Cryptanalysis on Asymmetric Ciphers — RSA & RSA-Based Signatures

1. Bối cảnh

RSA là thuật toán mã hóa được sử dụng rộng rãi ở nhiều công nghệ:

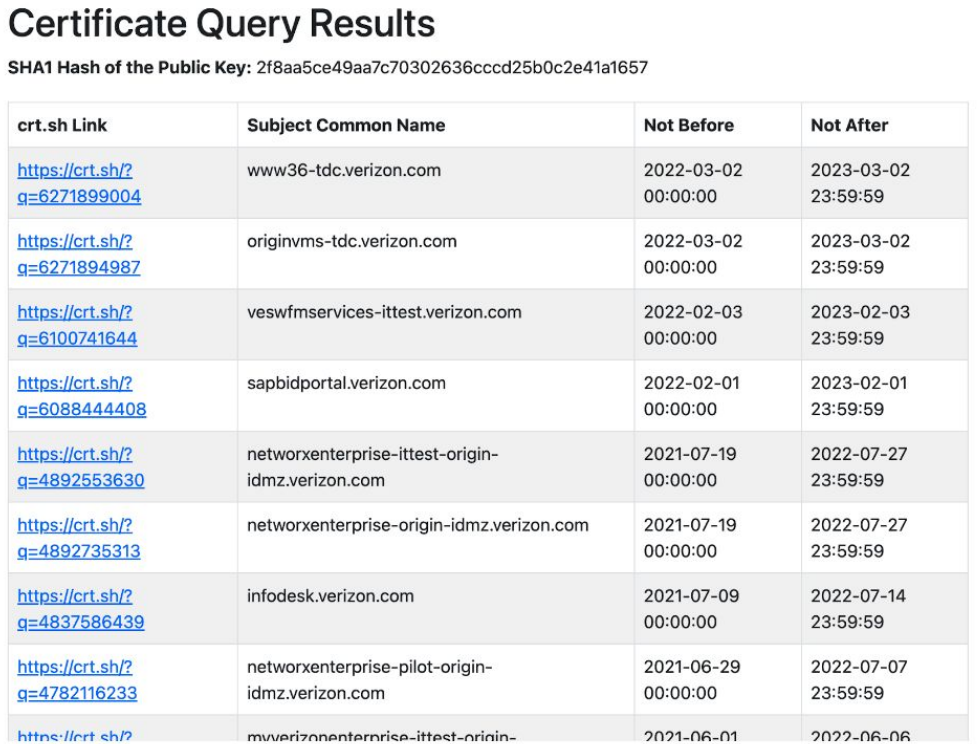
- Giao thức mạng như SSL/TLS, SSH, …
- Dịch vụ VPN
- Smart Cards, các thiết bị IoT

Vấn đề: tuy phổ biến, RSA vẫn tiềm ẩn rủi ro cao nếu triển khai sai hoặc tham số đầu vào không đạt chuẩn.

Rủi ro

Nếu có thể phân tích thừa số N và tìm được p,q, hệ thống sẽ gặp những rủi ro sau:

- Khôi phục lại được private key
- Mất tính bảo mật: dịch lại các dữ liệu như thông tin đăng nhập, các khóa phiên, …
- Mất tính tính toàn vẹn, tính xác thực: giả mạo chữ ký
- Nếu các cặp khóa được tái sử dụng trên nhiều service của hệ thống thì có thể bị ảnh hưởng



Certificate Query Results

SHA1 Hash of the Public Key: 2f8aa5ce49aa7c70302636cccd25b0c2e41a1657

crt.sh Link	Subject Common Name	Not Before	Not After
https://crt.sh/?q=6271899004	www36-tdc.verizon.com	2022-03-02 00:00:00	2023-03-02 23:59:59
https://crt.sh/?q=6271894987	originvms-tdc.verizon.com	2022-03-02 00:00:00	2023-03-02 23:59:59
https://crt.sh/?q=6100741644	veswfmservices-ittest.verizon.com	2022-02-03 00:00:00	2023-02-03 23:59:59
https://crt.sh/?q=6088444408	sapbidportal.verizon.com	2022-02-01 00:00:00	2023-02-01 23:59:59
https://crt.sh/?q=4892553630	networxenterprise-ittest-origin-idmz.verizon.com	2021-07-19 00:00:00	2022-07-27 23:59:59
https://crt.sh/?q=4892735313	networxenterprise-origin-idmz.verizon.com	2021-07-19 00:00:00	2022-07-27 23:59:59
https://crt.sh/?q=4837586439	infodesk.verizon.com	2021-07-09 00:00:00	2022-07-14 23:59:59
https://crt.sh/?q=4782116233	networxenterprise-pilot-origin-idmz.verizon.com	2021-06-29 00:00:00	2022-07-07 23:59:59
https://crt.sh/?	myverizonenterprise-ittest-origin-	2021-06-01	2022-06-06

Security Goals

- Bảo vệ dữ liệu: đảm bảo người được phép mới được đọc dữ liệu.

- Tính toàn vẹn: dữ liệu gốc không bị thay đổi khi được mã hóa.

- Tính xác thực: biết chắc chắn dữ liệu đến từ đâu, xác định được người gửi.

- Chống các cuộc tấn công khi triển khai thực tế.

2. Các câu hỏi nghiên cứu và giả thuyết

- RQ1: Trong kịch bản triển khai thực tế (TLS, JWT, code signing), đâu là các điểm yếu phổ biến nhất liên quan tới RSA và chúng dẫn tới hậu quả gì (forgery, key recovery, signature replay)?

2. Các câu hỏi nghiên cứu và giả thuyết

- RQ2: Các tấn công Bleichenbacher (padding oracle) và timing attacks còn khả thi trên các stack hiện đại (OpenSSL/LibreSSL/BoringSSL) khi gặp cấu hình cũ hoặc lỗi không?

2. Các câu hỏi nghiên cứu và giả thuyết

- Giả thuyết: Các vụ compromise thực tế thường xảy ra do: xử lý padding không an toàn (PKCS#1 v1.5), RNG yếu tạo primes dễ đoán, misuse của small exponent. Việc chuyển sang RSASSA-PSS để kí, RSA-OAEP để mã hóa, áp dụng kích thước khóa lớn (≥ 3072bit), sử dụng HSM và library thực hiện constant-time sẽ giảm đáng kể rủi ro.

3. Kịch bản triển khai

1) Công cụ/Thư viện: OpenSSL, GMP, Docker, Python (pycryptodome)

2) Môi trường cô lập, chỉ chạy PoC trên hệ thống lab

3) Data collection & metrics

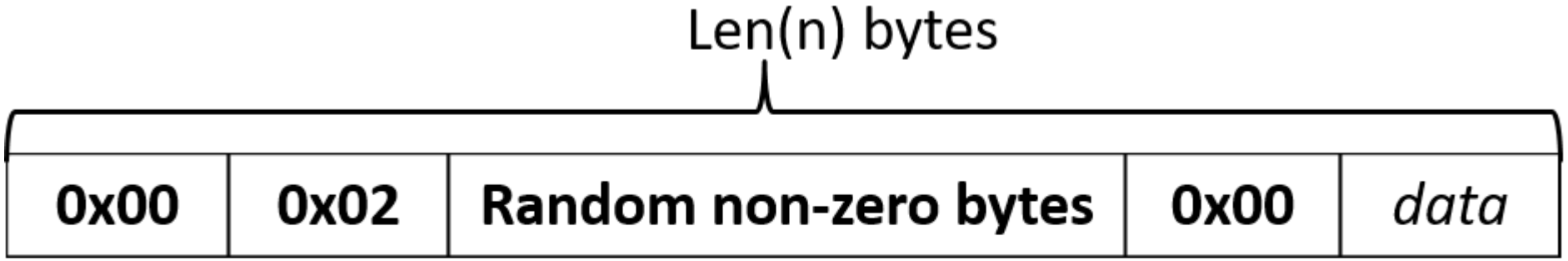
Các attack triển khai

- Attack A — Bleichenbacher padding oracle PoC: dựng server TLS-like dùng PKCS#1 v1.5 decryption cho PreMaster Secret, tạo oracle, phục hồi session secret; sau đó vá (uniform error, PSS) và kiểm chứng.
- Attack B — Timing attack: đo local và remote timing để dò rò rỉ thông tin về d; đánh giá countermeasures (exponent blinding, constant-time lib).
- Attack C — Wiener / low-d: sinh khóa vulnerable (small d) và dùng thuật toán Wiener để phục hồi private key

Attack A - Bleichenbacher padding oracle

- Mục tiêu
  - Đánh vào điểm yếu padding của hệ thống
  - Không cần private key, chỉ cần tồn tại oracle
- Điều kiện
  - Server sử dụng RSA key exchange + PKCS#1 v1.5
  - Server phản hồi khác nhau khi padding hợp và không hợp lệ
- Attacker được gửi ciphertext tùy ý liên tục và quan sát phản hồi

PKCS#1 v1.5 Padding



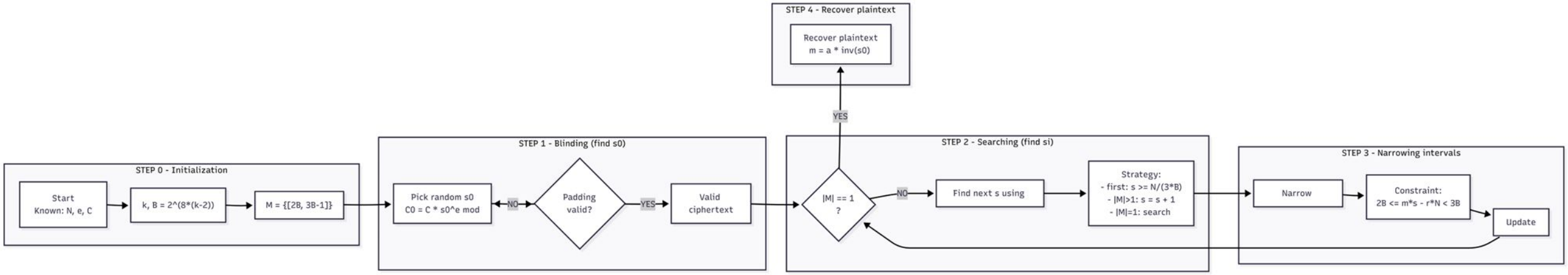
Padding used for RSA encryption

Padding used for RSA signature

Padding Oracle là gì?

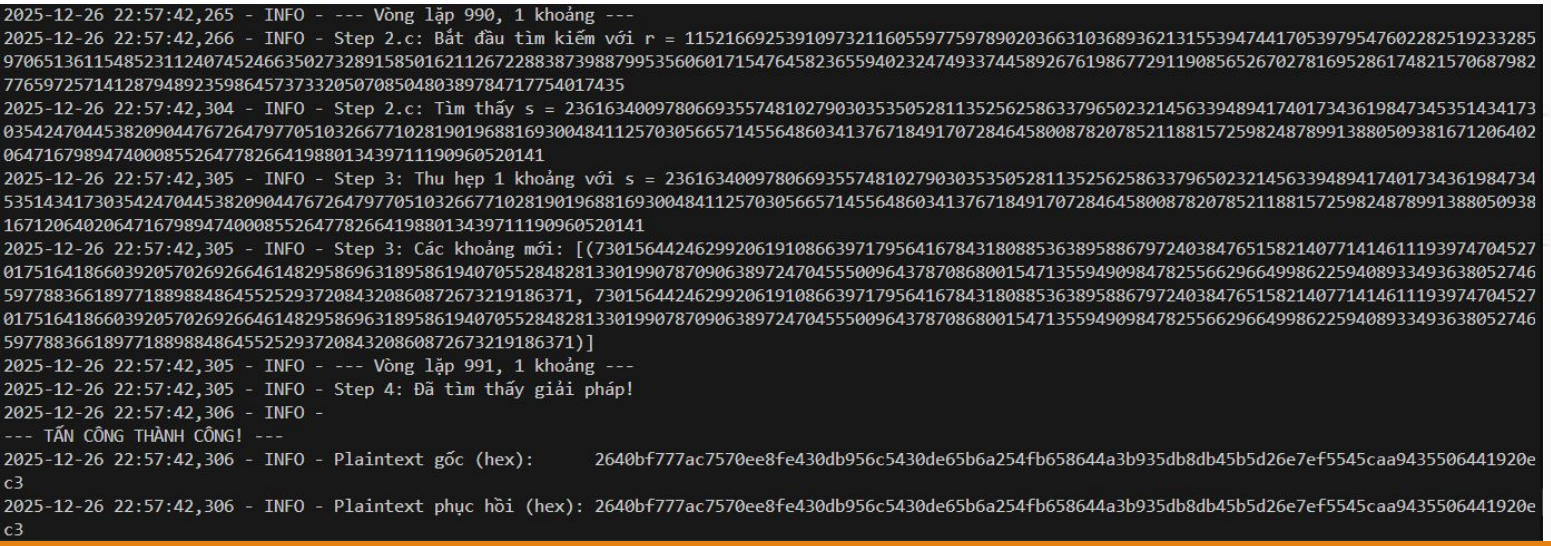
- Là một thành phần cho phép attacker biết **ciphertext sau khi decrypt có vượt qua kiểm tra padding hay không**, thông qua **phản hồi hoặc timing khác nhau**.
- Khi server thực hiện giải mã RSA hoặc kiểm tra padding PKCS#1 v1.5, server sẽ để lộ kết quả kiểm tra padding:
  - Trả về OK/FAIL
  - Timing khác nhau

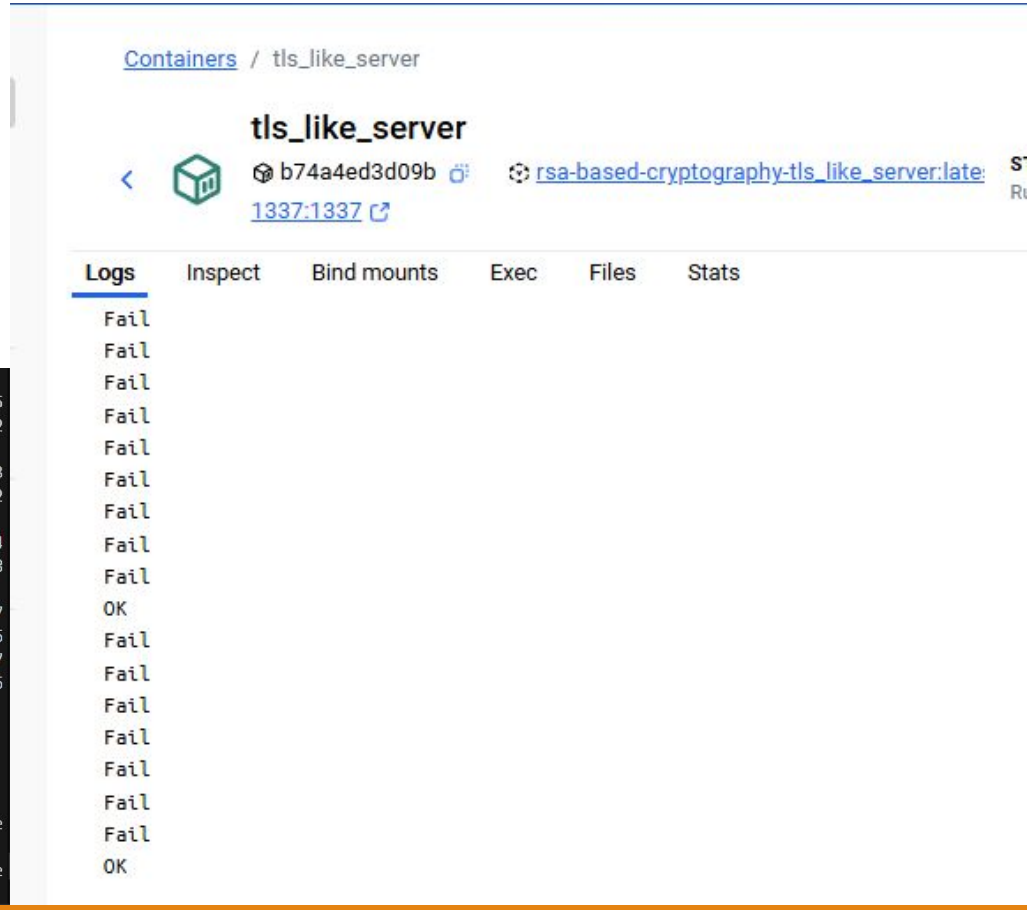
Attack A - Bleichenbacher padding oracle



Attack A - Bleichenbacher padding oracle

- Demo Lab:
  - Server dùng RSA PKCS#1 v1.5 để gen key và giải mã
  - Server trả về OK/FAIL
- Phục hồi 48 byte premaster secret (plaintext)





Kết quả - Attack A

Run	Queries	Time (s)	Avg / query (s)	QPS
1	183,346	360.38	0.001966	508.76
2	194,874	381.57	0.001958	510.72
3	298,513	576.78	0.001932	517.55
4	53,511	107.14	0.002002	499.44
5	46,068	101.14	0.002195	455.51
6	366,681	709.11	0.001934	517.10
7	582,094	1152.26	0.001980	505.18
8	405,583	851.15	0.002099	476.51
9	779,375	1722.48	0.002210	452.47
10	372,417	821.20	0.002205	453.50

Kết quả - Attack A

- Số oracle queries trung bình≈ 327,846 queries / run
- Thời gian tấn công trung bình≈ 678.72 giây ≈ 11.31 phút
- Thời gian trung bình / query≈ 0.002048 giây ≈ 2.05 ms
- Throughput (QPS) trung bình≈ 489.82 queries / second
- Avg time / query:Rất ổn định: ~1.93–2.21 ms -> chứng minh uniform timing path
- QPS: ~450–520 QPS

Mitigation - Attack A

- Vá bằng cách:
  - Thay PKCS#1 v1.5 thành OAEP
  - Đổi response của server thành cùng một kết quả (OK)
  - Thêm tính random cho thời gian trả về
- Sau khi vá, ta gửi các gói tin để kiểm thử

=> Các response từ server đều là OK và có thời gian trả lời gần nhau

```
[+] Target = tls_like_server_patched:1338
[+] Public key = poc/Bleichenbacher/public_patched.pem
[+] Modulus length k = 128 bytes
[+] Sending 200 valid OAEP ciphertexts and 200 invalid blobs...

== VALID OAEP ==
OK replies: 200/200
RTT ms: mean=1.903, median=1.884, p95=2.148, min=1.691, max=2.431

== INVALID RANDOM ==
OK replies: 200/200
RTT ms: mean=1.849, median=1.867, p95=2.008, min=1.556, max=2.180
```

Attack B – Timing Attack on RSA

- Loại tấn công: **Side-channel attack**
- Không phá vỡ giả định toán học của RSA
- Khai thác **rò rỉ thời gian thực thi** trong quá trình:
- RSA signing
- RSA decryption

Điều kiện

- Biết public key (N,e)
- Có quyền gửi nhiều message / ciphertext tuỳ ý
- Đo được thời gian xử lý T_i của mỗi lần ký / giải mã
- Mục tiêu:
  - Khôi phục private exponent d từng bit

Decryption/Signing

- RSA sẽ giải mã theo công thức:

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

- Khi số mũ d rất lớn, ta sẽ dùng các thuật toán để tính toán nhanh hơn

-> thuật toán phổ biến: Square and Multiply

Square and Multiply

- d sẽ được chuyển về hệ nhị phân, duyệt với mỗi bit d:
  - gặp 0 sẽ bình phương
  - gặp 1 sẽ bình phương sau đó nhân với cơ số
- cuối cùng chia có dư ở mỗi bước.

Lưu ý: gặp bit 1 sẽ có nhiều bước hơn => thời gian tính lâu hơn => oracle

Thu thập dữ liệu và thống kê

- Gửi nhiều ciphertext/message
- Mỗi lần gửi, ta sẽ đo thời gian thực thi của server(từ lúc gửi request đến khi server reply)
- Sau đó, lưu lại:
  - Message m
  - Thời gian thực thi (duration_ns)
  - Nhãn dự đoán: có / không Montgomery Multiplication

Với mỗi giả thuyết bit 0 hoặc 1:

  - Chia dữ liệu thành 2 nhóm timing
  - So sánh bằng thống kê (Welch’s t-test)

=> Bit có t-score cao hơn là bit đúng

Kết quả attack B

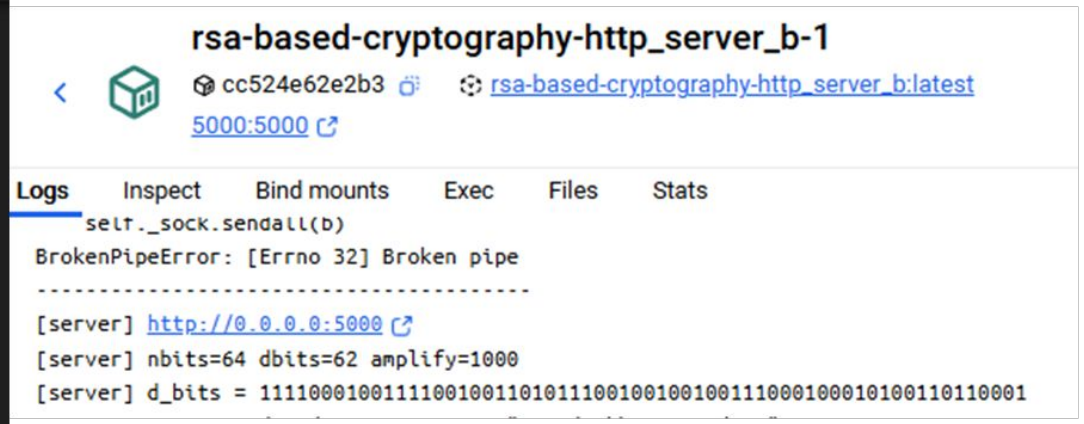
- Thử khôi phục 62 bits d =
111100010011110010011010111001001001001110001000101001101100 01

=> Thời gian: 1 tiếng

poc > TimingAttack > timing_log.csv > data

```
[bit   53] t0=-22.12 t1= 22.12 -> 1
[bit   54] t0=-20.48 t1= 20.48 -> 1
[bit   55] t0=  4.42 t1= -4.42 -> 0
[bit   56] t0=-20.39 t1= 20.39 -> 1
[bit   57] t0=-22.30 t1= 22.30 -> 1
[bit   58] t0= 10.39 t1=-10.39 -> 0
[bit   59] t0=  4.76 t1= -4.76 -> 0
[bit   60] t0=  1.28 t1= -1.28 -> 0
[bit   61] t0=-10.34 t1= 10.34 -> 1

Recovered d_bits:
111100010011110010011010111001001001001110001000101001101100 01
```



Mitigation - Attack B

Giải pháp hạn chế rủi ro:

- Constant-time implementation: bảo đảm thời gian chạy không phụ thuộc vào dữ liệu bí mật (bit của d, giá trị trung gian)--> thuật toán Montgomery Powering Ladder
- Blinding (message/exponent blinding): ngẫu nhiên hoá đầu vào hoặc số mũ khi tính toán private-key ops (RSA/ECC) để che tương quan giữa thời gian và secret;

Trả lời câu hỏi RQ2

RQ2: Các tấn công Bleichenbacher (padding oracle) và timing attacks còn khả thi trên các stack hiện đại (OpenSSL / LibreSSL / BoringSSL) khi gặp cấu hình cũ hoặc lỗi không?

-> Có, nhưng chỉ trong trường hợp triển khai sai:

- Hệ thống sử dụng RSA PKCS#1 v1.5 cho RSA Encryption
- Vẫn còn bật RSA key exchange (TLS ≤ 1.3)
- Có tồn tại padding/timing oracle

Attack C – Wiener's Attack / low d

- Khai thác cấu hình RSA **không an toàn** khi **private exponent d quá nhỏ**, cho phép:
  - Khôi phục private key
  - Chỉ cần khóa công khai (N,e)

=> Dẫn tới forgery chữ ký số và Broken Access Control

Điều kiện

Wiener chứng minh tấn công thành công nếu:

$$d < \frac{1}{3} N^{\frac{1}{4}}$$

với giả thiết RSA chuẩn: p, q có kích thước gần nhau.

- Thường thấy trong việc tự triển khai RSA
- Các phần cứng bị hạn chế, phải tối ưu

Ý tưởng

Quan sát chính

Trong RSA, ta có:

$$ed \equiv 1 \left(mod\varphi(N)\right) \Rightarrow ed - k\varphi(N) = 1$$

Chia hai vế cho $d\varphi(N)$:

$$\frac{e}{\varphi(N)} - \frac{k}{d} = \frac{1}{d\varphi(N)}$$

Vì $\varphi(N) \approx N$ (khi $p \approx q$), nên:

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

Điểm mấu chốt của Wiener, nếu d rất nhỏ:

$$d < \frac{1}{3}N^{\frac{1}{4}}$$

Thì $\frac{k}{d}$ sẽ xuất hiện như một convergent trong continued fraction của $\frac{e}{N}$

Ý tưởng

Cách tấn công

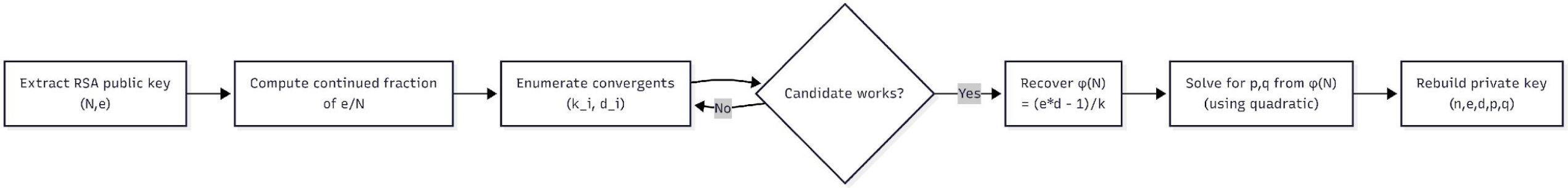
1. Tính từng continued fraction của $\frac{e}{N}$

2. Duyệt từng convergent của $\frac{k_i}{d_i}$

3. Với mỗi d_i, suy ra:

$$\varphi(N) = \frac{ed_i - 1}{k_i}$$

4. Từ $\varphi(N)$ giải ra p,q => dựng lại private key



Kết quả - Attack C

⇒ Tốn khoảng 0.02s để attack thành công

- Qua đó chứng minh rằng:

Giả thuyết small exponent misue là nguyên nhân compromise thực tế, có thể dẫn đến cơ chế xác thực không còn đáng tin

```
2025-12-26 09:09:49,556 - INFO -  Lab Vuln - Wiener's Attack on RSA with small d
2025-12-26 09:09:49,577 - INFO - publickey: -----BEGIN PUBLIC KEY-----
MIIBHzANBgkqhkiG9w0BAQEFAAOCAQwAMIIBBwKBgQC8rWJU6Ards7Btb1Soxj9s
UwvHbC1lneSeaZFVCD+PNXAh0n4dYOxBAvIUq2wdxtD68LlnDALfSoCllporb+lF
heKXH/yausIYH4qeC0pIEDsNUDXy6PwTtChyI3tsQTJWcZYyOs77oKpAJYCxUARX
7AUt55UqSYIMkbGl4oyFFQKBgD14XoBgtAx2xvR6JUgYuKe07QT2bf/Uqx6RTNx5
sD1k6s2Yh/iHMEJcf9jze7EeTDj7Saxtg4AHlEsP0EzmVZ2qyuElfkaqr4i9wIOV
xovE/Zb8waq7T0t3NBeJ1vhv5iK+uKiMvUInRG/KtKitwbt6PTQPKft8wVfzc1Fn
Wjzt
-----END PUBLIC KEY-----
2025-12-26 09:09:49,578 - INFO - [*] Parsed public key: n bitlen=1024, e=43165836764713451942510732090867132516270778103221544173148201893074711591629068210406143791
63954659745049131178084590126013923248172592991310010953226782510585100689305313406186560853219685810862329059365387457066526888372244924444076629229812193648737794 4
69129925424969578539216673688128041144483544735213
2025-12-26 09:09:49,578 - INFO - [*] Running Wiener's attack ...
2025-12-26 09:09:49,579 - INFO - [+] Wiener success: d=357624733010296676510751288710313847652645466873831484406427157934241086039 41
2025-12-26 09:09:49,593 - INFO - [+] Forged JWT:
eyJhbGciOiJSUzI1NiIsImtpZCI6ImxhYi12dWxuIiwidHlwIjoiSldUIn0.eyJleHAiOjE3NjY3NDM3ODksImlhdCI6MTc2Njc0MDE4OSwicm9sZSI6ImFkbWluIiwic3ViIjoiYXR0YWNrZXIifQ.sKelXYBeCnUIOZ
VoteJOG1-IzCiczY4nbNRXGKty97QikWqzxztqRX-sigVZWkHf6Yrk6p_zoSQGm6m_ofkd4fgwoBz9qX--1jTgTytMjNY0Yt7hcaYw_lXoSuLy7_rLZDyWow1AdU_bbfOBZmK_5AyFWLocpXy3pImqIzxKwq8
2025-12-26 09:09:49,594 - INFO - [+] Verified local signature OK
2025-12-26 09:09:49,594 - INFO - [*] Trying admin endpoint (via cookie): http://http_server_c:8000/admin
2025-12-26 09:09:49,598 - INFO - [*] /admin returned HTTP 200:
2025-12-26 09:09:49,598 - INFO - {
  "flag": "FLAG{weiner_attack_with_jwt}",
  "message": "Welcome, admin!",
  "ok": true,
  "payload": {
    "exp": 1766743789,
    "iat": 1766740189,
    "role": "admin",
    "sub": "attacker"
  },
  "role": "admin"
}
2025-12-26 09:09:49,598 - INFO - [*] Complete. Total elapsed: 0.021s
```

Mitigation - Attack C

- D bits trước khi vá:

localhost:8000/rsa.json

Pretty-print ☑


```
{
  "d": "342134740493215271975295280388356611226174324598270102389541360317384585349 63",
  "d_bits": 255,
```


- D bits sau khi vá:

localhost:8000/patched/rsa_patched.json

Pretty-print ☑


```
{
  "d":
"370249180121988379295391279123042119003903482790973076526655137832441135338250116074519530529318062140726037697726903997389638421583649756957998568700037016395433932177238169616110859815052859897531209780955007941258100623336318828509466886964413662924970232326303206391147139257336615702250838950187338898468627405730416792526065894612473803672980277400193456166508515416247003402225384838042164985580622407712379714755545921133112108062577519500987370450967171879461328345198881397645095257953750594120884671186120005541697218230943611341056426166607345936150935545265011915833255940215519281987707991816473064 73",
  "d_bits": 2042,
```

Hastad’s broadcast attack : Mở rộng

#) Kịch bản 1: Người dùng sử dụng khóa công khai nhỏ để gửi nhiều messages tới nhiều người

$\longrightarrow$ Thu thập đủ nhiều messages sau đó sử dụng định lý Thặng dư trung Hoa để khôi phục lại tin nhắn gốc

$$c_B \equiv m^3 \bmod N_B, c_C = m^3 \bmod N_c,\ c_D = m^3 \bmod N_D$$
$$\rightarrow\ m^3 = crt([c_B, c_C, c_D], [N_B, N_C, N_D])$$

Hastad’s broadcast attack: Mở rộng

#) Trong trường hợp số mũ e lớn hơn có thể sử dụng phương pháp chia để trị để tối ưu hóa thuật toán CRT

```python
def fast_crt(X, M, segment_size=8):
    assert len(X) == len(M)
    assert len(X) > 0
    while len(X) > 1:
        X_ = []
        M_ = []
        for i in range(0, len(X), segment_size):
            if i == len(X) - 1:
                X_.append(X[i])
                M_.append(M[i])
            else:
                X_.append(crt(X[i:i + segment_size], M[i:i + segment_size]))
                M_.append(lcm(*M[i:i + segment_size]))
        X = X_
        M = M_

    return X[0], M[0]
```

Hastad’s broadcast attack: Mở rộng

- Thời gian chạy trung bình của thuật toán sau khi tối ưu đối với một số trường hợp:

size of N (bits)	e	running time
2048	3	2.00s
2048	37	12.05s
2048	53	24.4s

Hastad’s broadcast attack: Mở rộng

#) Kịch bản 2: Người dùng sử dụng khóa công khai nhỏ để gửi nhiều messages tới nhiều người

⟶ Lúc này giữa các message được padding theo cách khác nhau.

⟶ Tuy vậy, nếu như thuật toán padding vô tình tạo ra quan hệ tuyến tính giữa các message thì ta vẫn có thể tìm cách khôi phục lại được message ban đầu

Hastad’s broadcast attack: Mở rộng

#) Triển khai:

- Ta có hai message m1 và m2, ở đây ta giả sử hai message có quan hệ tuyến tính với nhau tức là m2 = f(m1) với f = ax+b
- Ta có hai ciphertext tương ứng với hai message này là c1 và c2.
- ⟶ Dựng hai đa thức g(x) = x^e - c1 và h(x)= (ax+b)^e-c2 trên vành đa thức Z/N[x] với N là modulus được sử dụng để mã hóa
- Tối ưu: Có hai hướng tối ưu chính
  - Sử dụng thuật toán Half GCD để tăng tốc tính toán GCD của hai đa thức, trích dẫn: faculty.sites.iastate.edu/jia/files/inline-files/polydivide.pdf
  - Sử dụng phần mềm đại số Sagemath để tính toán đa thức

Hastad’s broadcast attack: Mở rộng

- Thời gian chạy trung bình cho một số trường hợp:

size of N (bits)	e	running time
2048	3	0.0091s
2048	37	0.0144s
2048	53	0.0160s
2048	46769	89.4209s
2048	65537	100.4038s

Hastad’s broadcast attack

#) Khắc phục

- Sử dụng padding ngẫu nhiên an toàn khi mã hóa RSA, chẳng hạn RSA-OAEP, nhằm đảm bảo mỗi lần mã hóa cùng một plaintext cho ra ciphertext khác nhau.
- Áp dụng mô hình hybrid encryption (KEM–DEM): RSA chỉ dùng để mã hóa khóa phiên (KEM), trong khi nội dung tin nhắn được mã hóa bằng các thuật toán mã hóa đối xứng an toàn như AES-GCM hoặc ChaCha20-Poly1305 (DEM).

Fault attack: mở rộng

- Khi triển khai RSA, định lý thặng dư Trung Hoa được dung để tăng tốc độ kí và giải mã (nhanh hơn khoảng 4 lần)

- Quy trình: Thay vì tính dư trên N, hệ thống tính trên hai nhánh:

$$\sigma_p = \mu(m)^{d \bmod p-1} \bmod p,\ \sigma_q = \mu(m)^{d \bmod q-1} \bmod q$$

- Sau đó, gộp lại bằng CRT để ra chữ kí σ.

Bellcore Attack

Giả định: attacker có thể tạo ra lỗi khi tính toán σ_q, trong khi đó server lại tính toán σ_p đúng:

$$\sigma_p = \mu(m)^{d \bmod p-1} \bmod p, \sigma_q \neq \mu(m)^{d \bmod q-1} \bmod q$$

Từ đó σ khi gộp lại sẽ chia hết cho p nhưng không chia hết cho q:

$$\gcd(\sigma^e - \mu(m) \bmod N, N) = p$$

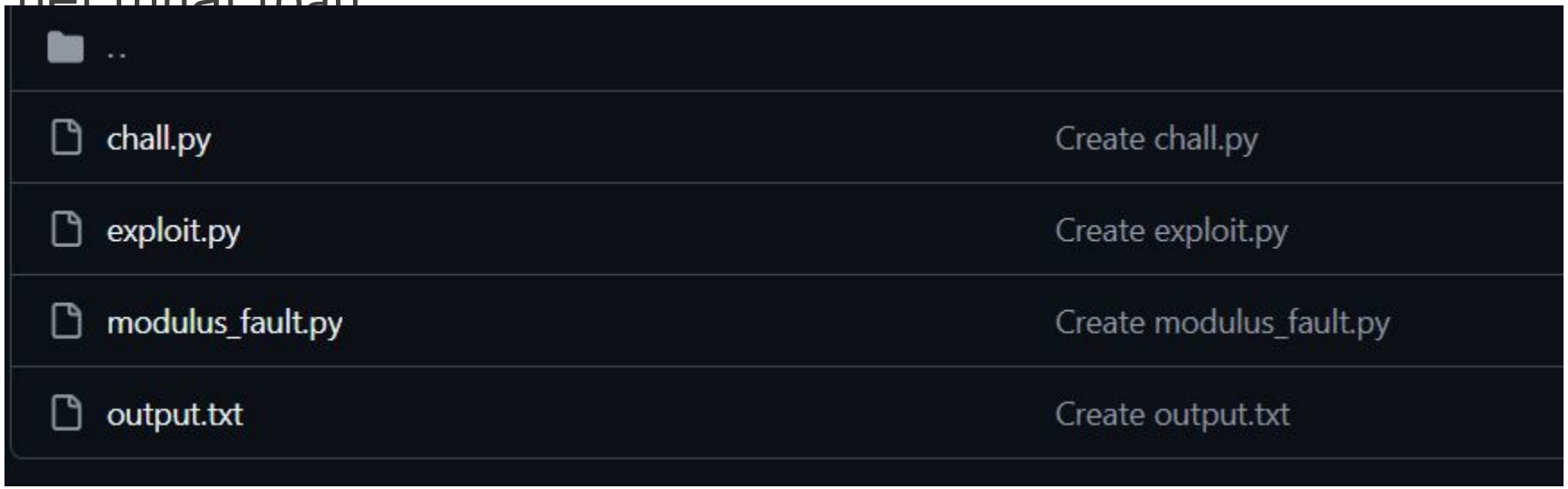
Modulus Fault

- Một trường hợp khác có thể xảy ra đó là việc tính chữ kí bị lỗi ở bước lấy modulo.
- Bằng cách thu thập các chữ ký lỗi, ta có thể tạo ra được một tổ hợp tuyến tính của các thành phần trong bước tính CRT
-

$$\sigma' = \sigma_p \cdot \alpha + \sigma_q \cdot \beta \bmod N', \; N' \neq N$$
$$\rightarrow \; v = \sigma_p \cdot \alpha + \sigma_q \cdot \beta \bmod N \cdot N'$$
$$\rightarrow v = \sigma_p \cdot \alpha + \sigma_q \cdot \beta$$

Modulus Fault

- Hướng tấn công:
  - Thu thập nhiều chữ kí lỗi và sử dụng thuật toán LLL để phân tích modulus
- Chi tiết thuật toán:



Modulus Fault

- Số lượng chữ kí cần để phân tích N:

Number of faulty signatures ℓ	4	5	6
1024-bit moduli	48%	100%	100%
1536-bit moduli	45%	100%	100%
2048-bit moduli	46%	100%	100%

Modulus Fault

Biện pháp hạn chế rủi ro:

- Sử dụng công thức Garner's formula thay cho thuật toán CRT → không bị phụ thuộc vào N

$$
\begin{aligned}
&t \leftarrow \sigma_p - \sigma_q \\
&\text{if } t < 0 \text{ then } t \leftarrow t + p \\
&\sigma \leftarrow \sigma_q + (t \cdot \gamma \bmod p) \cdot q \\
&\text{return}(\sigma)
\end{aligned}
$$

- Sử dụng các scheme chữ ký có tính ngẫu nhiên chẳng hạn RSA - PSS, hoặc stateful scheme (mỗi lần ký sẽ cập nhật trạng thái, kết quả sẽ khác nhau)

4. Kiến trúc giải pháp

1) Key size phải đạt chuẩn: ít nhất 2048 bit, khuyến khích tăng lên 3072 bit hoặc cao hơn nếu muốn bảo mật dài hạn.

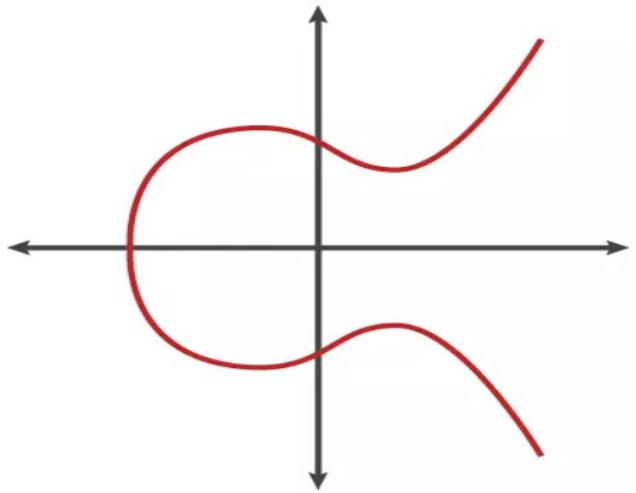
2) Đảm bảo tính ngẫu nhiên khi sinh khóa.

3) Tránh dùng chung 1 khóa cho nhiều dịch vụ khác nhau.

4) Chuẩn bị các giải pháp bảo vệ dài hạn:
Giám sát các kĩ thuật phá mã và điều chỉnh độ dài khóa. Lên kế hoạch chuyển sang các thuật toán mới, đặc biệt là mật mã hậu lượng tử.

5. Mở rộng hướng đi tiếp theo

- Chuyển sang sử dụng các hệ mật mã hậu lượng tử (PQC, Lattice based, isogeny based), ECC, v.v..
- ECC - Hệ mật mã dựa trên đường cong, có một số ưu điểm nhất định so với RSA
  - TLS/Key agreement: ECDHE
  - Chữ ký số/Cert: RSA-PSS → ECDSA hoặc EdDSA



RSA vs ECC

Điểm hạn chế của RSA:

- Thuật toán sinh khóa của RSA rất chậm, đối với khóa có độ lớn 16384 bit có thể mất đến vài phút trên máy tính cá nhân.
- Kích thước chữ kí lớn, độ lớn bằng với modulo của khóa → tốn nhiều tài nguyên khi lưu trữ trên các hệ thống.

  Ví dụ → https://github.com/phpseclib/phpseclib/issues/963

RSA vs ECC

1) Một số ưu điểm của ECC

a) Kích thước khóa , chữ kí và bản mã nhỏ: Để đạt 128-bit security thì cần một khóa EC có độ lớn 256-bits, trong khi RSA sẽ cần khóa 3072 bits.

b) Fast key generation: Đối với hệ mật ECC, thuật toán sinh khóa sẽ dựa vào hai thuật toán cơ sở là PRNG và EC point multiplication → nhanh hơn so với RSA

c) Fast signing, key agreement and encryption/decryption

RSA vs ECC

2) Nhược điểm của ECC

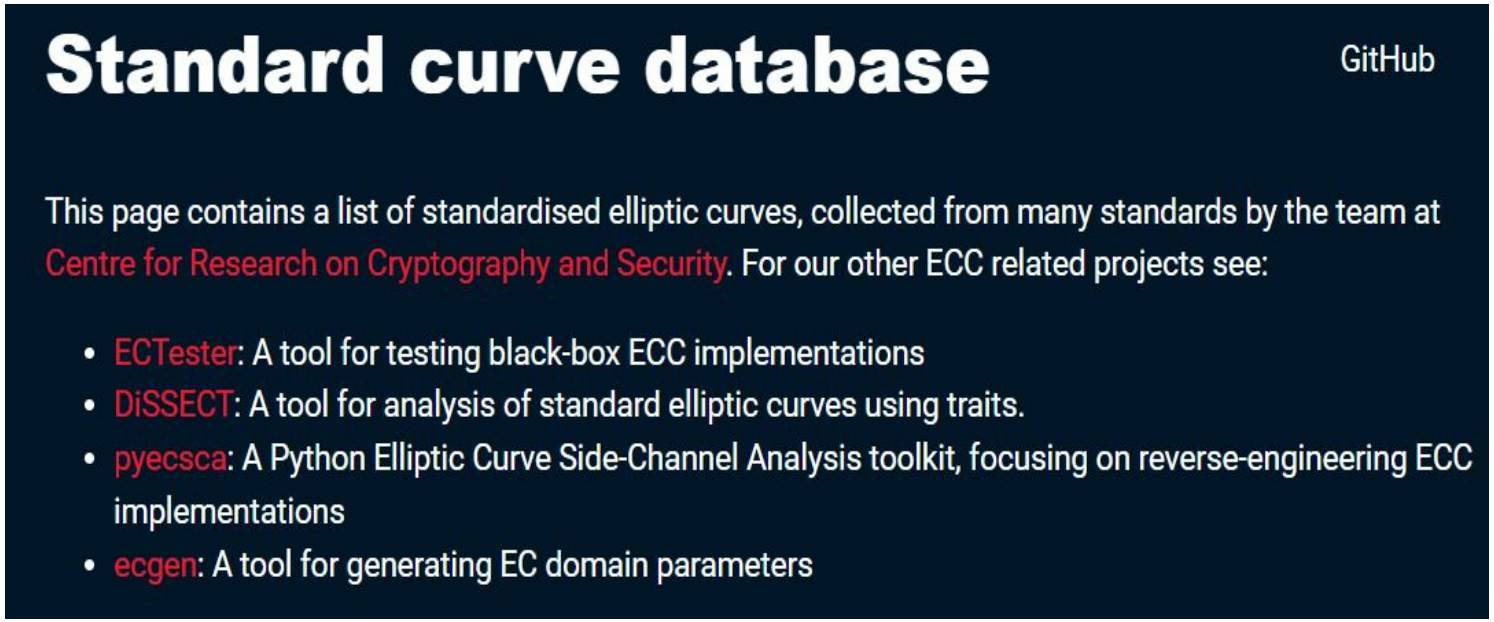
a) Thuật toán ECC có yêu cầu cài đặt an toàn phức tạp hơn RSA → tuy nhiên ta có thể xài các thư viện đã được audit như OpenSSL, BoringSSL, v.v…
b) Lỗi ở thuật toán sinh số ngẫu nhiên PRNG có thể khiến cho chữ kí số bị giả mạo (ECDSA Nonce)
c) Không phải tất cả các curve đều an toàn, chỉ nên sử dụng những curve đã được kiểm chứng và phổ biến rộng rãi

RSA vs ECC

name	curve equation over $\mathbb{F}_p$	prime p	source
NIST P-256	$y^2 = x^3 - 3x + b$; "large" b	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	[220]
brainpoolP256t1	$y^2 = x^3 - 3x + b$; "large" b	"dense"	[66]
Curve25519	$y^2 = x^3 + 486662x^2 + x$	$2^{255} - 19$	[23]

Nguồn: https://eprint.iacr.org/2024/1265.pdf

RSA vs ECC



Nguồn: https://std.neuromancer.sk/

RSA vs ECC

Table 1: Different key sizes for equivalent security levels

security level (bits)	80	112	128	192	256
RSA modulus n (modulus)	1024	2048	3072	8192	15360
DL parameter q (order)	160	224	256	384	512
EC parameter n (order)	160	224	256	384	512

So sánh security level dựa trên key size giữa hai thuật toán

RSA vs ECC

Thuật toán (worst-case theo script)	Thời gian chạy (giây)
RSA modular exponentiation `pow(m, n-2, n)`	0.0189222010
ECC scalar multiplication over (F_q)	0.0018611840
ECC scalar multiplication over (F_{q^2})	0.7118690460
Weil pairing over (F_q)	0.0035008340
Weil pairing over (F_{q^2})	1.5980586600

So sánh thời gian chạy của một số thuật toán

RSA vs ECC

Triển khai mẫu: CA Certificate cho HTTP Server

```
sudo openssl genpkey -algorithm EC \
  -pkeyopt ec_paramgen_curve:secp384r1 \
  -pkeyopt ec_param_enc:named_curve \
  -out ca.key
sudo openssl req -x509 -new -key ca.key -sha256 -days 3650 \
  -subj "/C=VN/ST=HCM/L=HCM/O=DemoLab/OU=CA/CN=DemoLab Root CA" \
  -out ca.crt
sudo openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial \
  -out server.crt -days 825 -sha256 \
  -extfile server.cnf -extensions req_ext
```

RSA vs ECC

Demo triển khai các thuật toán RSA bao gồm chữ kí số, mã hóa và giải mã trên hai thư viện OpenSSL và Pycryptodome.

Kết luận

- RSA vẫn an toàn nếu triển khai chuẩn, đầy đủ, thuật toán RSA được đánh giá là đơn giản và cho thời gian mã hóa nhanh (số mũ e = 65537 nhỏ)
- Ưu tiên RSASSA-PSS cho chữ ký và tránh PKCS#1 v1.5 khi có thể.
- Dùng kích thước khóa lớn (≥3072 bit) hoặc chuyển sang ECC/PQC cho bảo vệ dài hạn.
- Vô hiệu hóa RSA key exchange trong TLS, ưu tiên ECDHE cho forward secrecy.
- Áp dụng exponent & CRT blinding, constant-time modular exponentiation, và CSPRNG mạnh cho keygen.
- Lưu private key kĩ, giới hạn API và dùng uniform error messages.

Cám ơn thầy và các bạn đã theo dõi