó là mã hóa khóa công khai hay còn gọi là mã hóa bất đối xứng. Đây có thể xem là một bước đột phá quan trọng nhất trong lĩnh vực mã hóa. Tuy nhiên, họ vẫn chưa đưa ra được một hệ mã hóa khóa công khai hoàn chỉnh.

Năm 1977, ba nhà nghiên cứu tại MIT gồm Ron Rivest, Adi Shamir và Leonard Adleman công bố thuật toán RSA trong bài báo “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems” (Communications of the ACM), tên

RSA được lấy theo chữ cái đầu của ba tác giả., và ngày nay đang được sử dụng rộng rãi. Đây là lần đầu tiên có một thuật toán mật mã khóa công khai vừa cho phép mã hóa vừa cho phép chữ ký số, với nền tảng toán học rõ ràng và có khả năng triển khai thực tế.

Về mặt tổng quát thuật toán RSA là một phương pháp mã hóa theo khối. Trong đó thông điệp M và bản mã C là các số nguyên từ 0 đến 2^i với i số bit của khối. Kích thước thường dùng của i là 1024 bit. Thuật toán RSA sử dụng hàm một chiều là vấn đề phân tích một số thành thừa số nguyên tố và bài toán Logarith rời rạc.

Sau khi ra mắt, RSA nhanh chóng được nghiên cứu và tối ưu:

- Tối ưu tính toán bằng các phương pháp như Square-and-Multiply, Chinese Remainder Theorem (CRT) giúp tăng tốc mã hóa/giải mã
- Các thuật toán tấn công ra đời để thử thách độ an toàn: Pollard's Rho, Quadratic Sieve, General Number Field Sieve (GNFS). Điều này thúc đẩy việc sử dụng khóa dài hơn.
- Chuẩn hóa RSA:
 - o PKCS#1 (Public-Key Cryptography Standards) định nghĩa cách sử dụng RSA trong mã hóa và chữ ký số.
 - o RFC 8017 (IETF) và NIST tiếp tục duy trì các tiêu chuẩn hiện đại.

1.2 Vai trò và ứng dụng của RSA trong mật mã và bảo mật thông tin

Cung cấp tính bảo mật và xác thực song song

- Mã hóa công khai: Cho phép gửi dữ liệu bí mật tới chủ sở hữu khóa bí mật mà không cần chia sẻ trước khóa, giải quyết vấn đề trao đổi khóa an toàn.
- Chữ ký số: Với cùng cơ chế toán học, RSA cho phép tạo và kiểm tra chữ ký số, đảm bảo tính xác thực (authentication), tính toàn vẹn (integrity) và không thể chối bỏ (non-repudiation).

Nền tảng cho Hệ thống Chứng thực Khóa Công khai (PKI)

- RSA là thuật toán chính trong Digital Certificates (X.509), nơi Certificate Authority (CA) phát hành chứng chỉ chứa khóa công khai của entity kèm chữ ký số của CA.
- PKI dựa vào RSA để xây dựng niềm tin: từ website (HTTPS) đến hạ tầng email (S/MIME), SSH, VPN, v.v.

Ứng dụng rộng rãi trong các giao thức bảo mật

- SSL/TLS: Tại bước handshake, RSA (hoặc RSA-based key exchange) được dùng để mã hóa khóa phiên (session key).

- SSH: Xác thực và mã hóa kênh SSH ban đầu thường sử dụng RSA key pairs.
- Email security (S/MIME, PGP): Mã hóa và ký số email bằng RSA.
- Code signing: Chữ ký phần mềm/ứng dụng phân phối (Java JAR signing, Windows Authenticode) sử dụng RSA để đảm bảo file chưa bị thay đổi.

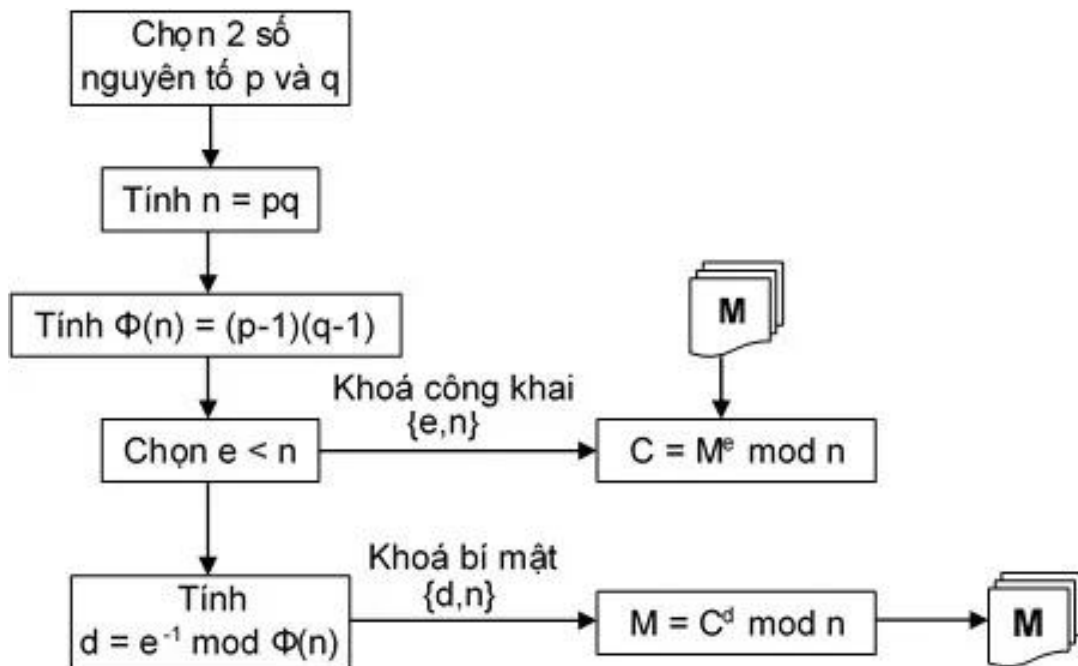
Hiện nay, RSA vẫn được sử dụng phổ biến vì tính đơn giản, độ tin cậy đã được khẳng định qua hơn 40 năm và hệ sinh thái công cụ hỗ trợ đa dạng. Nhưng với sự phát triển của công nghệ, kỹ thuật ngày một phát triển, NIST khuyến cáo sử dụng RSA-2048 bit cho đến năm 2030, đồng thời các hệ thống mới nên xem xét elliptic-curve cryptography (ECC) hoặc mật mã kháng lượng tử (Post-Quantum Cryptography) trong dài hạn.

1.3 Kết chương

RSA là một trong những phát minh quan trọng nhất trong lịch sử mật mã, đặt nền móng cho mật mã khóa công khai và chữ ký số. Dù đã ra đời từ lâu, nó vẫn đóng vai trò thiết yếu trong bảo mật thông tin. Tuy nhiên, với sự phát triển của máy tính lượng tử, RSA có thể dần được thay thế bằng các thuật toán kháng lượng tử trong tương lai.

CHƯƠNG 2. CƠ CHẾ VÀ NGUYÊN LÝ HOẠT ĐỘNG THUẬT TOÁN RSA

Hoạt động của RSA dựa trên 4 bước chính: sinh khóa, chia sẻ key, mã hóa và giải mã



Hình 1. RSA

2.1 Sinh khóa

- Chọn hai số nguyên tố lớn p và q và tính $N = pq$. Cần chọn p và q sao cho: $M < 2^{i-1} < N < 2^i$ với i là chiều dài bản rõ.
- Tính $\phi(n) = (p - 1)(q - 1)$
- Tìm một số e sao cho: $\gcd(e, \phi(n)) = 1$ và $1 < e < \phi(n)$
- Tìm một số d sao cho: $e \cdot d \equiv 1 \bmod \phi(n)$ (hay: $d \equiv e^{-1} \bmod \phi(n)$) (d là nghịch đảo của e trong phép module $\phi(n)$)
- Chọn khóa công khai K_U là cặp (e, N) , khóa riêng K_R là cặp (d, N)

Ví dụ sinh khóa:

- Chọn 2 số nguyên tố: $p = 61, q = 53$
- Tính $N = pq = 3233$
- Tính $\phi(n) = (p - 1)(q - 1) = 3120$
- Chọn e sao cho $\gcd(e, \phi(n)) = 1$ và $1 < e < \phi(n)$: $e = 17$
- Tìm $d \equiv e^{-1} \bmod \phi(n) \Rightarrow d \times e \equiv 1 \bmod 3120$
 - Giải bằng thuật toán Eculid mở rộng, ta được: $d = 2753$

- Kết quả sinh khóa:
 - o Khóa công khai: $(n = 3233, e = 17)$
 - o Khóa riêng: $(n = 3233, d = 2753)$

2.2 Chia sẻ khóa

Khóa công khai (Public Key): Được công bố rộng rãi (ví dụ: qua chứng chỉ số, website, email).

Khóa riêng (Private Key): Giữ bí mật tuyệt đối, không chia sẻ.

Lưu ý:

- Không cần trao đổi khóa bí mật như mật mã đối xứng.
- Đảm bảo tính xác thực của khóa công khai thông qua PKI và chứng chỉ số (X.509).

2.3 Mã hóa và giải mã

Phương án 1: Mã hóa thông điệp

Mục đích: Truyền tin bí mật từ người gửi đến người nhận

Quy trình:

- Mã hóa bằng khóa công khai (K_U): $C = E(M, K_U) = M^e \bmod N$
 - o Đầu vào: Bản rõ M (kích thước $i - 1$ bit), khóa công khai (e, N)
 - o Đầu ra: Bản mã C (kích thước i bit)
- Giải mã bằng khóa riêng (K_R): $M = D(C, K_R) = C^d \bmod N$
 - o Điều kiện: $M < N$

Lưu ý:

- Bắt buộc thêm padding (ví dụ: OAEP) trước khi mã hóa để tránh tấn công.
- M phải có kích thước $\leq i - 1$ bit để đảm bảo $M < 2^{i-1} < N < 2^i$

Phương án 2: Mã hóa chứng thực (Chữ ký số)

Mục đích: Xác thực nguồn gốc và tính toàn vẹn của thông điệp

Quy trình:

- Ký số bằng khóa riêng (K_R): $C = E(M, K_R) = M^d \bmod N$
- Đầu vào: Bản rõ M, khóa riêng (d, N)
- Đầu ra: Chữ ký C

Lưu ý:

- Thực tế, chữ ký số thường áp dụng trên hash của M (ví dụ: SHA-256) thay vì M trực tiếp.
- Tiêu chuẩn PKCS#1 v2.2 (RFC 8017) quy định cách ký/kiểm tra chữ ký RSA.

Ví dụ mã hóa và giải mã:

- Giả sử mã hóa thông điệp $M = 65$
- Mã hóa bằng công thức: $C = M^e \bmod N = 65^{17} \bmod 3233 = 2790$
- Giải mã: $M = C^d \bmod N = 2790^{2753} \bmod 3233 = 65$

2.4 Kết chương

Chương này cho chúng ta hiểu về nguyên lý hoạt động của thuật toán RSA, hiểu được cách RSA sinh khóa, chia sẻ khóa, mã hóa và giải mã. Qua đó, có cái nhìn rõ hơn về RSA.

CHƯƠNG 3. HIỆU NĂNG VÀ ĐỘ AN TOÀN RSA

3.1 Tốc độ mã hóa và giải mã của RSA

3.1.1 Các yếu tố ảnh hưởng đến hiệu năng

Hiệu năng của RSA phụ thuộc vào nhiều yếu tố, bao gồm độ dài khóa, cấu hình phần cứng, và các kỹ thuật tối ưu hóa được áp dụng. Độ phức tạp tính toán trong RSA chủ yếu đến từ các phép toán lũy thừa modulo với các số nguyên lớn.

Theo nghiên cứu của Boneh và Shoup (2020), các phép toán cơ bản trong RSA có độ phức tạp như sau:

- Tạo khóa: $O(k^4)$, với k là số bit của khóa
- Mã hóa: $O(k^2)$, với k là số bit của khóa
- Giải mã: $O(k^3)$, với k là số bit của khóa

Theo thực nghiệm từ NIST (National Institute of Standards and Technology), thời gian mã hóa và giải mã RSA trên một máy tính thông thường có thể được ước tính như sau:

Độ dài khóa	Thời gian mã hóa (ms)	Thời gian giải mã (ms)
1024 bit	0.08	1.5
2048 bit	0.16	7.8
3072 bit	0.31	17.1
4096 bit	0.6	37.4

Bảng 1. Thời gian mã hóa và giải mã RSA

3.1.2 Tối ưu hóa hiệu năng

Thuật toán RSA sử dụng các phép toán lũy thừa module lớn, thường chậm hơn so với các thuật toán mã hóa khác. Để cải thiện hiệu năng của RSA, một số kỹ thuật tối ưu hóa đã được đề xuất, đặc biệt trong giai đoạn giải mã và ký số, vốn yêu cầu tính toán phức tạp hơn so với mã hóa.

3.1.2.1 Định lý thặng dư Trung Hoa (CRT)

Kỹ thuật này có thể tăng tốc quá trình giải mã lên đến 4 lần. Định lý CRT cho phép thực hiện các phép tính modulo nhỏ hơn và sau đó kết hợp kết quả, thay vì thực hiện một phép tính modulo lớn.

- Thay vì tính toán trực tiếp: $M = C^d \bmod n$ ta chia nhỏ phép tính này thành hai phép toán nhỏ hơn: $M_p = C^{d_p} \bmod p$ và $M_q = C^{d_q} \bmod q$

- Sau đó, áp dụng Định lý thặng dư Trung Hoa (CRT) để kết hợp m_1 và m_2 thành $m \bmod n$
- Hiệu quả: Vì p và q chỉ bằng khoảng một nửa độ dài của n , việc tính toán nhanh hơn đáng kể

Theo Rivest (2018), việc áp dụng CRT trong giải mã RSA có thể giảm thời gian tính toán xuống còn khoảng 1/3 so với phương pháp truyền thống.

3.1.2.2 Lũy thừa nhanh (Fast modular exponentiation)

- Phép tính, thuật toán lũy thừa nhanh sử dụng phương pháp "bình phương và nhân" để giảm số phép nhân xuống $O(\log b)$.
- Với mã hóa: $C = M^e \bmod N$ và giải mã: $M = C^d \bmod N$, vì số mũ e và d có thể rất lớn (thường vài trăm bit), cần tối ưu bằng cách:
 - Biểu diễn e hoặc d dưới dạng nhị phân
 - Lặp qua từng bit: mỗi bước bình phương và nếu bit = 1 thì nhân thêm

3.1.2.3 Phần cứng chuyên dụng (Hardware Acceleration)

Việc sử dụng các bộ xử lý mật mã chuyên dụng hoặc các card mở rộng mật mã có thể cải thiện đáng kể hiệu năng của RSA. RSA có thể được xử lý bởi:

- TPM (Trusted Platform Module)
- HSM (Hardware Security Module)
- Cryptographic accelerator (ví dụ: Intel QuickAssist, nShield...)

Ưu điểm:

- Tăng tốc độ xử lý RSA hàng chục lần
- Bảo vệ khóa bí mật vật lý (không thể bị trích xuất)
- Rất hiệu quả trong các hệ thống quy mô lớn như: máy chủ web, hệ thống tài chính, blockchain...

3.1.2.4 Cơ số Montgomery

- Kỹ thuật này chuyển đổi các phép toán modulo thành dạng thuận tiện hơn để tính toán, giảm đáng kể chi phí tính toán cho các phép nhân modulo lặp lại.

3.1.2.5 Tiền tính toán

- Tính toán và lưu trữ sẵn một số giá trị trung gian để sử dụng trong quá trình mã hóa và giải mã

Việc áp dụng CRT trong giải mã RSA có thể giảm thời gian tính toán xuống còn khoảng 1/3 so với phương pháp truyền thống

3.2 Độ dài khóa và ảnh hưởng đến tính bảo mật

3.2.1 *Mối quan hệ giữa độ dài khóa và độ bảo mật*

Độ dài khóa là yếu tố quan trọng nhất ảnh hưởng đến tính bảo mật của RSA. Theo nguyên lý, độ khó của việc phân tích một số nguyên lớn thành các thừa số nguyên tố tăng theo hàm mũ khi độ dài khóa tăng.

Năm 2001, Lenstra và Verheul đã phát triển một mô hình để đánh giá độ an toàn của các độ dài khóa khác nhau dựa trên tiến bộ của phần cứng và các thuật toán phân tích số nguyên. Theo mô hình này, một khóa RSA có độ dài k bit cung cấp độ bảo mật tương đương với khoảng $k/3$ bit đối xứng.

Độ dài khóa RSA	Độ bảo mật tương đương (bit)
1024 bit	Khoảng 80 bit
2048 bit	Khoảng 112 bit
3072 bit	Khoảng 128 bit
4096 bit	Khoảng 152 bit
8192 bit	Khoảng 192 bit

Bảng 2. Độ dài khóa bảo mật tương đương của RSA

3.2.2 *Khuyến nghị về độ dài khóa*

Các tổ chức uy tín trong lĩnh vực bảo mật đã đưa ra các khuyến nghị về độ dài khóa RSA tối thiểu:

- Từ năm 2023, NIST khuyến nghị sử dụng khóa RSA tối thiểu 2048 bit cho các ứng dụng cần bảo mật đến năm 2030, và 3072 bit cho các ứng dụng cần bảo mật lâu dài.
- ANSSI (Pháp) khuyến nghị sử dụng khóa RSA ít nhất 2048 bit cho đến năm 2030 và 3072 bit sau đó.
- BSI (Đức) khuyến nghị sử dụng khóa RSA tối thiểu 2000 bit và nên chuyển sang 3000 bit từ năm 2023.
- ECRYPT-CSA (EU) đề xuất sử dụng khóa RSA 3072 bit cho bảo mật dài hạn.

3.3 So sánh RSA với các hệ mã khác

3.3.1 *RSA và ElGamal*

ElGamal là một hệ mã khóa công khai dựa trên bài toán logarithm rời rạc.

Tiêu chí	RSA	ElGamal
Cơ sở toán học	Bài toán phân tích số nguyên	Bài toán logarithm rời rạc

Tốc độ mã hóa	Chậm hơn	Nhanh hơn
Tốc độ giải mã	Nhanh hơn khi sử dụng CRT	Chậm hơn
Độ dài khóa	Lớn hơn để đạt cùng độ bảo mật	Nhỏ hơn RSA ở cùng độ bảo mật
Độ mở rộng văn bản	Không đáng kể	Gấp đôi kích thước thông điệp gốc
Ứng dụng chính	Chữ ký số, trao đổi khóa	Mã hóa, trao đổi khóa

Bảng 3. So sánh RSA và ElGamal

Theo Smart (2016), với cùng mức độ bảo mật, kích thước khóa ElGamal có thể nhỏ hơn RSA khoảng 1.5 lần.

3.3.2 RSA và ECC (Elliptic Curve Cryptography)

ECC là hệ mã dựa trên tính chất của đường cong elliptic trên trường hữu hạn:

Tiêu chí	RSA	ECC
Cơ sở toán học	Bài toán phân tích số nguyên	Bài toán logarithm rời rạc trên đường cong elliptic
Độ dài khóa	Lớn	Nhỏ (1/6 đến 1/10 so với RSA ở cùng độ bảo mật)
Tốc độ tạo khóa	Chậm	Nhanh hơn nhiều
Tốc độ mã hóa	Chậm	Nhanh hơn
Tốc độ giải mã	Chậm	Nhanh hơn
Yêu cầu tài nguyên	Cao	Thấp, phù hợp với thiết bị cấu hình thấp
Tối ưu hóa phần cứng	Khó khăn hơn	Dễ dàng hơn, hiệu quả trên IoT và thiết bị di động

Bảng 4. So sánh RSA và ECC

Theo nghiên cứu của Bernstein và Lange năm 2017, khóa ECC 256 bit cung cấp độ bảo mật tương đương với khóa RSA 3072 bit, đồng thời giảm đáng kể chi phí tính toán. Khiến ECC trở thành lựa chọn tốt hơn cho các thiết bị có tài nguyên hạn chế như thiết bị IoT, điện thoại di động và thẻ thông minh.

3.3.3 RSA và AES (Advanced Encryption Standard)

AES là một thuật toán mã hóa đối xứng, trong khi RSA là thuật toán mã hóa bất đối xứng:

Tiêu chí	RSA	AES
Loại thuật toán	Bất đối xứng (khóa công khai)	Đối xứng (khóa bí mật)
Tốc độ	Chậm (100-1000 lần chậm hơn AES)	Nhanh
Độ dài khóa	2048-4096 bit (phổ biến)	128, 192, 256 bit
Ứng dụng chính	Trao đổi khóa, chữ ký số	Mã hóa dữ liệu lớn
Độ an toàn	Dựa trên bài toán phân tích số	Dựa trên sự bền vững của cấu trúc khối
Hiệu quả mã hóa dữ liệu lớn	Kém	Tốt

Bảng 5. So sánh RSA và AES

Theo ECRYPT-CSA (2018), để mã hóa một khối dữ liệu 1MB, RSA 2048 bit có thể mất khoảng 7-8 giây, trong khi AES-256 chỉ mất khoảng 0.03 giây trên cùng một phần cứng.

Trong thực tế, RSA và AES thường được sử dụng kết hợp trong các hệ thống mã hóa: RSA được dùng để trao đổi khóa AES, sau đó AES được sử dụng để mã hóa dữ liệu thực tế. Đây là tiêu chuẩn trong các giao thức như TLS/SSL, PGP.

3.3.4 RSA và các thuật toán mật mã hậu lượng tử

Với sự phát triển của máy tính lượng tử, các thuật toán mật mã hậu lượng tử đang được phát triển để thay thế RSA và các hệ mã khóa công khai truyền thống:

Tiêu chí	RSA	Kyber (Lattice based)	NTRU	SPHINCS+ (Hash-based)
Cơ sở toán học	Phân tích số nguyên	Bài toán mạng lưới	Bài toán mạng lưới	Hàm băm
Độ dài khóa thông dụng	Lớn	Nhỏ hơn RSA	Nhỏ hơn RSA	Rất lớn

Độ dài chữ ký	Nhỏ	Nhỏ	Nhỏ	Lớn
Tốc độ	Chậm	Nhanh	Nhanh	Chậm (chữ ký)
Khả năng chống tấn công lượng tử	Kém	Tốt	Tốt	Rất tốt
Mức độ tin cậy	Cao (đã được kiểm chứng)	Trung bình (mới)	Trung bình (mới)	Cao (dựa trên cơ sở toán học đơn giản)

Bảng 6. So sánh RSA và các thuật toán mật mã hậu lượng tử

NIST hiện đang trong quá trình tiêu chuẩn hóa các thuật toán mật mã hậu lượng tử để thay thế RSA và ECC trong tương lai

3.3.5 Bảng so sánh tổng hợp

Tiêu chí	RSA	ElGamal	ECC	AES
Loại thuật toán	Bất đối xứng	Bất đối xứng	Bất đối xứng	Đối xứng
Độ dài khóa thông dụng	Lớn	2048-4096 bit	256-384 bit	128-256 bit
Tốc độ xử lý	Chậm	Chậm	Trung bình	Nhanh
Khả năng chống tấn công lượng tử	Yếu	Yếu	Tốt hơn RSA	Trung bình
Yêu cầu tài nguyên	Cao	Cao	Thấp	Thấp

Bảng 7. So sánh tổng hợp

3.4 Hiệu năng RSA trong các hệ thống thực tế

Các nghiên cứu đã chỉ ra hiệu năng của RSA trong các hệ thống thực tế:

- Máy chủ web có lưu lượng cao: Một máy chủ web xử lý 100 kết nối TLS mới mỗi giây với RSA 2048-bit có thể tiêu tốn khoảng 10-15% CPU chỉ cho các phép toán RSA.
- Thiết bị di động: Trên thiết bị di động, các phép toán RSA có thể tiêu tốn đáng kể pin và tài nguyên CPU. Theo Sullivan (2019), giải mã RSA 2048-bit trên smartphone trung bình tiêu tốn khoảng 30 mJ năng lượng, so với 0.03 mJ cho AES-128.

- Thiết bị IoT: Các thiết bị có tài nguyên hạn chế thường gặp khó khăn với RSA. Theo Kumar et al. (2018), việc triển khai RSA trên các thiết bị IoT điển hình có thể tiêu tốn đến 80% tài nguyên tính toán có sẵn.

3.5 Kết chương

RSA vẫn là một trong những hệ mã khóa công khai được sử dụng rộng rãi nhất, với độ an toàn đáng tin cậy khi sử dụng độ dài khóa phù hợp và triển khai đúng cách. Tuy nhiên, hiệu năng của RSA kém hơn đáng kể so với các hệ mã đối xứng như AES, và kém hơn một số hệ mã khóa công khai mới hơn như ECC.

CHƯƠNG 4. LỖ HỔNG, ĐIỂM YẾU VÀ KỸ THUẬT TẤN CÔNG VÀO RSA

4.1 Giới thiệu về các lỗ hổng của RSA

Mặc dù RSA đã được chứng minh là một hệ mã khóa công khai có độ an toàn cao khi được triển khai đúng cách, nhưng vẫn tồn tại nhiều lỗ hổng và điểm yếu có thể bị khai thác trong các tình huống thực tế. Các lỗ hổng này không phải đến từ khuyết điểm trong thuật toán RSA cơ bản, mà thường liên quan đến quá trình triển khai hoặc sử dụng không đúng cách.

Theo nghiên cứu của Dan Boneh (1999) trong bài báo “Twenty Years of Attacks on the RSA Cryptosystem”, các tấn công vào RSA có thể được phân loại thành ba nhóm chính:

- Tấn công toán học nhắm vào bản thân thuật toán
- Tấn công vào quá trình triển khai RSA
- Tấn công kênh phụ (Side-channel) dựa trên quan sát các đặc điểm vật lý của hệ thống

Ví dụ thực tế:

- ROCA (2017): Lỗ hổng trong chip Infineon do sinh số nguyên tố bằng phương pháp xác suất yếu, cho phép tấn công khôi phục khóa RSA-2048 trong vài giờ.
- Bleichenbacher Attack (1998): Tấn công PKCS#1 v1.5, ảnh hưởng đến TLS và HTTPS.
- Debian OpenSSL Bug (2008): Lỗi sinh số ngẫu nhiên yếu khiến hàng nghìn khóa RSA có thể bị phá vỡ.

4.2 Tấn công phân tích khóa

4.2.1 Tấn công vét cạn (Brute-force attack)

Tấn công vét cạn về cơ bản là thử phân tích n bằng cách kiểm tra các ước số nguyên tố. Đối với RSA, điều này đồng nghĩa với việc thử tất cả các cặp số nguyên tố có thể để tìm ra p và q sao cho $n = p \times q$.

Theo Lenstra et al. (2012), thì mức độ an toàn tương đương so với mã đối xứng của RSA:

- Với khóa RSA 1024 bit là khoảng 80-bit security (không còn an toàn)
- Với khóa RSA 2048 bit là khoảng 112-bit security
- Với khóa RSA 3072 bit là khoảng 128-bit security

Với công nghệ hiện tại, một tấn công vét cạn đối với khóa RSA 2048 bit trở lên được coi là không khả thi trong thực tế. Theo Koblitz và Menezes (2016), ngay cả khi sử dụng tất cả sức mạnh tính toán trên toàn cầu, việc phá vỡ một khóa RSA 2048 bit bằng tấn công vét cạn sẽ mất hàng tỷ năm.

4.2.2 Tấn công phân tích thừa số (Factoring attack)

Phân tích thừa số là phương pháp chính để tấn công RSA, dựa trên việc phân tích modulo n thành thừa số nguyên tố p và q . Các thuật toán phân tích thừa số hiện đại bao gồm:

4.2.2.1 Sàng trường số (Number Field Sieve - NFS)

- Đây là thuật toán hiệu quả nhất hiện nay để phân tích số nguyên lớn.
- Độ phức tạp của NFS là: $L(n) = e^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}}$ với $c \approx 1.923$ và n là số cần phân tích.
- Nó ứng dụng hiệu quả cho số nguyên lớn hơn 100 chữ số (VD: RSA-1024 trở lên)

4.2.2.2 Sàng thể vuông (Quadratic Sieve)

- Độ phức tạp: $L(n) = e^{(1+o(1))(\ln n)^{1/2}(\ln \ln n)^{1/2}}$
- Nó ứng dụng hiệu quả cho các số có ít hơn 110 chữ số thập phân

4.2.2.3 Phân tích thừa số ECM (Elliptic Curve Method)

- Hiệu quả cho việc tìm các thừa số nguyên tố nhỏ.
- Hiệu quả khi một trong các thừa số p hoặc q có kích thước ≤ 80 chữ số (dù n rất lớn).

Theo Kleinjung et al. (2010), kỷ lục phân tích số RSA 768 bit (232 chữ số thập phân) vào năm 2009 đã sử dụng thuật toán NFS và tiêu tốn khoảng 2000 năm CPU. Với tiến bộ của phần cứng và thuật toán, Aoki et al. (2012) ước tính rằng phân tích một số RSA 1024 bit sẽ tốn khoảng 1 triệu năm máy tính.

4.2.3 Tấn công bằng phân tích các trường hợp đặc biệt

Một số tấn công nhắm vào các trường hợp đặc biệt khi sinh khóa RSA không đúng cách:

4.2.3.1 Tấn công p và q quá gần nhau

- Nếu $|p - q|$ quá nhỏ, thuật toán Fermat có thể hiệu quả phân tích n .
- Để đảm bảo an toàn, nên chọn p và q sao cho $|p - q| > 2^{n/2-100}$, với n là số bit của modulus.

4.2.3.2 Tấn công Wiener

- Khi sử dụng số mũ giải mã d quá nhỏ ($d < n^{0.25}$), tấn công Wiener có thể khôi phục d từ khóa công khai (n, e) trong thời gian đa thức.
- Năm 2000, Boneh và Durfee đã mở rộng giới hạn này lên $d < n^{0.292}$

4.2.3.3 Tấn công thừa số chung

- Theo Lenstra et al. (2012), có thể tìm thấy các thừa số chung giữa các modulo RSA khác nhau bằng thuật toán Euclid. Nghiên cứu của họ trên 6.4 triệu chứng chỉ SSL cho thấy 0.2% các chứng chỉ chia sẻ thừa số nguyên tố, cho phép tính toán khóa riêng.

4.3 Tấn công kênh phụ (Side-channel attacks)

4.3.1 Tấn công thời gian (Timing attack)

Tấn công thời gian khai thác sự khác biệt về thời gian thực hiện các phép toán mật mã để suy ra thông tin về khóa. Tấn công này được giới thiệu lần đầu vào 1996 bởi Kocher và đã được chứng minh là hiệu quả đối với các triển khai RSA không được bảo vệ.

Nguyên lý cơ bản: Trong quá trình tính lũy thừa modulo bằng thuật toán bình phương và nhân (square-and-multiply), thời gian thực hiện phụ thuộc vào các bit của số mũ.

- Đối với mỗi bit là 1, thuật toán thực hiện cả bình phương và nhân \rightarrow Thời gian lâu hơn
- Đối với bit là 0, chỉ thực hiện bình phương \rightarrow Thời gian ngắn hơn

Năm 2003, David Brumley và Dan Boneh đã chứng minh khả năng thực hiện tấn công thời gian từ xa thông qua mạng đối với máy chủ web OpenSSL qua bài “Remote Timing Attacks are Practical”: họ thực hiện hàng triệu truy vấn tới một server OpenSSL qua mạng nội bộ, đo thời gian phản hồi cho mỗi decrypt và thành công trích xuất khóa RSA 1024-bit trong vài giờ. Kết quả này buộc OpenSSL phải mặc định bật blinding để chống lại tấn công

Tấn công thời gian còn hiệu quả hơn khi kết hợp với việc sử dụng Định lý thặng dư Trung Hoa (CRT) trong quá trình giải mã RSA: Quá trình giảm dư của thuật toán Montgomery có thể xảy ra “extra reduction” khi giá trị tạm vượt ngưỡng, và điều này phụ thuộc vào các bit của p hoặc q . Onur Aciicmez cùng cộng sự đã trình bày cách khai thác chính xác độ trễ của các bước Montgomery này trong “Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations” (ACM CCS 2005), cho thấy việc kết hợp CRT làm tăng độ chính xác của tấn công thời gian

4.3.2 Tấn công phân tích công suất (Power analysis attack)

Tấn công phân tích công suất khai thác sự khác biệt về mức tiêu thụ điện năng trong quá trình thực hiện các phép toán mật mã. Có hai loại chính:

- Phân tích công suất đơn giản (Simple Power Analysis - SPA): Quan sát trực tiếp mức tiêu thụ điện năng trong một lần thực hiện để xác định các phép toán và trích xuất thông tin khóa. Theo Kocher et al. (1999), SPA có thể trực tiếp phân biệt giữa các phép toán bình phương và nhân trong thuật toán lũy thừa modulo.
- Phân tích công suất vi sai (Differential Power Analysis - DPA): Sử dụng phân tích thống kê trên nhiều lần đo để loại bỏ nhiễu và trích xuất thông tin khóa. Messerges et al. (1999) đã sử dụng phân tích thống kê trên hàng nghìn lần đo để tách tín hiệu khỏi nhiễu, chứng minh rằng DPA có thể phá vỡ RSA ngay cả khi có các biện pháp chống SPA.

4.3.3 Tấn công phát xạ điện từ (Electromagnetic analysis attack)

Tương tự như tấn công phân tích công suất, nhưng thay vì đo tiêu thụ điện năng, kẻ tấn công đo phát xạ điện từ từ thiết bị. Năm 2001 Quisquater và Samyde1 đã giới thiệu khái niệm "TEMPEST" (phát hiện phát xạ điện từ) và chứng minh khả năng khôi phục khóa từ phát xạ điện từ.

Gandolfi et al. (2001) đã so sánh phân tích công suất và phân tích điện từ, cho thấy trong một số trường hợp, phân tích điện từ có thể hiệu quả hơn do có thể định vị chính xác các thành phần xử lý mật mã trên chip (Ví dụ: ALU, Bộ nhớ).

4.3.4 Tấn công lỗi (Fault attack)

Tấn công lỗi liên quan đến việc cố ý gây ra lỗi trong quá trình tính toán mật mã và phân tích kết quả sai để suy ra thông tin khóa.

Phương pháp gây lỗi phổ biến:

- Xung điện (Voltage Glitching): Thay đổi điện áp đột ngột.
- Laser Injection: Chiếu tia laser vào transistor để gây nhiễu.
- Thay đổi nhiệt độ: Làm chậm phản ứng của chip.

Boneh et al. (1997) đã trình bày tấn công lỗi đối với RSA sử dụng CRT. Nếu kẻ tấn công có thể gây ra lỗi trong một trong hai phép tính modulo ($\text{mod } p$ hoặc $\text{mod } q$), thì từ chữ ký đúng s và chữ ký lỗi s' , có thể tính được $p = \gcd(n, s - s')$.

4.4 Tấn công dựa trên lỗ hổng padding

4.4.1 Tấn công Bleichenbacher và PKCS#1 v.5

Tấn công Bleichenbacher, được Daniel Bleichenbacher giới thiệu vào năm 1998, nhắm vào lỗ hổng trong cơ chế đệm PKCS#1 v1.5 được sử dụng trong mã hóa RSA.

Đây là một loại tấn công "oracle" (với oracle là một hệ thống hoặc thiết bị cho phép kẻ tấn công biết liệu một ciphertext có định dạng PKCS#1 v1.5 hợp lệ hay không).

Nguyên lý của tấn công này:

- Kẻ tấn công bắt được một ciphertext C được mã hóa với khóa công khai của nạn nhân.
- Kẻ tấn công gửi nhiều ciphertext được chọn $C' = C \cdot r^e \bmod n$ cho máy chủ.
- Dựa vào phản hồi của máy chủ (thành công hay thất bại khi giải mã), kẻ tấn công thu hẹp dần phạm vi có thể của plaintext.

Trong nghiên cứu ban đầu, Bleichenbacher (1998) cho thấy cần khoảng một triệu truy vấn để giải mã một ciphertext 512 bit. Các cải tiến sau này của Bardou et al. (2012) đã giảm số lượng truy vấn xuống còn khoảng vài nghìn truy vấn bằng tối ưu toán học.

Mặc dù đã được công bố từ lâu, tấn công Bleichenbacher vẫn tiếp tục được phát hiện trong nhiều triển khai hiện đại. Một biến thể gần đây được gọi là "ROBOT" (Return Of Bleichenbacher's Oracle Threat) được Böck et al. (2018) phát hiện, ảnh hưởng đến 27% trong mẫu 100 trang web hàng đầu sử dụng HTTPS.

4.4.2 Tấn công Manger vào PKCS#1 v2.0 (OAEP)

Mặc dù OAEP (Optimal Asymmetric Encryption Padding) được thiết kế để khắc phục các lỗ hổng trong PKCS#1 v1.5, Manger (2001) đã phát hiện một lỗ hổng trong việc triển khai OAEP. Tấn công này tương tự như tấn công Bleichenbacher nhưng khai thác một oracle khác.

Nguyên lý: OAEP yêu cầu byte đầu tiên của plaintext giải mã phải là 0x00. Manger (2001) khai thác oracle dựa trên việc máy chủ trả lời "Invalid Ciphertext" nếu byte đầu tiên khác 0.

4.4.3 Tấn công vào Coppersmith

Tấn công Coppersmith, được Don Coppersmith phát triển vào năm 1996, là một kỹ thuật toán học sử dụng cơ sở Gröbner và thuật toán LLL (Lenstra-Lenstra-Lovász) để tìm nghiệm nhỏ của đa thức modulo.

- Ta có thể xây dựng một lattice từ các hệ số của một đa thức $f(x)$ và rồi dùng thuật toán giảm chuẩn LLL để tìm tất cả các nghiệm m sao cho:

$$f(m) \equiv 0 \pmod{N} \text{ và } |m| < N^{1/d}, \text{ với } d \text{ là bậc của đa thức}$$

Trong bối cảnh RSA, tấn công này có thể áp dụng khi:

- Một phần của plaintext đã biết (ví dụ: định dạng chuẩn, tiêu đề, v.v.)
- Các bit có giá trị thấp hoặc cao của khóa riêng bị lộ

- Mã hóa với số mũ e nhỏ mà không sử dụng padding thích hợp

Theo Boneh và Shoup (2020), nếu không sử dụng padding và số mũ $e = 3$, một thông điệp được gửi đến ba người nhận khác nhau có thể bị khôi phục hoàn toàn thông qua Định lý thặng dư Trung Hoa và tính căn bậc ba.

4.5 Những sai lầm khi triển khai thực tế RSA

4.5.1 Sinh khóa không an toàn

Một trong những sai lầm phổ biến nhất trong triển khai RSA là quá trình sinh khóa không đảm bảo tính ngẫu nhiên và an toàn.

- Số nguyên tố yếu: Theo Heninger et al. (2012), nhiều thiết bị nhúng sử dụng các bộ sinh số ngẫu nhiên chất lượng thấp, dẫn đến việc tạo ra các số nguyên tố có thể dự đoán được. Nghiên cứu của họ trên 5.8 triệu địa chỉ IPv4 cho thấy 0.75% chứng chỉ TLS và 0.03% chứng chỉ SSH có khóa dễ bị tấn công do sinh khóa kém, dẫn đến việc chia sẻ thừa số nguyên tố tạo ra các khóa dễ bị tấn công bằng GCD scan. Đến nay, các hệ thống đã cải thiện đáng kể nhờ FIPS 186-5.
- Số nguyên tố p và q quá gần nhau: Như đã đề cập, việc chọn p và q quá gần nhau sẽ làm modulo n dễ bị phân tích bằng thuật toán Fermat.
- Sử dụng số mũ công khai e quá nhỏ: Mặc dù giá trị $e = 3$ hoặc $e = 65537$ thường được sử dụng để tối ưu hóa hiệu năng, việc sử dụng e nhỏ mà không có padding thích hợp có thể dẫn đến tấn công RSA cơ bản.
- Sử dụng cùng một số nguyên tố cho nhiều khóa: Lenstra et al. (2012) phát hiện ra rằng nhiều chứng chỉ SSL chia sẻ các thừa số nguyên tố, có thể do sử dụng cùng một nguồn entropy kém.

4.5.2 Triển khai RSA dễ bị tấn công kênh phụ

Nhiều triển khai không bảo vệ chống lại các tấn công kênh phụ:

- Triển khai không bất biến về thời gian: Các thuật toán lũy thừa modulo không bất biến về thời gian dễ bị tấn công thời gian. Theo Aciicmez và Schindler (2008), ngay cả việc sử dụng cache CPU cũng có thể gây ra các kênh phụ có thể bị khai thác.
- Phản hồi lỗi quá chi tiết: Việc cung cấp thông tin chi tiết về lỗi giải mã (như trong tấn công Bleichenbacher) tạo ra một "oracle" có thể bị khai thác.
- Thiếu xác thực trước khi giải mã: Một số triển khai giải mã dữ liệu trước khi xác thực, có thể dẫn đến các tấn công timing oracle.

4.5.3 Sử dụng padding không đúng cách

Việc sử dụng padding không đúng cách hoặc không sử dụng padding là nguồn gốc của nhiều lỗ hổng:

- Thiếu padding: RSA không sử dụng padding dễ bị tấn công toán học như tấn công Coppersmith.
- Tiếp tục sử dụng PKCS#1 v1.5: Mặc dù đã biết các lỗ hổng, PKCS#1 v1.5 vẫn được sử dụng rộng rãi do tính tương thích ngược. Theo Jager et al. (2012), các biến thể của tấn công Bleichenbacher vẫn hiệu quả đối với nhiều triển khai TLS hiện đại.
- Triển khai OAEP không đầy đủ: Klingner et al. (2014) đã phát hiện ra rằng một số triển khai OAEP không kiểm tra đầy đủ định dạng sau khi giải mã, dẫn đến các lỗ hổng bảo mật.

4.5.4 Quản lý khóa kém

Các vấn đề về quản lý khóa cũng góp phần tạo ra các lỗ hổng:

- Sử dụng cùng một cặp khóa cho cả mã hóa và chữ ký, việc này có thể dẫn đến các tấn công liên quan đến chữ ký mù.
- Khóa riêng không được bảo vệ đầy đủ: Việc lưu trữ khóa riêng không an toàn làm tăng nguy cơ rò rỉ khóa.
- Sử dụng khóa quá lâu: Khuyến nghị thay đổi khóa RSA định kỳ, tùy thuộc vào độ dài khóa và mục đích sử dụng, để giảm thiểu nguy cơ bị tấn công theo thời gian.

4.6 Giải pháp phòng chống

4.6.1 Sinh khóa an toàn

Để đảm bảo sinh khóa an toàn, cần tuân thủ các nguyên tắc sau:

- Sử dụng nguồn entropy đủ mạnh: Theo khuyến nghị của NIST SP 800-90A Rev. 1 (2015), nên sử dụng các bộ sinh số ngẫu nhiên mật mã học (CSPRNG) đã được chứng thực.
- Chọn p và q cách xa nhau: Rivest và Silverman (1999) đề xuất :

$$|p - q| > 2^{\frac{n}{2}-100} \text{ với } n \text{ là số bit của modulo.}$$

- Kiểm tra tính nguyên tố kỹ lưỡng: FIPS 186-4 (2013) khuyến nghị sử dụng kiểm tra Miller-Rabin với ít nhất 40 lần lặp ngẫu nhiên, hoặc các phương pháp chứng minh nguyên tố như Lucas đối với những ứng dụng đòi hỏi tính chất chặt chẽ cao.

- Chọn độ dài khóa phù hợp: NIST SP 800-57 (2020) khuyến nghị sử dụng khóa RSA tối thiểu 2048 bit cho các ứng dụng hiện tại và 3072 bit cho bảo mật lâu dài (đến sau năm 2030).

4.6.2 Phòng chống tấn công kênh phụ

Để bảo vệ chống lại các tấn công kênh phụ, cần áp dụng các biện pháp sau:

- Thuật toán bất biến về thời gian:
 - Để chống tấn công timing, cần dùng các thuật toán lũy thừa modulo không phụ thuộc vào dữ liệu bí mật.
 - Theo Yarom và Falkner (2014), nên sử dụng các triển khai lũy thừa modulo bất biến về thời gian như "Montgomery ladder", tránh rò rỉ thông tin qua thời gian thực thi.
- Thêm độ trễ ngẫu nhiên: Paul Kocher (1996) từng khuyến nghị chèn ngẫu nhiên khoảng trễ vào quá trình tính toán để làm nhiễu tín hiệu thời gian, khiến kẻ tấn công khó xác định chính xác chuỗi bit "1" và "0" trong số mũ
- Sử dụng mù (blinding): Cũng theo Kocher (1996), nên sử dụng kỹ thuật message blinding (Cho trước $C' = C \cdot r^e \bmod N$ trước khi giải mã, sau đó nhân ngược r^{-1} để phá hủy mối liên kết giữa ciphertext và thời gian xử lý, triệt tiêu kênh thời gian).
- Chống tấn công phân tích công suất bằng nhiều biện pháp như mù dữ liệu, mù số mũ, và masking intermediate values để chống cả SPA (Simple Power Analysis) và DPA (Differential Power Analysis).
- Chống tấn công lỗi: Shamir (1999) đề xuất phương pháp "infective computing" để phát hiện và vô hiệu hóa kết quả của các tấn công lỗi.

4.6.3 Sử dụng padding hiện đại

Sử dụng các kỹ thuật padding hiện đại là rất quan trọng:

- Chuyển từ PKCS#1 v1.5 sang OAEP: RSA-OAEP được khuyến nghị cho mã hóa khóa và RSA-KEM (Key Encapsulation Mechanism) được khuyến nghị cho các ứng dụng yêu cầu bảo mật cao.
- Sử dụng PSS cho chữ ký: Nên sử dụng RSA-PSS (Probabilistic Signature Scheme) thay vì PKCS#1 v1.5 cho chữ ký số.
- Triển khai đầy đủ các kiểm tra: Cần thực hiện kiểm tra cẩn thận định dạng padding sau khi giải mã và xử lý lỗi một cách an toàn.

4.6.4 Quản lý khóa hiệu quả

Quản lý khóa hiệu quả bao gồm các khía cạnh sau:

- Phân tách khóa theo mục đích sử dụng: Nên sử dụng các cặp khóa riêng biệt cho mã hóa và chữ ký.
- Bảo vệ khóa riêng: NIST SP 800-57 (2020) cung cấp hướng dẫn toàn diện về việc bảo vệ khóa mật mã, bao gồm sử dụng mô-đun bảo mật phần cứng (HSM) cho các khóa quan trọng.
- Luân chuyển khóa theo định kỳ: Khuyến nghị thời gian sử dụng khóa RSA từ 1-3 năm tùy theo độ dài khóa và mức độ bảo mật cần thiết.
- Kế hoạch thu hồi khóa: Nên có kế hoạch thu hồi khóa trong trường hợp khóa bị xâm phạm.

4.6.5 Triển khai an toàn

Ngoài các biện pháp cụ thể nêu trên, một số nguyên tắc chung cho việc triển khai RSA an toàn bao gồm:

- Sử dụng các thư viện mật mã đã được chứng thực: Nên sử dụng các thư viện mật mã đã được kiểm chứng kỹ lưỡng như OpenSSL, BouncyCastle, hoặc libsodium thay vì tự triển khai các thuật toán mật mã.
- Cập nhật thường xuyên: Tiếp tục cập nhật các thư viện mật mã để khắc phục các lỗ hổng mới phát hiện.
- Kiểm tra bảo mật: Kiểm tra định kỳ hệ thống đối với các lỗ hổng RSA đã biết bằng các công cụ như TLS-Scanner hoặc SSLyze.
- Lập kế hoạch chuyển đổi hậu lượng tử: Theo NIST IR 8105 (2021), các tổ chức nên bắt đầu chuẩn bị cho việc chuyển đổi từ RSA sang các thuật toán mật mã hậu lượng tử.

4.7 Kết chương

RSA đã chứng minh là một hệ mã khóa công khai đáng tin cậy trong hơn bốn thập kỷ, nhưng tính bảo mật của nó phụ thuộc rất nhiều vào việc triển khai đúng cách. Mặc dù thuật toán cơ bản là an toàn khi sử dụng độ dài khóa đầy đủ, nhiều cuộc tấn công thành công đã khai thác các lỗi trong quá trình triển khai. Để tránh những rủi ro không mong muốn, hiểu rõ cách triển khai đúng cách là vô cùng quan trọng.

CHƯƠNG 5. ỨNG DỤNG CỦA RSA TRONG BẢO MẬT THÔNG TIN

5.1 Mã hóa dữ liệu

5.1.1 Tổng quan về việc mã hóa dữ liệu bằng RSA

Thuật toán RSA được sử dụng rộng rãi để mã hóa các loại dữ liệu khác nhau, từ văn bản, tệp tin, hình ảnh đến âm thanh. Quá trình mã hóa dữ liệu bằng RSA dựa trên việc sử dụng khóa công khai để mã hóa thông tin và chỉ người sở hữu khóa riêng tương ứng mới có thể giải mã được thông tin đó.

Tuy nhiên, do tính chất tính toán phức tạp của RSA, thuật toán này thường không được sử dụng để mã hóa trực tiếp lượng dữ liệu lớn. Thay vào đó, RSA thường được kết hợp với các thuật toán mã hóa đối xứng trong một hệ thống mã hóa lai (hybrid encryption).

Quá trình mã hóa lai diễn hình:

- Hệ thống tạo khóa đối xứng ngẫu nhiên (ví dụ: AES) để mã hóa dữ liệu gốc
- Dữ liệu được mã hóa bằng khóa đối xứng
- Khóa đối xứng sau đó được mã hóa bằng khóa công khai RSA của người nhận
- Cả dữ liệu đã mã hóa và khóa đối xứng đã mã hóa được gửi cho người nhận
- Người nhận sử dụng khóa riêng RSA của mình để giải mã khóa đối xứng, sau đó sử dụng khóa đối xứng để giải mã dữ liệu

Theo NIST (National Institute of Standards and Technology), phương pháp này kết hợp hiệu quả tốc độ mã hóa của thuật toán đối xứng với tính bảo mật của RSA khi trao đổi khóa.

5.1.2 Mã hóa file

Khi mã hóa tệp tin bằng RSA trong quy trình lai, quá trình được thực hiện như sau:

- File được chia thành các khối dữ liệu (nếu kích thước lớn)
- Một khóa đối xứng (thường là AES-256) được tạo ra
- Từng khối dữ liệu được mã hóa bằng khóa đối xứng
- Khóa đối xứng được mã hóa bằng khóa công khai RSA
- File cuối cùng bao gồm khóa đối xứng đã mã hóa và các khối dữ liệu đã mã hóa

Các phần mềm như PGP (Pretty Good Privacy) và GnuPG triển khai phương pháp này để bảo vệ tệp tin. Nghiên cứu từ Elsevier năm 2022 cho thấy phương pháp

này đạt hiệu suất tốt với tốc độ mã hóa đạt 25-50MB/giây trên phần cứng thông thường, trong khi vẫn duy trì mức độ bảo mật cao.

5.1.3 Mã hóa hình ảnh

Mã hóa hình ảnh bằng RSA đòi hỏi phương pháp đặc biệt do:

- Kích thước dữ liệu lớn
- Yêu cầu về hiệu suất
- Tính chất tương quan cao giữa các pixel

Trong thực tế, các kỹ thuật lai được áp dụng:

- Hình ảnh được tiền xử lý (nén, chuyển đổi miền,...)
- Khóa đối xứng được sử dụng để mã hóa dữ liệu hình ảnh
- RSA được sử dụng để mã hóa và trao đổi khóa đối xứng

Để tăng cường bảo mật, một số hệ thống kết hợp RSA với các phương pháp như hoán vị hỗn loạn (chaotic permutation). Theo nghiên cứu từ IEEE Transactions on Information Forensics and Security, phương pháp này làm tăng đáng kể sức mạnh chống lại các cuộc tấn công thống kê và phân tích tần suất.

5.1.3.1 Mã hóa âm thanh

Mã hóa âm thanh sử dụng RSA thường áp dụng các kỹ thuật:

- Phân tích tín hiệu âm thanh thành các thành phần tần số
- Mã hóa các thành phần quan trọng bằng hệ thống lai RSA-AES
- Áp dụng các kỹ thuật chèn nhiễu có kiểm soát

Trong các ứng dụng VoIP và truyền dẫn âm thanh thời gian thực, RSA thường chỉ được sử dụng trong quá trình thiết lập kết nối ban đầu để trao đổi khóa phiên. Sau đó, các thuật toán nhẹ hơn được sử dụng để bảo đảm hiệu suất.

5.2 Chữ ký số và bảo đảm tính toàn vẹn dữ liệu

5.2.1 Khái niệm và đặc điểm của chữ ký số RSA

Chữ ký số là một cơ chế mật mã học được sử dụng để xác minh tính xác thực và toàn vẹn của thông điệp hoặc tài liệu số. Chữ ký số RSA dựa trên cơ chế mã hóa khóa công khai-riêng tư, trong đó khóa riêng được sử dụng để tạo chữ ký và khóa công khai được sử dụng để xác minh chữ ký.

Đặc điểm của chữ ký số RSA:

- Xác thực nguồn gốc (Authentication): Xác nhận danh tính của người gửi
- Toàn vẹn dữ liệu (Integrity): Đảm bảo dữ liệu không bị thay đổi

- Không thể chối bỏ (Non-repudiation): Người gửi không thể phủ nhận đã ký tài liệu
- Không thể giả mạo (Unforgeable): Không thể tạo chữ ký hợp lệ mà không biết khóa riêng

Theo tiêu chuẩn FIPS 186-4 của NIST, chữ ký số RSA là một trong ba thuật toán chữ ký số được phê duyệt cho các ứng dụng chính phủ và thương mại tại Hoa Kỳ (cùng với DSA và ECDSA).

5.2.2 Cơ sở toán học, quy trình và xác minh chữ ký số RSA

- Chữ ký số RSA dựa trên các nguyên lý toán học tương tự như mã hóa RSA, nhưng với quy trình ngược lại:
 - o Trong mã hóa RSA: Dữ liệu được mã hóa bằng khóa công khai và giải mã bằng khóa riêng
 - o Trong chữ ký số RSA: "Chữ ký" được tạo bằng khóa riêng và xác minh bằng khóa công khai

Độ an toàn của chữ ký số RSA, giống như mã hóa RSA, dựa trên sự khó khăn của bài toán phân tích số nguyên lớn thành các thừa số nguyên tố.

Quy trình tạo chữ ký:

- Chuẩn bị thông điệp
- Băm thông điệp: Thực hiện hàm băm $h = H(m)$, ví dụ SHA-256
- Ký băm: Tính $s = RSASP1((n, d), h) = h^d \bmod n$
- Đính kèm chữ ký: Gửi cặp (m, s) hoặc $(H(m), s)$

Xác minh chữ ký:

- Nhận (m, s) . Lấy khóa công khai (n, e) từ nguồn tin cậy.
- Băm lại thông điệp: $h = H(m)$
- Giải mã chữ ký: $h' = RSASP1((n, e), s) = s^e \bmod n$
- So sánh: chấp nhận nếu và chỉ nếu $h' = h$

5.2.3 Vai trò của hàm băm trong chữ ký số RSA

Hàm băm đóng vai trò quan trọng trong quy trình chữ ký số RSA vì các lý do sau:

- Hiệu suất: RSA hoạt động chậm trên thông điệp lớn, hàm băm tạo ra giá trị nhỏ hơn
- Tính toàn vẹn: Thay đổi nhỏ trong thông điệp gây thay đổi lớn trong giá trị băm

- Độ an toàn: Ngăn chặn các tấn công toán học cụ thể
- Các hàm băm phổ biến được sử dụng với RSA:
 - SHA-256, SHA-384, SHA-512 (được khuyến nghị)
 - SHA-1 (không còn được khuyến nghị vì lỗ hổng bảo mật)
 - MD5 (không còn được sử dụng cho các ứng dụng bảo mật)

Theo NIST Special Publication 800-57, các thuật toán băm được khuyến nghị là SHA-2 và SHA-3, với SHA-1 không còn được chấp nhận cho các ứng dụng chữ ký số mới từ năm 2014.

5.2.4 Ứng dụng của chữ ký số RSA

Chữ ký số RSA được ứng dụng rộng rãi trong nhiều lĩnh vực:

5.2.4.1 Xác thực tệp tin và tài liệu

- Tệp tin thực thi (EXE, DLL): Hệ điều hành kiểm tra chữ ký trước khi thực thi
- Tài liệu PDF: Tiêu chuẩn PDF hỗ trợ chữ ký số RSA (PAdES - PDF Advanced Electronic Signatures)
- Tài liệu Microsoft Office: Các tệp Word, Excel, PowerPoint có thể được ký số
- Mã nguồn phần mềm: Kiểm tra tính toàn vẹn trong hệ thống kiểm soát phiên bản

5.2.4.2 Bảo mật email và xác thực người gửi

- S/MIME: Tiêu chuẩn email bảo mật, sử dụng RSA cho chữ ký số
- DKIM (DomainKeys Identified Mail): Xác thực email ở cấp độ tên miền
- Xác thực email công ty: Chứng minh email thực sự đến từ tổ chức được xác minh

5.2.4.3 Giao dịch tài chính

- Ngân hàng trực tuyến: Ký các giao dịch để đảm bảo tính xác thực
- Hợp đồng điện tử: Đảm bảo tính pháp lý và không thể chối bỏ
- Thanh toán di động: Xác thực các lệnh thanh toán
- Hóa đơn điện tử: Đảm bảo tính toàn vẹn của thông tin hóa đơn

5.2.4.4 Đảm bảo tính toàn vẹn dữ liệu trong truyền thông

- Giao thức truyền thông: TLS/SSL sử dụng RSA để ký các thông điệp trong quá trình bắt tay
- Dữ liệu IoT: Xác minh dữ liệu từ các thiết bị cảm biến

- Truyền thông doanh nghiệp: Đảm bảo thông tin nội bộ không bị giả mạo

5.2.5 So sánh RSA với các hệ thống chữ ký số khác

5.2.5.1 RSA và DSA (Digital Signature Algorithm)

Tiêu chí	RSA	DSA
Cơ sở toán học	Phân tích số nguyên lớn	Logarithm rời rạc
Tốc độ tạo chữ ký	Chậm hơn	Nhanh hơn
Tốc độ xác minh	Nhanh hơn	Chậm hơn
Kích thước chữ ký	Lớn hơn	Nhỏ hơn
Độ phổ biến	Rất phổ biến	Ít phổ biến hơn

Bảng 8. So sánh RSA và DSA

5.2.5.2 RSA và ECDSA (Elliptic Curve Digital Signature Algorithm)

Tiêu chí	RSA	ECDSA
Cơ sở toán học	Phân tích số nguyên lớn	Đường cong elliptic
Độ dài khóa cần thiết	2048-4096 bit	256-384 bit
Tốc độ tạo chữ ký	Chậm hơn	Rất nhanh
Tốc độ xác minh	Nhanh	Khá nhanh
Kích thước chữ ký	Lớn hơn	Nhỏ hơn
Hiệu suất trên thiết bị di động	Kém	Tốt

Bảng 9. So sánh RSA và ECDSA

ECDSA ngày càng được ưa chuộng do kích thước khóa và chữ ký nhỏ hơn đáng kể, cũng như hiệu suất tốt hơn trên thiết bị có tài nguyên hạn chế. Theo Cloudflare, khoảng 30% chứng chỉ TLS mới được phát hành vào năm 2023 sử dụng ECDSA, tăng từ dưới 10% vào năm 2018.

5.3 Quản lý chứng thực

5.3.1 Cơ sở hạ tầng khóa công khai (PKI)

Cơ sở hạ tầng khóa công khai (PKI - Public Key Infrastructure) là một hệ thống toàn diện gồm phần cứng, phần mềm, chính sách và quy trình cần thiết để tạo, quản

lý, phân phối, sử dụng, lưu trữ và thu hồi chứng chỉ số và quản lý mã hóa khóa công khai. RSA đóng vai trò nền tảng trong hầu hết các triển khai PKI hiện đại.

Thành phần chính của PKI:

- Cơ quan chứng nhận (CA - Certificate Authority): Phát hành và quản lý chứng chỉ số
- Cơ quan đăng ký (RA - Registration Authority): Xác minh danh tính trước khi CA phát hành chứng chỉ
- Kho lưu trữ chứng chỉ: Cơ sở dữ liệu về chứng chỉ đã phát hành
- Danh sách thu hồi chứng chỉ (CRL - Certificate Revocation List): Danh sách các chứng chỉ đã bị thu hồi

Theo RFC 5280, tiêu chuẩn X.509 cho PKI, RSA là thuật toán chữ ký được hỗ trợ rộng rãi nhất. Nghiên cứu từ Globalsign cho thấy tính đến năm 2023, RSA vẫn được sử dụng trong khoảng 85% chứng chỉ SSL/TLS toàn cầu, mặc dù có xu hướng chuyển dần sang các thuật toán dựa trên đường cong elliptic (ECC).

5.3.2 SSL/TLS và HTTPS

SSL (Secure Sockets Layer) và phiên bản kế nhiệm TLS (Transport Layer Security) là các giao thức mật mã bảo mật thông tin liên lạc qua mạng máy tính. HTTPS là ứng dụng của SSL/TLS cho truy cập web an toàn.

Vai trò của RSA trong SSL/TLS:

- Xác thực máy chủ: Máy chủ chứng minh danh tính thông qua chứng chỉ RSA được ký bởi CA tin cậy
- Thiết lập khóa phiên: Trong các phiên bản cũ của SSL/TLS, RSA được sử dụng để trao đổi khóa phiên (thường là AES)
- Chữ ký số: Xác minh tính toàn vẹn của dữ liệu trao đổi

Quy trình bắt tay SSL/TLS với RSA:

- Client gửi "ClientHello" với thông tin phiên bản TLS hỗ trợ
- Server phản hồi "ServerHello" với chứng chỉ chứa khóa công khai RSA
- Client xác minh chứng chỉ qua CA
- Client tạo Pre-Master Secret, mã hóa nó bằng khóa công khai RSA của server
- Server giải mã Pre-Master Secret bằng khóa riêng RSA
- Cả hai bên tạo Master Secret từ Pre-Master Secret
- Master Secret được sử dụng để tạo khóa phiên cho mã hóa dữ liệu

Tuy nhiên, hầu hết phương pháp trao đổi khóa RSA truyền thống đã bị loại bỏ để ủng hộ các thuật toán cung cấp tính chất Forward Secrecy (như Diffie-Hellman). Theo Cloudflare, đến năm 2023, hơn 70% lưu lượng web toàn cầu đã sử dụng TLS 1.3.

5.3.3 Ứng dụng của RSA trong chứng chỉ số

Chứng chỉ số X.509 là chuẩn định dạng cho chứng chỉ khóa công khai trong PKI. RSA đóng vai trò quan trọng trong cả quá trình phát hành và sử dụng chứng chỉ:

- Phát hành chứng chỉ: CA sử dụng khóa riêng RSA để ký chứng chỉ
- Xác thực: Khóa công khai của CA được sử dụng để xác minh chữ ký trên chứng chỉ
- Cấu trúc chứng chỉ: Chứng chỉ chứa thông tin về chủ sở hữu, khóa công khai RSA và thông tin khác

Theo DigiCert, một trong những CA lớn nhất thế giới, độ dài khóa RSA tiêu chuẩn cho chứng chỉ tăng từ 1024 bit (trước 2012) lên 2048 bit và cao hơn (từ 2012 đến nay) nhằm đảm bảo an toàn trước các mối đe dọa mới.

5.4 Truyền thông an toàn

5.4.1 VPN (Virtual Private Network)

VPN sử dụng RSA trong các thành phần sau:

- Xác thực người dùng: Chứng chỉ RSA được sử dụng để xác thực danh tính
- Thiết lập đường hầm mã hóa: RSA được sử dụng trong giai đoạn trao đổi khóa
- Xác thực máy chủ VPN: Ngăn chặn tấn công man-in-the-middle

Các giao thức VPN phổ biến như OpenVPN và IPSec thường sử dụng RSA cho xác thực và thiết lập kết nối ban đầu. Theo nghiên cứu của AV-TEST, đến năm 2023, khoảng 68% các dịch vụ VPN doanh nghiệp sử dụng RSA với độ dài khóa tối thiểu 2048 bit.

5.4.2 Email bảo mật

RSA đóng vai trò quan trọng trong các tiêu chuẩn email bảo mật:

- S/MIME (Secure/Multipurpose Internet Mail Extensions): Tiêu chuẩn cho email an toàn, sử dụng RSA cho chữ ký số và mã hóa
- PGP/OpenPGP: Hệ thống bảo mật email phổ biến, sử dụng RSA cho quản lý khóa và mã hóa

Quy trình email bảo mật với RSA thường bao gồm:

- Người gửi mã hóa nội dung email bằng khóa đối xứng
- Khóa đối xứng được mã hóa bằng khóa công khai RSA của người nhận
- Người gửi có thể ký email bằng khóa riêng RSA của mình
- Người nhận sử dụng khóa riêng để giải mã khóa đối xứng
- Người nhận kiểm tra chữ ký bằng khóa công khai của người gửi

Theo báo cáo từ Proton Mail, tỷ lệ email được mã hóa đầu-cuối (end-to-end) đã tăng từ khoảng 5% (2018) lên khoảng 20% (2023) trên toàn cầu, và hầu hết đều sử dụng RSA hoặc các thuật toán dựa trên đường cong elliptic.

5.4.3 Truyền tin nhắn bảo mật

Các ứng dụng nhắn tin bảo mật như Signal, WhatsApp và Telegram sử dụng mã hóa đầu-cuối để bảo vệ nội dung tin nhắn. Mặc dù phần lớn đã chuyển sang sử dụng giao thức Signal (dựa trên đường cong elliptic), RSA vẫn được sử dụng trong một số thành phần:

- Xác thực ban đầu: Xác minh danh tính người dùng
- Trao đổi khóa phiên: Thiết lập phiên trò chuyện an toàn
- Xác minh thiết bị: Xác minh các thiết bị của người dùng

Theo Signal.org, phiên bản mới nhất của giao thức Signal kết hợp các thuật toán khác nhau để cải thiện hiệu suất và bảo mật, nhưng RSA vẫn được sử dụng trong một số trường hợp đặc biệt khi cần khả năng tương thích ngược hoặc xác thực mạnh.

5.5 Tích hợp vào phần mềm và hệ điều hành

5.5.1 Xác thực phần mềm và cập nhật an toàn

RSA đóng vai trò quan trọng trong việc đảm bảo an toàn cho quy trình phân phối và cập nhật phần mềm:

- Ký phần mềm: Nhà phát triển sử dụng khóa riêng RSA để ký các gói phần mềm
- Xác minh tính toàn vẹn: Hệ điều hành kiểm tra chữ ký bằng khóa công khai của nhà phát triển
- Cập nhật an toàn: Cập nhật được xác minh tương tự trước khi cài đặt

Microsoft, Apple và các nhà phát triển phần mềm lớn khác đều sử dụng RSA trong quy trình ký số phần mềm. Theo Microsoft, tất cả các ứng dụng Windows Store và các cập nhật Windows đều được ký bằng RSA (hoặc ECC trong một số trường hợp gần đây).

5.5.2 Tích hợp RSA vào hệ điều hành

RSA được tích hợp sâu vào các hệ điều hành hiện đại:

- Windows: RSA được tích hợp vào CryptoAPI và sau này là Cryptography Next Generation.
- Linux: Thông qua OpenSSL và các thư viện mật mã khác
- macOS/iOS: Thông qua Security Framework và CommonCrypto

Các hệ điều hành cung cấp API cho các ứng dụng sử dụng RSA cho các tác vụ bảo mật. Ví dụ, Android KeyStore cho phép các ứng dụng sử dụng RSA cho xác thực và mã hóa mà không cần xử lý trực tiếp các khóa nhạy cảm.

5.5.3 Bảo vệ dữ liệu trong ứng dụng

RSA được sử dụng để bảo vệ dữ liệu nhạy cảm trong các ứng dụng:

- Lưu trữ thông tin xác thực: Mã hóa mật khẩu và thông tin đăng nhập
- Bảo vệ khóa đối xứng: Mã hóa các khóa được sử dụng cho dữ liệu lớn
- Xác thực API: Bảo vệ giao tiếp giữa client và server

Theo OWASP (Open Web Application Security Project), việc sử dụng RSA với độ dài khóa tối thiểu 2048 bit là một trong những biện pháp bảo mật cơ bản cho ứng dụng xử lý dữ liệu nhạy cảm.

5.5.4 Secure Boot và Trusted Platform Module (TPM)

Secure Boot và TPM sử dụng RSA để đảm bảo tính toàn vẹn của hệ thống:

- Secure Boot: Sử dụng chữ ký RSA để xác minh bootloader và kernel
- TPM: Sử dụng RSA cho các chức năng như đo lường hệ thống và xác thực từ xa

Microsoft yêu cầu Secure Boot cho các thiết bị Windows, sử dụng khóa RSA 2048 bit để ký các thành phần hệ thống quan trọng. TPM 2.0 hỗ trợ RSA với độ dài khóa lên đến 3072 bit.

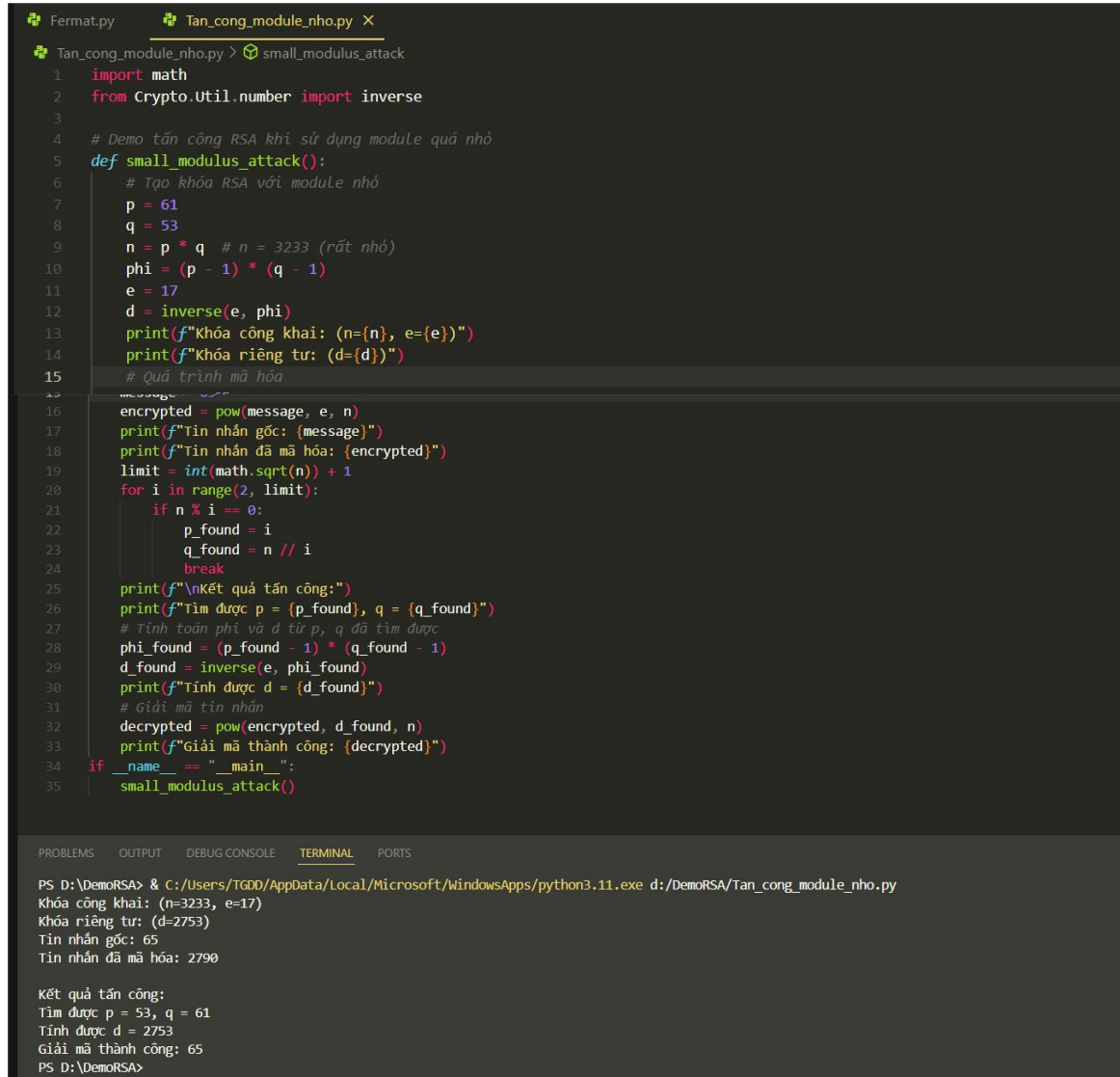
5.6 Kết chương

RSA đã và đang đóng vai trò nền tảng trong cơ sở hạ tầng bảo mật thông tin toàn cầu. Mặc dù đang dần được bổ sung bởi các thuật toán hiệu quả hơn cho các ứng dụng cụ thể, RSA vẫn sẽ tiếp tục là một phần quan trọng của hệ sinh thái mật mã trong nhiều năm tới, đặc biệt là trong các hệ thống đã triển khai.

CHƯƠNG 6. DEMO

6.1 Tấn công module nhỏ (Small modulus attack)

Khi sử dụng khóa RSA có độ dài quá ngắn, việc phân tích ra các thừa số nguyên tố của n trở nên khả thi:



```
1 import math
2 from Crypto.Util.number import inverse
3
4 # Demo tấn công RSA khi sử dụng module quá nhỏ
5 def small_modulus_attack():
6     # Tạo khóa RSA với module nhỏ
7     p = 61
8     q = 53
9     n = p * q # n = 3233 (rất nhỏ)
10    phi = (p - 1) * (q - 1)
11    e = 17
12    d = inverse(e, phi)
13    print(f"Khóa công khai: (n={n}, e={e})")
14    print(f"Khóa riêng tư: (d={d})")
15    # Quá trình mã hóa
16    message = 65
17    encrypted = pow(message, e, n)
18    print(f"Tin nhắn gốc: {message}")
19    print(f"Tin nhắn đã mã hóa: {encrypted}")
20    limit = int(math.sqrt(n)) + 1
21    for i in range(2, limit):
22        if n % i == 0:
23            p_found = i
24            q_found = n // i
25            break
26    print(f"\nKết quả tấn công:")
27    print(f"Tìm được p = {p_found}, q = {q_found}")
28    # Tính toán phi và d từ p, q đã tìm được
29    phi_found = (p_found - 1) * (q_found - 1)
30    d_found = inverse(e, phi_found)
31    print(f"Tính được d = {d_found}")
32    # Giải mã tin nhắn
33    decrypted = pow(encrypted, d_found, n)
34    print(f"Giải mã thành công: {decrypted}")
35 if __name__ == "__main__":
36     small_modulus_attack()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\DemoRSA> & C:/Users/TGDD/AppData/Local/Microsoft/windowsApps/python3.11.exe d:/DemoRSA/Tan_cong_module_nho.py

Khóa công khai: (n=3233, e=17)
Khóa riêng tư: (d=2753)
Tin nhắn gốc: 65
Tin nhắn đã mã hóa: 2790

Kết quả tấn công:
Tìm được p = 53, q = 61
Tính được d = 2753
Giải mã thành công: 65
PS D:\DemoRSA>

Hình 2. Tấn công module nhỏ

6.2 Tấn công số mũ nhỏ (Small exponent attack - Håstad's broadcast attack)

Khi sử dụng cùng một tin nhắn và số mũ nhỏ (thường là $e=3$) để gửi tin nhắn đến nhiều người nhận khác nhau, dễ bị tấn công:


```

1  from sympy.ntheory.modular import crt
2  import gmpy2
3  def hastad_broadcast_attack():
4      # Giả sử 3 người nhận, mỗi người có module RSA khác nhau nhưng cùng sử dụng e=3
5      n1 = 589 * 601 # = 353989
6      n2 = 557 * 613 # = 341441
7      n3 = 569 * 631 # = 359039
8      e = 3 # Số mũ mã hóa nhỏ
9      # Tin nhắn gốc (cần nhỏ hơn min(n1,n2,n3)^(1/e) để tránh wrap-around)
10     message = 12345
11     # Mô phỏng quá trình mã hóa thành 3 bản mã khác nhau
12     c1 = pow(message, e, n1)
13     c2 = pow(message, e, n2)
14     c3 = pow(message, e, n3)
15     print(f"Tin nhắn gốc: {message}")
16     print(f"Mã hóa với n1: {c1}")
17     print(f"Mã hóa với n2: {c2}")
18     print(f"Mã hóa với n3: {c3}")
19     # Giả sử kẻ tấn công bắt được 3 bản mã và biết các giá trị n1, n2, n3 và e, sau đó thực hiện CRT
20     moduli = [n1, n2, n3]
21     remainders = [c1, c2, c3]
22     x, _ = crt(moduli, remainders)
23     recovery_message, is_perfect_cube = gmpy2.iroot(x, e)
24     print(f"\nKết quả tấn công:")
25     print(f"Tính toán được giá trị m^e = {x}")
26     if is_perfect_cube:
27         print(f"Tin nhắn khôi phục được: {int(recovery_message)}")
28         if int(recovery_message) == message:
29             print("Tấn công thành công!")
30         else:
31             print("Tấn công thất bại - tin nhắn khôi phục không chính xác.")
32     else:
33         print("Tấn công thất bại - không tìm được căn bậc 3 chính xác.")
34
35 if __name__ == "__main__":
36     hastad_broadcast_attack()

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS D:\DemoRSA> & C:/Users/TGDD/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/DemoRSA/Tan_cong_so_mu_nho.py
Tin nhắn gốc: 12345
Mã hóa với n1: 93963
Mã hóa với n2: 104109
Mã hóa với n3: 167469

Kết quả tấn công:
Tính toán được giá trị m^e = 1881365963625
Tin nhắn khôi phục được: 12345
Tấn công thành công!
PS D:\DemoRSA>

```

Hình 3. Tấn công số mũ nhỏ

6.3 Tấn công thông tin phụ (Common modulus attack)

Khi cùng một tin nhắn được mã hóa với cùng một module n nhưng với các số mũ e khác nhau:

```

1 import math
2 from Crypto.Util.number import inverse
3 def extended_gcd(a, b):
4     """
5     Thuật toán Euclid mở rộng để tìm gcd(a,b) và hệ số x,y của phương trình ax + by = gcd(a,b)
6     """
7     if a == 0:
8         return b, 0, 1
9     else:
10         gcd, x1, y1 = extended_gcd(b % a, a)
11         x = y1 - (b // a) * x1
12         y = x1
13         return gcd, x, y
14 def common_modulus_attack():
15     # Khởi tạo các tham số RSA
16     p = 61
17     q = 53
18     n = p * q # n = 3233
19     # Hai số mũ mã hóa khác nhau nhưng cùng nguyên tố với phi(n)
20     e1 = 17
21     e2 = 23
22     # Tin nhắn gốc
23     message = 123
24     # Mã hóa cùng một tin nhắn với hai số mũ khác nhau
25     c1 = pow(message, e1, n) # Mã hóa với e1
26     c2 = pow(message, e2, n) # Mã hóa với e2
27     print(f"Module n = {n}")
28     print(f"tin nhắn gốc: {message}")
29     print(f"Mã hóa với e1={e1}: c1 = {c1}")
30     print(f"Mã hóa với e2={e2}: c2 = {c2}")
31     # Tấn công bắt đầu từ đây
32     # Ké tấn công biết e1, e2, c1, c2
33     # Tìm gcd(e1, e2) và các hệ số s1, s2 sao cho:
34     # s1*e1 + s2*e2 = gcd(e1, e2)
35     gcd, s1, s2 = extended_gcd(e1, e2)
36
37     print(f"\nKết quả tính toán:")
38     print(f"gcd(e1, e2) = {gcd}")
39     print(f"s1 = {s1}, s2 = {s2}")
40     print(f"Kiểm tra: {s1}*{e1} + {s2}*{e2} = {s1*e1 + s2*e2}")
41
42     # Nếu gcd(e1, e2) = 1, chúng ta có thể khôi phục tin nhắn
43     if gcd == 1:
44         # Nếu s2 < 0, cần chuyển đổi
45         if s2 < 0:
46             s2 = -s2
47             c2 = inverse(c2, n)
48         else:
49             s1 = -s1
50             c1 = inverse(c1, n)
51
52         # Khôi phục tin nhắn: m = (c1*s1 + c2*s2) mod n
53         if s1 < 0:
54             part1 = pow(c1, abs(s1), n)
55             part1_inv = inverse(part1, n)
56             recovered = (part1_inv * pow(c2, s2, n)) % n
57         else:
58             recovered = (pow(c1, s1, n) * pow(c2, s2, n)) % n
59
60     print(f"\nKết quả tấn công:")
61     print(f"tin nhắn khôi phục được: {recovered}")
62     if recovered == message:
63         print("Tấn công thành công!")
64     else:
65         print("Tấn công thất bại - tin nhắn khôi phục không chính xác.")
66
67     # Nếu tấn công thất bại - gcd(e1, e2) != 1
68     print(f"tin nhắn công thất bại - gcd(e1, e2) = {gcd} khác 1")
69     print("Trong trường hợp này, chúng ta chỉ có thể khôi phục m* gcd mod n")
70
71 if __name__ == "__main__":
72     common_modulus_attack()

```

Hình 4. Tấn công thông tin phụ

6.4 Tấn công số p, q gần nhau (Fermat's factorization attack)

Khi p và q được chọn quá gần nhau, có thể sử dụng phương pháp phân tích Fermat để tìm ra chúng.

```

1 import math
2 from Crypto.Util.number import inverse
3
4 def fermat_factorization(n):
5     # Kiểm tra n có phải số lẻ không
6     if n % 2 == 0:
7         return 2, n//2
8     # Bắt đầu với căn bậc hai của n làm tròn lên
9     a = math.ceil(math.sqrt(n))
10    b2 = a*a - n
11    # Tìm b2 là số chính phương
12    while not math.isqrt(b2)**2 == b2:
13        a += 1
14        b2 = a*a - n
15    # Khi tìm được b là số chính phương
16    b = math.isqrt(b2)
17    # Tính p và q
18    p = a - b
19    q = a + b
20    return p, q
21
22 def demonstrate_fermat_attack():
23     # Ví dụ với 2 số p và q gần nhau
24     p = 1000000007 # p là số nguyên tố
25     q = 1000000009 # q là số nguyên tố gần p
26     n = p * q
27     e = 65537 # Số mũ mã hóa thông dụng
28     phi = (p-1) * (q-1)
29     d = inverse(e, phi) # Số mũ giải mã
30     message = 12345 # Tin nhắn mẫu
31     # Mã hóa
32     encrypted = pow(message, e, n)
33     print(f"Tạo khóa RSA với p và q gần nhau:")
34     print(f"p = {p}")
35     print(f"q = {q}")
36     print(f"n = p*q = {n}")
37     print(f"tin nhắn gốc: {message}")
38     print(f"tin nhắn đã mã hóa: {encrypted}")
39     # Giả sử kẻ tấn công chỉ biết n và e
40     print("\nBắt đầu tấn công với n và e:")
41     print(f"Sử dụng phương pháp phân tích Fermat...")
42     # Thực hiện tấn công
43     start_time = __import__('time').time()
44     p_found, q_found = fermat_factorization(n)
45     end_time = __import__('time').time()
46
47     print(f"Tìm được: p = {p_found}, q = {q_found}")
48     print(f"Thời gian thực hiện: {end_time - start_time:.6f} giây")
49
50     # Kiểm tra kết quả
51     if p_found * q_found == n:
52         print("Phân tích thành công!")
53
54     # Tính phi(n) và d từ p, q đã tìm được
55     phi_found = (p_found - 1) * (q_found - 1)
56     d_found = inverse(e, phi_found)
57
58     # Giải mã tin nhắn
59     decrypted = pow(encrypted, d_found, n)
60     print(f"tin nhắn giải mã được: {decrypted}")
61
62     if decrypted == message:
63         print("Khôi phục tin nhắn thành công!")
64     else:
65         print("Khôi phục tin nhắn thất bại!")
66
67     else:
68         print("Phân tích thất bại!")
69
70 if __name__ == "__main__":
71     demonstrate_fermat_attack()

```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```

n = p*q = 1000000016000000063
Tin nhắn gốc: 12345
Tin nhắn đã mã hóa: 58539820554885323

Bắt đầu tấn công với n và e:
Sử dụng phương pháp phân tích Fermat...
Tìm được: p = 1000000007, q = 1000000009
Thời gian thực hiện: 0.000000 giây
Phân tích thành công!
Tin nhắn giải mã được: 12345
Khôi phục tin nhắn thành công!
PS D:\DemoRSA>

```

Hình 5. Tấn công p, q gần nhau

6.5 Tấn công Wiener trên số mũ riêng tư d quá nhỏ

Khi số mũ riêng tư d quá nhỏ so với n , có thể sử dụng tấn công Wiener để khôi phục d từ e và n :

```

1 from fractions import Fraction
2 import math
3 from Crypto.Util.number import inverse
4
5 def rational_to_contfrac(x, y):
6     """
7     Chuyển đổi x/y thành dạng phân số liên tục
8     """
9     a = x // y
10    pquotients = [a]
11    while a * y != x:
12        x, y = y, x - a * y
13        a = x // y
14        pquotients.append(a)
15    return pquotients
16
17 def convergents_from_contfrac(frac):
18     """
19     Tính các phân số hội tụ từ phân số liên tục
20     """
21    convs = []
22    for i in range(len(frac)):
23        convs.append(contfrac_to_rational(frac[0:i]))
24    return convs
25
26 def contfrac_to_rational(frac):
27     """
28     Chuyển đổi phân số liên tục thành phân số thường
29     """
30    if len(frac) == 0:
31        return (0, 1)
32    elif len(frac) == 1:
33        return (frac[0], 1)
34    else:
35        remainder = frac[1:len(frac)]
36        (p, q) = contfrac_to_rational(remainder)
37        return (frac[0] * p + q, p)
38
39 def is_perfect_square(n):
40     """
41     Kiểm tra xem n có phải là số chính phương không
42     """
43    h = n & 0xF # Lấy 4 bit cuối
44    if h > 9:
45        return False
46
47    if h != 2 and h != 3 and h != 5 and h != 6 and h != 7 and h != 8:
48        t = int(math.isqrt(n))
49        return t * t == n
50    return False
51
52 def wiener_attack(e, n):
53     """
54     Thực hiện tấn công Wiener để tìm d khi d quá nhỏ
55     """
56    # Tính phân số liên tục của e/n
57    frac = rational_to_contfrac(e, n)
58    convergents = convergents_from_contfrac(frac)
59
60    for (k, d) in convergents:
61        # Bỏ qua trường hợp d chẵn và k=0
62        if k == 0 or d % 2 == 0 or d < 2:
63            continue
64        # Kiểm tra điều kiện ed = 1 (mod φ(n))
65        # Tức là ed = 1 + kφ(n) với k nào đó
66        # Do đó φ(n) = (ed-1)/k
67        phi = (e * d - 1) // k
68
69        # Tính S = n - phi + 1 = p + q
70        s = n - phi + 1
71
72        # Tính Delta = S^2 - 4n
73        delta = s * s - 4 * n
74
75        # Nếu Delta là số chính phương, chúng ta đã tìm đúng d
76        if delta > 0 and is_perfect_square(delta):
77            p = (s + math.isqrt(delta)) // 2
78            q = (s - math.isqrt(delta)) // 2
79
80            # Kiểm tra p*q = n để xác nhận
81            if p * q == n:
82                return d, p, q
83
84    return None, None, None

```

```

84
85 def demonstrate_wiener_attack():
86
87     # Tạo khóa RSA với d nhỏ (điểm yếu)
88     p = 1013
89     q = 2039
90     n = p * q # = 2065507
91     phi = (p-1) * (q-1) # = 2062456
92
93     # Chọn một d nhỏ (điểm yếu của mật mã)
94     d = 3 # Số mũ riêng tư nhỏ
95
96     # Tính e từ d (thường trong RSA chúng ta chọn e và tính d,
97     # nhưng ở đây chúng ta chọn d nhỏ và tính e để minh họa điểm yếu)
98     e = inverse(d, phi)
99
100    # Thông điệp mẫu
101    message = 12345
102
103    # Mã hóa
104    encrypted = pow(message, e, n)
105
106    print(f"Tạo khóa RSA với d nhỏ:")
107    print(f"p = {p}, q = {q}")
108    print(f"n = p*q = {n}")
109    print(f"phi(n) = {phi}")
110    print(f"d = {d} (nhỏ so với n^0.25)")
111    print(f"e = {e}")
112    print(f"Thông điệp gốc: {message}")
113    print(f"Thông điệp đã mã hóa: {encrypted}")
114
115    # Giả sử kẻ tấn công chỉ biết n và e
116    print("\nBắt đầu tấn công Wiener với n và e:")
117
118    # Thực hiện tấn công Wiener
119    recovered_d, recovered_p, recovered_q = wiener_attack(e, n)
120
121    if recovered_d:
122        print("Tấn công thành công!")
123        print(f"Khôi phục được d = {recovered_d}")
124        print(f"Khôi phục được p = {recovered_p}")
125        print(f"Khôi phục được q = {recovered_q}")
126
127    # Giải mã thông điệp
128    decrypted = pow(encrypted, recovered_d, n)
129    print(f"Thông điệp giải mã được: {decrypted}")
130
131    if decrypted == message:
132        print("Khôi phục thông điệp thành công!")
133    else:
134        print("Khôi phục thông điệp thất bại!")
135    else:
136        print("Tấn công thất bại - không tìm được d.")
137
138 if __name__ == "__main__":
139     demonstrate_wiener_attack()

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

e = 1374971
Thông điệp gốc: 12345
Thông điệp đã mã hóa: 1453879

Bắt đầu tấn công Wiener với n và e:
Tấn công thành công!
Khôi phục được d = 3
Khôi phục được p = 2039
Khôi phục được q = 1013
Thông điệp giải mã được: 12345
Khôi phục thông điệp thành công!
PS D:\DemoRSA>

```

Hình 6. Tấn công Wiener

6.6 Ứng dụng xác thực SSH bằng khóa công khai

Thay vì sử dụng mật khẩu (dễ bị tấn công brute force), SSH sử dụng cặp khóa RSA để xác thực.

- Chuẩn bị 2 máy ảo: Kali Linux và Ubuntu

The image shows two overlapping windows. The background window is a Kali Linux terminal with the following output:

```

valid_lft forever preferred_lft forever

$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.100.3 netmask 255.255.255.0 broadcast 192.168.100.255
    ether 00:0c:29:b6:1b:61 txqueuelen 1000 (Ethernet)
    RX packets 1 bytes 60 (60.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 131 bytes 22901 (22.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 480 (480.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 480 (480.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

The foreground window is a Windows Command Prompt with the following output:

```

Microsoft Windows [Version 10.0.27813.1000]
(c) Microsoft Corporation. All rights reserved.

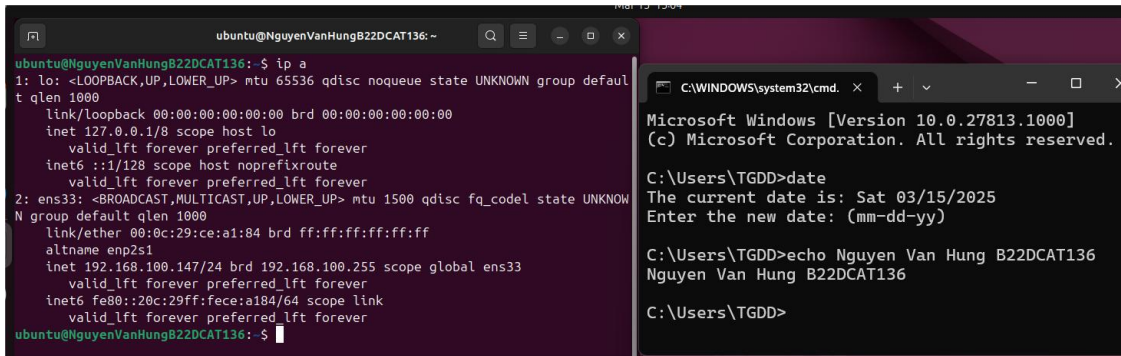
C:\Users\TGDD>date
The current date is: Sat 03/15/2025
Enter the new date: (mm-dd-yy)

C:\Users\TGDD>echo Nguyen Van Hung B22DCAT136
Nguyen Van Hung B22DCAT136

C:\Users\TGDD>

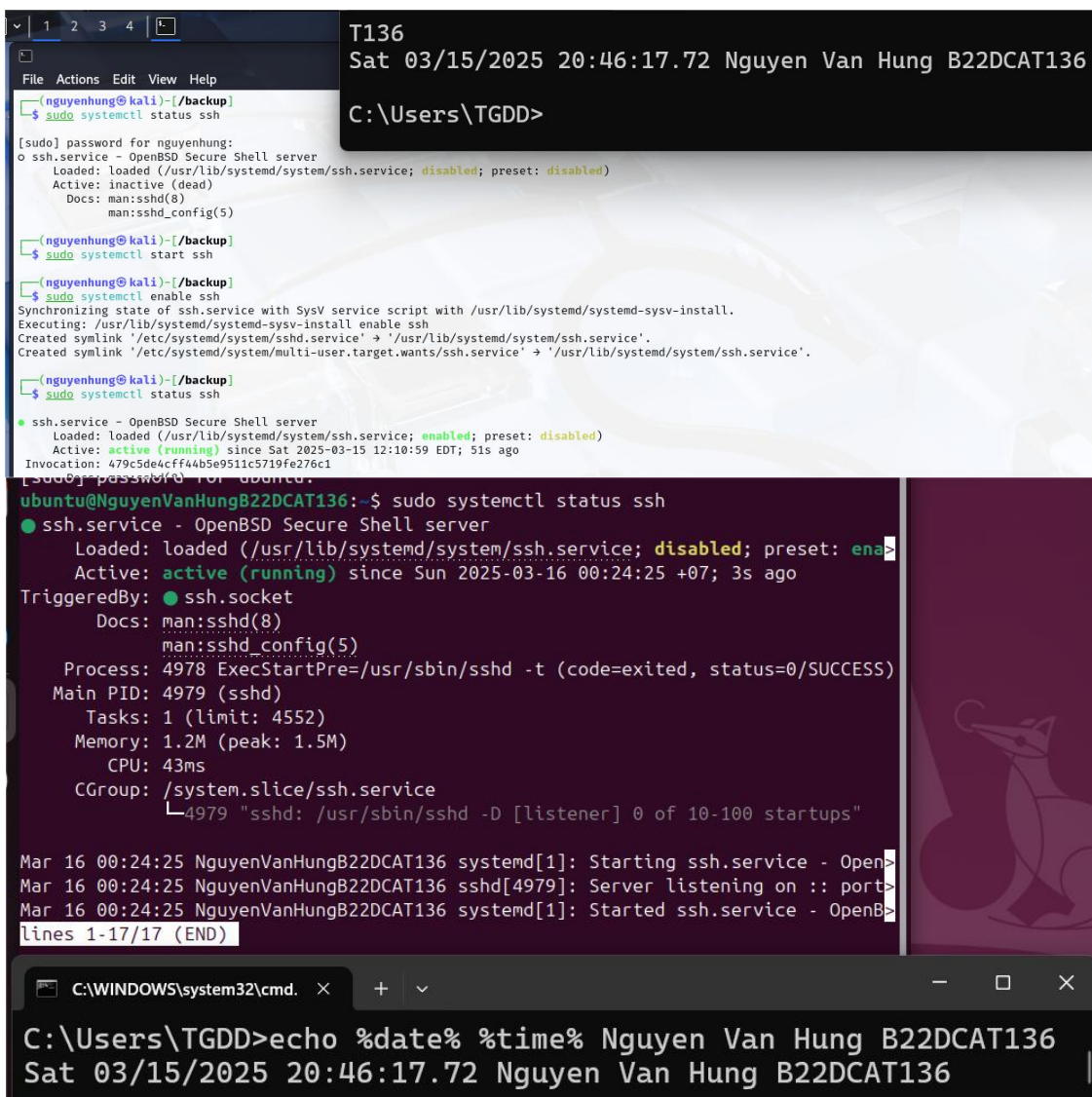
```


Hình 7. Máy ảo Kali Linux



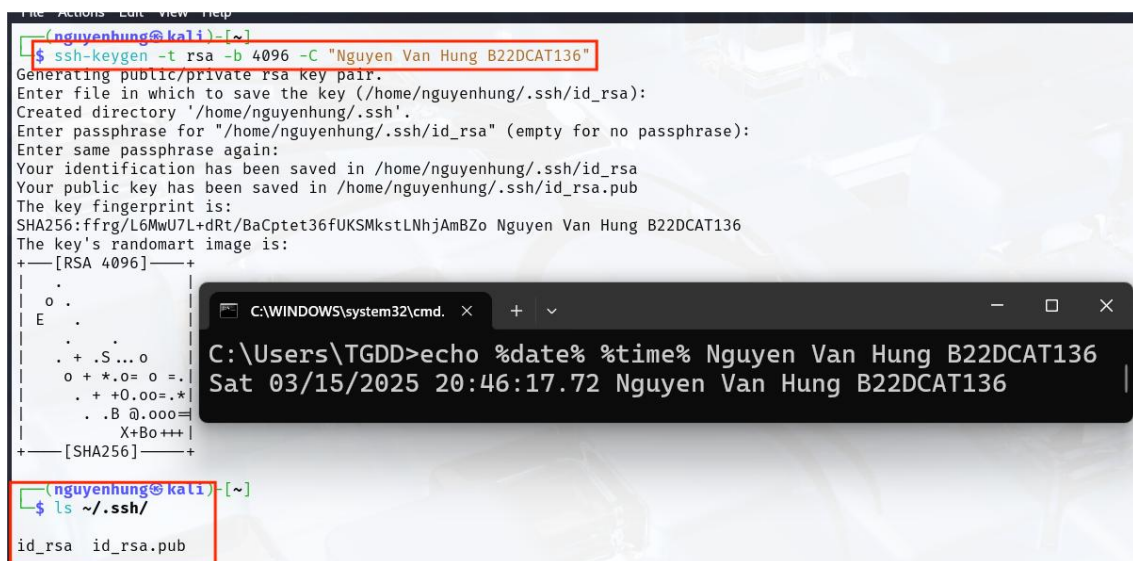
Hình 8. Máy ảo Ubuntu

- Đảm bảo cả 2 máy ảo đều cài đặt SSH:



Hình 9. Đảm bảo cài đặt SSH trên 2 máy ảo

- Tạo Secure Shell Keys trên máy Kali Linux:
`ssh-keygen -t rsa -b 4096 -C "Nguyen Van Hung B22DCAT136"`
- Giải thích:
 - o `ssh-keygen`: Lệnh tạo cặp khóa SSH.
 - o `-t rsa`: Chọn thuật toán mã hóa RSA (một trong những thuật toán phổ biến nhất cho SSH).
 - o `-b 4096`: Tạo khóa có độ dài 4096 bit (càng dài, càng bảo mật hơn).
 - o `-C "Nguyen Van Hung B22DCAT136"`: Thêm phần mô tả cho key (comment) – đặt tên và mã sinh viên để dễ nhận diện.
- Sau khi chạy lệnh, hệ thống sẽ hỏi nơi lưu trữ khóa, nếu nhấn enter, nó sẽ lưu tại mặc định:
`~/.ssh/id_rsa` # Khóa riêng tư
`~/.ssh/id_rsa.pub` # Khóa công khai
- Kiểm tra key đã tạo ra: `ls ~/.ssh/`



Hình 10. Tạo Secure Shell Keys

- Sao chép khóa công khai sang máy Ubuntu:
`ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@192.168.100.147`

```
File Actions Edit View Help
(nguyenhung@kali)-[~]
$ ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@192.168.100.147
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/nguyenhung/.ssh/id_rsa.pub"
The authenticity of host '192.168.100.147 (192.168.100.147)' can't be established.
ED25519 key fingerprint is SHA256:kyzknCVQshpqG6Kxqein3i827S2z5JekPBz+LXg0ViQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ubuntu@192.168.100.147's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh -i /home/nguyenhung/.ssh/id_rsa 'ubuntu@192.168.100.147'"
and check to make sure that only the key(s) you wanted were added.
```

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\TGDD>echo %date% %time% Nguyen Van Hung B22DCAT136
Sat 03/15/2025 20:46:17.72 Nguyen Van Hung B22DCAT136
```

Hình 11. Sao chép khóa công khai sang ubuntu

- Kiểm tra kết nối SSH không cần mật khẩu từ Kali Linux vào Ubuntu victim:

ssh -i ~/.ssh/id_rsa ubuntu@192.168.100.147

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\TGDD>echo %date% %time% Nguyen Van Hung B22DCAT136
Sat 03/15/2025 20:46:17.72 Nguyen Van Hung B22DCAT136

(nguyenhung@kali)-[~]
$ ssh -i ~/.ssh/id_rsa ubuntu@192.168.100.147

Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-17-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

99 updates can be applied immediately.
48 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
```

Hình 12. Kiểm tra kết quả SSH

Nếu kết nối thành công mà không yêu cầu mật khẩu, nghĩa là thiết lập SSH Key đúng cách.

KẾT LUẬN

Qua bài cá nhân này, ta hiểu sâu hơn về RSA, cụ thể:

- Hiểu nguyên lý RSA: Bài toán phân tích thừa số, quá trình sinh khóa, mã hóa - giải mã, ký số và xác minh. Nhờ vậy có thể nắm rõ cách thức vận hành của một hệ mật khóa công khai
- Nhận diện lỗ hổng triển khai qua các demo tấn công, ta thấy RSA mạnh nhưng cũng dễ bị tổn thương khi gặp các trường hợp như:
 - Sinh khóa thiếu entropy hoặc chọn p, q, qp, q quá gần nhau
 - Dùng exponent nhỏ mà không có padding an toàn
 - Không bảo vệ constant-time, không dùng blinding hay masking
- Hiểu rõ các ứng dụng của RSA trong thực tế, như đảm bảo toàn vẹn dữ liệu, xác thực, chữ ký số, ...

Có thể thấy rằng, RSA dù đã ra đời gần nửa thế kỷ vẫn là nền tảng then chốt, nhưng chỉ vững chắc khi mọi khâu - từ sinh khóa đến xử lý padding và chống kênh phụ - đều được triển khai cẩn trọng. Bài tập này giúp ta củng cố và nâng cao kiến thức về RSA.

TÀI LIỆU THAM KHẢO

- [1] Đỗ Xuân Chợt, Bài giảng mật mã học cơ sở, Học viện Công nghệ Bưu chính Viễn Thông, 2021
- [2] R.L. Rivest, A. Shamir, and L. Adleman* , A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, 1978
- [3] Boneh, D., & Shoup, V., A Graduate Course in Applied Cryptography, 2020
- [4] Katz., & Lindell, Y. , Introduction to Modern Cryptography, 2021
- [5] Stallings, W., Cryptography and Network Security: Principles and Practice, 2022
- [6] NIST Special Publication 800-57 Part 1 Revision 5, 2020
- [7] FIPS 186-5, Digital Signature Standard (DSS), 2023
- [8] RFC 8017 (PKCS #1 v2.2), 2016
- [9] RFC 8446 (TLS 1.3), 2018
- [10] Boneh, D. , Twenty Years of Attacks on the RSA Cryptosystem, 1999
- [11] Lenstra, A. K., & Verheul, E. R., Selecting Cryptographic Key Sizes (2001)
- [12] Kleinjung, T., et al., Factorization of a 768-bit RSA modulus version 1.4, 2010
- [13] Paul C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, 1996
- [14] Bernstein, D. J., & Lange, T., Post-Quantum Cryptography, 2017
- [15] NIST, Post-Quantum Cryptography, 2022
- [16] Gidney, C., & Ekerå, M. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, 2019
- [17] Cooper, D., et al. , Internet X.509 Public Key Infrastructure, 2008
- [18] OWASP, Application Security Verification Standard (ASVS), 2023
- [19] Nguyễn Khanh Văn, Trần Đức Khánh, Giáo trình An toàn thông tin, Đại học Bách Khoa Hà Nội, 2012
- [20] Cloudflare, TLS Encryption Adoption Report, 2023
- [21] Let's Encrypt, SSL Certificate Statistics, 2023
- [22] Gartner, Quantum Computing Security Impact, 2023