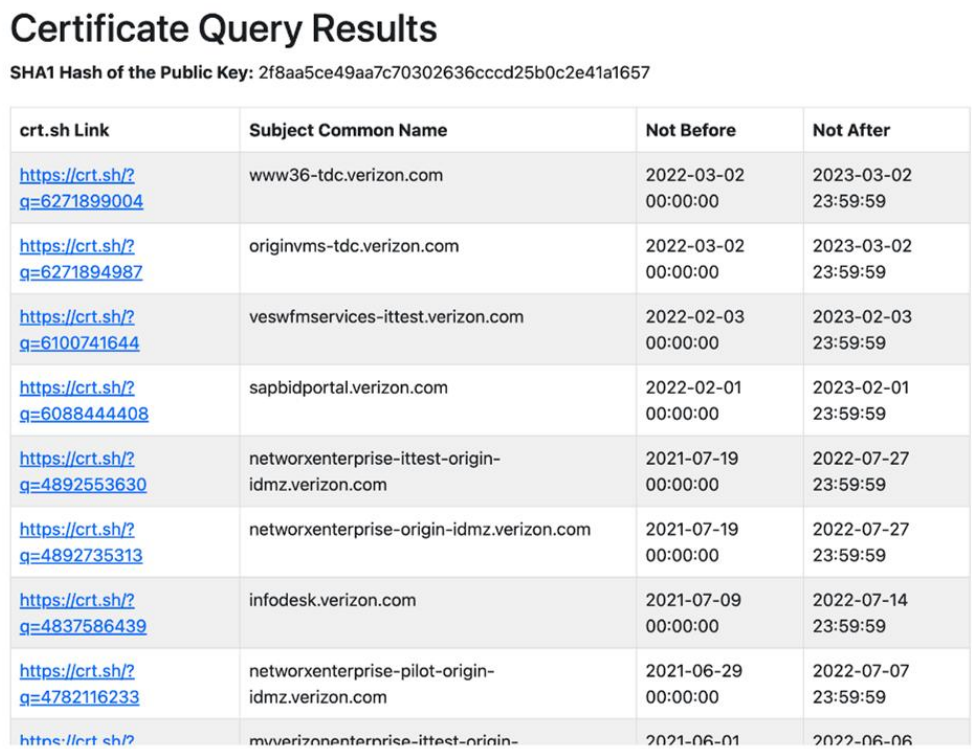
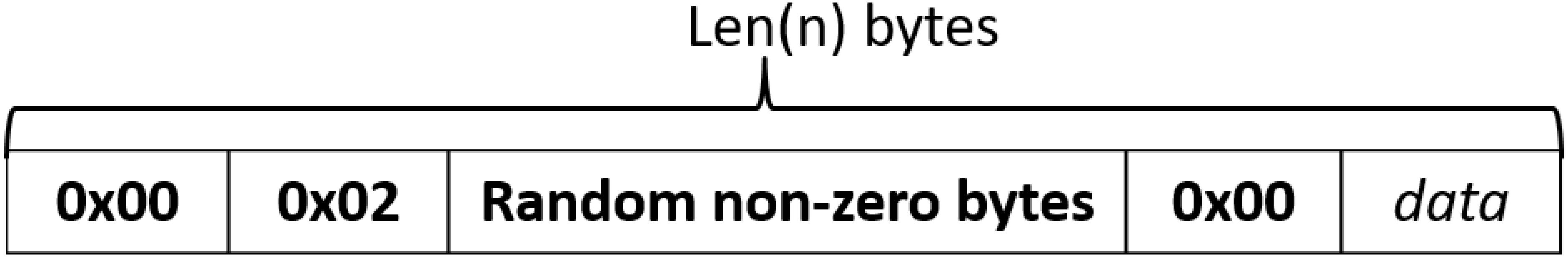
Cryptanalysis on Asymmetric Ciphers — RSA & RSA-Based Signatures



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



Len(n) bytes

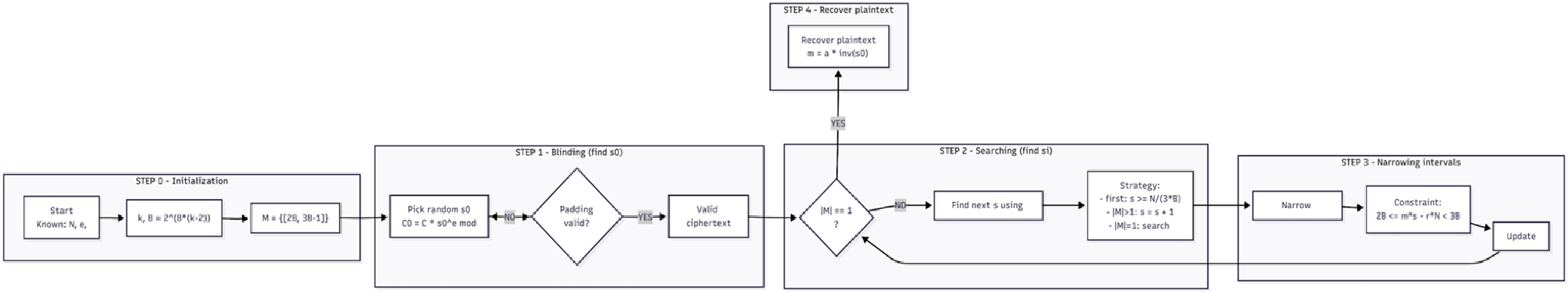
0x00	**0x02**	**Random non-zero bytes**	**0x00**	*data*

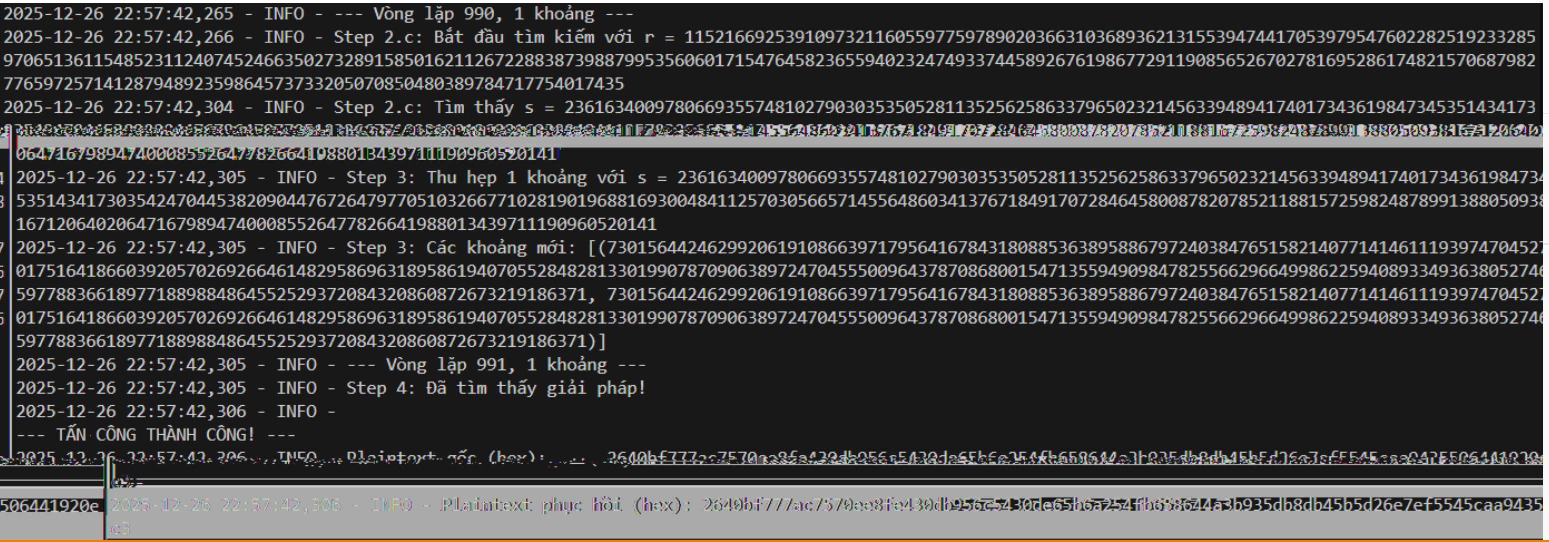
Padding used for RSA encryption

0x00	**0x01**	**0xFF…FF**	**0x00**	*data*

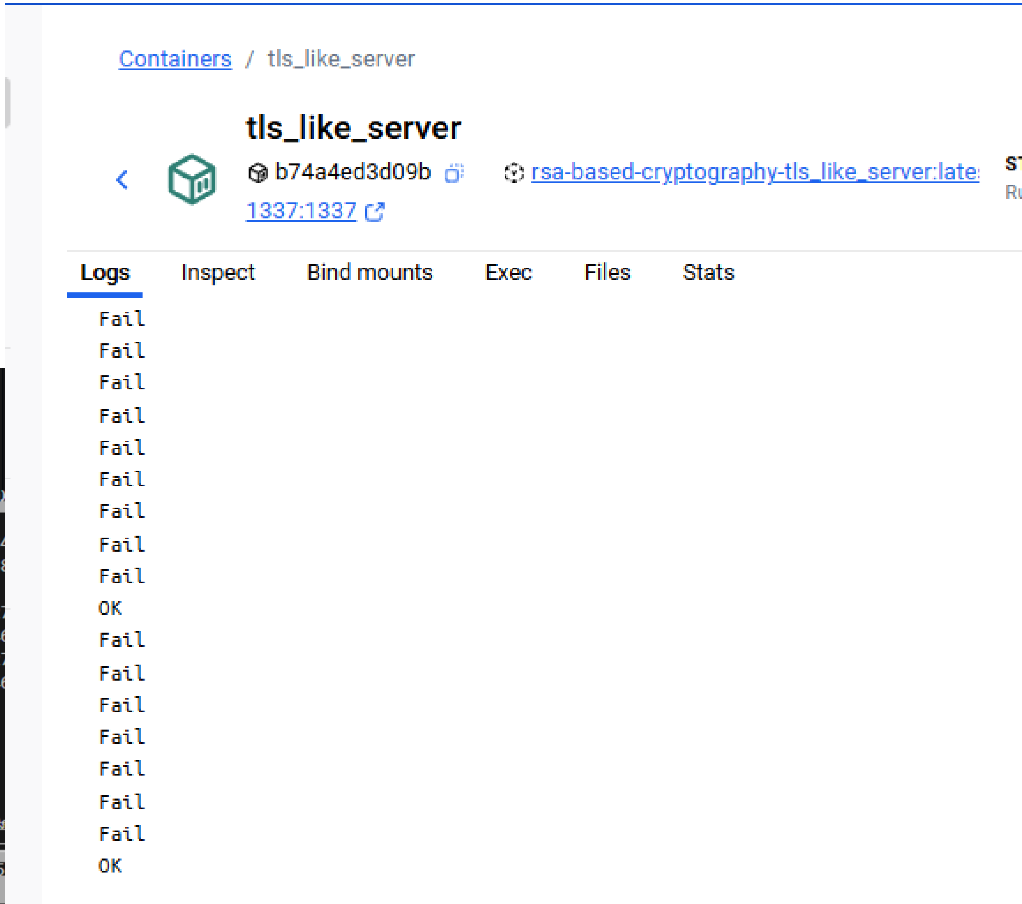
Padding used for RSA signature

ciphertext sau khi decrypt có vượt qua kiểm tra padding hay không phản hồi hoặc timing khác nhau


STEP 0 - Initialization
Start
Known: N, e,
k, B = 2^(8*(k-2))
M = {[2B, 3B-1]}
STEP 1 - Blinding (find s0)
Pick random s0
C0 = C * s0^e mod
NO
Padding valid?
YES
Valid ciphertext
STEP 2 - Searching (find si)
|M| == 1 ?
NO
Find next s using
Strategy:
- first: s >= N/(3*B)
- |M|>1: s = s + 1
- |M|=1: search
YES
STEP 4 - Recover plaintext
Recover plaintext
m = a * inv(s0)
STEP 3 - Narrowing intervals
Narrow
Constraint:
2B <= m*s - r*N < 3B
Update



```
2025-12-26 22:57:42,265 - INFO - --- Vòng lặp 990, 1 khoảng ---
2025-12-26 22:57:42,266 - INFO - Step 2.c: Bắt đầu tìm kiếm với r = 115216692539109732116055977597890203663103689362131553947441705397954760228251923328
970651361154852311240745246635027328915850162112672288387398879953560601715476458236559402324749337445892676198677291190856526702781695286174821570687982
77659725714128794892359864573733205070850480389784717754017435
2025-12-26 22:57:42,304 - INFO - Step 2.c: Tìm thấy s = 2361634009780669355748102790303535052811352562586337965023214563394894174017343619847345351434173
[illegible]
064716798947400085526477826641988013439711190960520141
2025-12-26 22:57:42,305 - INFO - Step 3: Thu hẹp 1 khoảng với s = 236163400978066935574810279030353505281135256258633796502321456339489417401734361984734
53514341730354247044538209044767264797705103266771028190196881693004841125703056657145564860341376718491707284645800878207852118815725982487899138805093
1671206402064716798947400085526477826641988013439711190960520141
2025-12-26 22:57:42,305 - INFO - Step 3: Các khoảng mới: [(730156442462992061910866397179564167843180885363895886797240384765158214077141461119397470452
01751641866039205702692664614829586963189586194070552848281330199078709063897247045550096437870868001547135594909847825566296649986225940893349363805274
597788366189771889884864552529372084320860872673219186371, 730156442462992061910866397179564167843180885363895886797240384765158214077141461119397470452
01751641866039205702692664614829586963189586194070552848281330199078709063897247045550096437870868001547135594909847825566296649986225940893349363805274
597788366189771889884864552529372084320860872673219186371)]
2025-12-26 22:57:42,305 - INFO - --- Vòng lặp 991, 1 khoảng ---
2025-12-26 22:57:42,305 - INFO - Step 4: Đã tìm thấy giải pháp!
2025-12-26 22:57:42,306 - INFO -
--- TẤN CÔNG THÀNH CÔNG! ---
2025-12-26 22:57:42,306 - INFO - Plaintext gốc (hex): 2640bf777ac7570ee8fe430db956c5430de65b6e254fb658644a3b935db8db45b5d26e7ef5545caa9425506441920
506441920e 2025-12-26 22:57:42,306 - INFO - Plaintext phục hồi (hex): 2640bf777ac7570ee8fe430db956c5430de65b6a254fb658644a3b935db8db45b5d26e7ef5545caa9435
c3
```



Containers / tls_like_server

tls_like_server

b74a4ed3d09b rsa-based-cryptography-tls_like_server:late

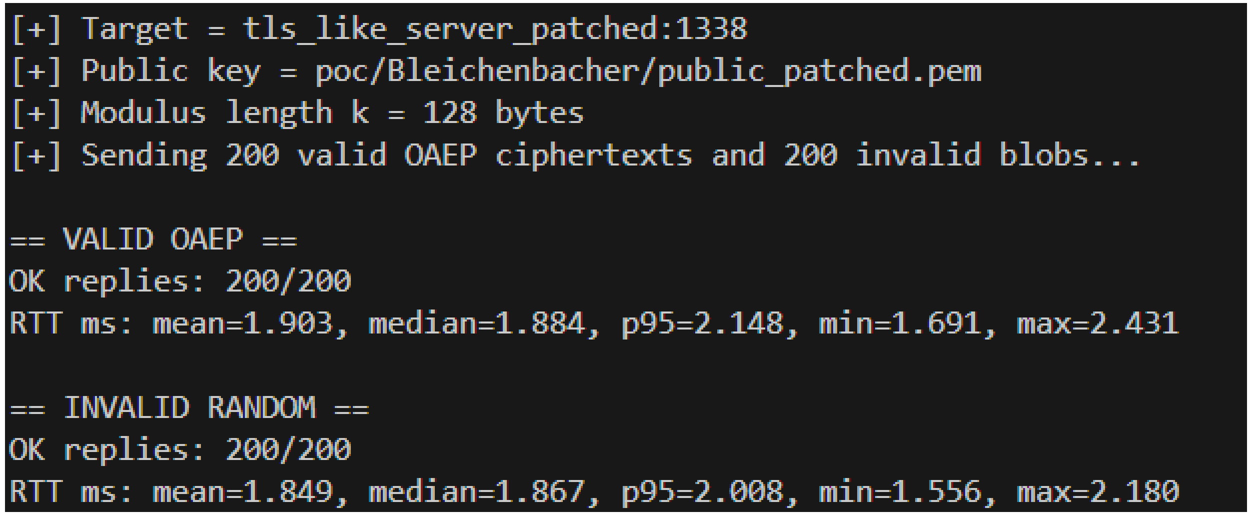
1337:1337

```
Fail
Fail
Fail
Fail
Fail
Fail
Fail
Fail
Fail
OK
Fail
Fail
Fail
Fail
Fail
Fail
Fail
OK
```

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

- 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
```

Side-channel attack

rò rỉ thời gian thực thi

$N\ e$

T_i

d

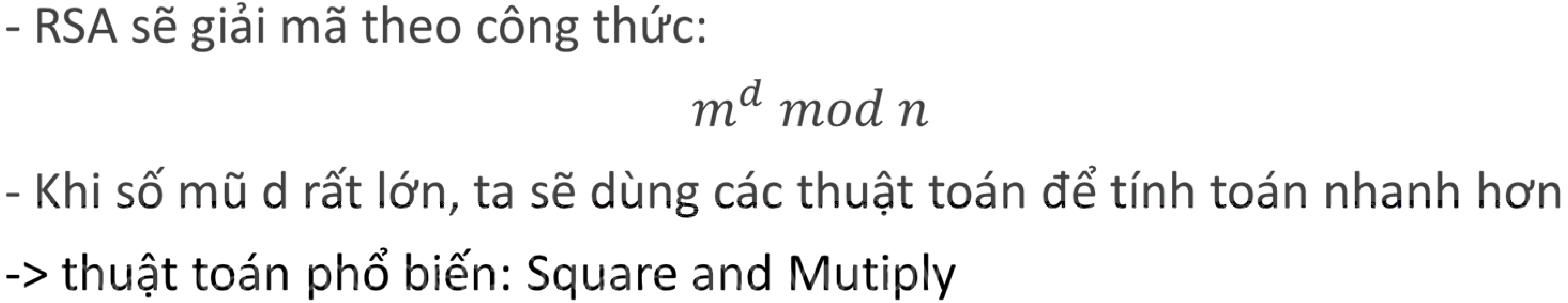
Điều kiện

$N \; e$

T_i

d

Decryption/Signing



- RSA sẽ giải mã theo công thức:

$$m^d \; mod \; n$$

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

-> thuật toán phổ biến: Square and Mutiply

Square and Multiply

- d sẽ được chuyển về cơ số 2, 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:

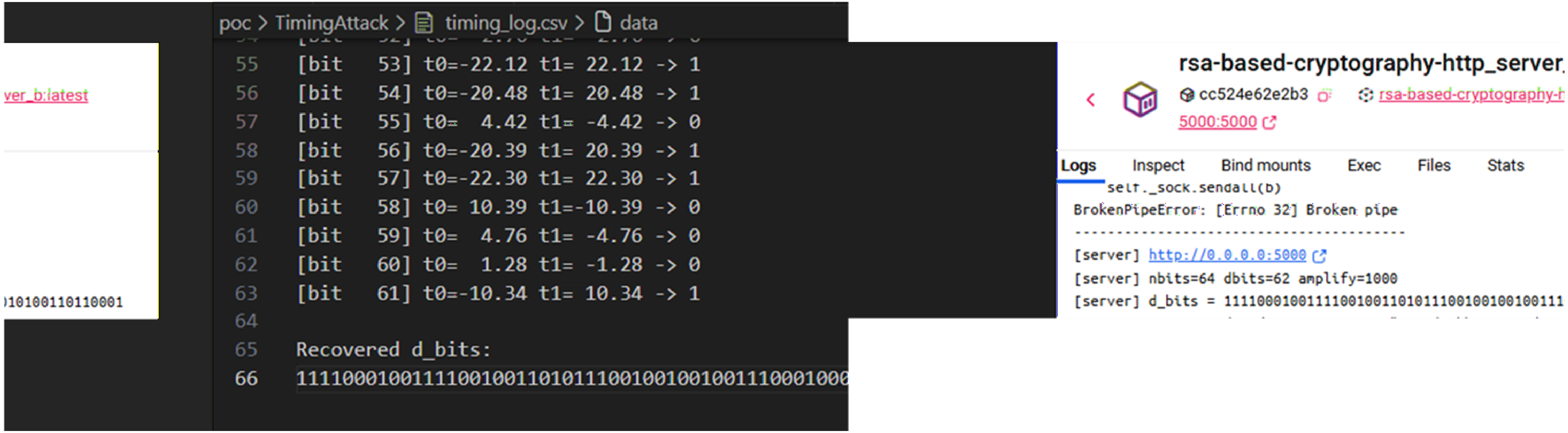
  - Chia dữ liệu thành 2 nhóm timing
  - So sánh bằng thống kê (Welch’s t-test)

=> Bit có t-score cao hơn là bit đúng

Kết quả attack B

- Thử khôi phục 62 bits d = 11110001001111001001101011100100100100111000100010100110110001

=> Thời gian: 1 tiếng



```
[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:
1111000100111100100110101110010010010011100010000
```

```
    self._sock.sendall(b)
BrokenPipeError: [Errno 32] Broken pipe
----------------------------------------
[server] http://0.0.0.0:5000
[server] nbits=64 dbits=62 amplify=1000
[server] d_bits = 1111000100111100100110101110010010010011
```

:

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?

không an toàn private exponent d quá nhỏ

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

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:

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

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

Cách tấn công

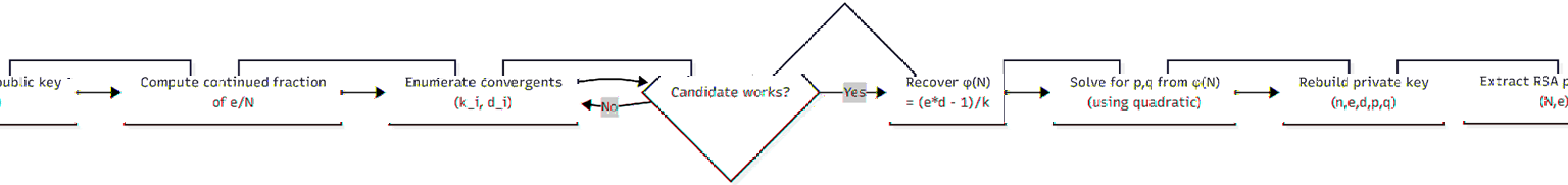
1. Tính từng continued fraction của $\frac{e}{N}$

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



ublic key

Compute continued fraction of e/N

Enumerate convergents (k_i, d_i)

Candidate works?

Yes

No

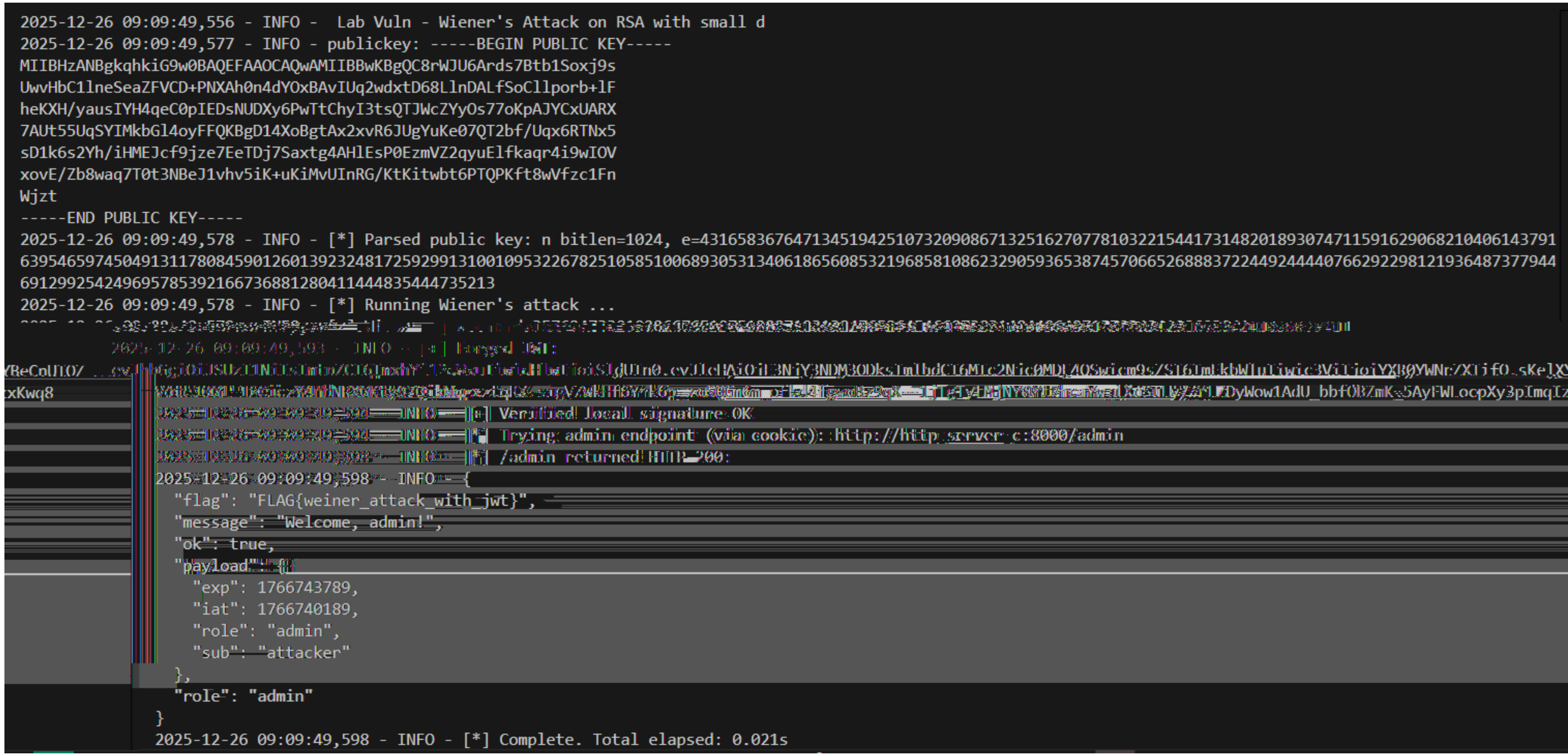
Recover φ(N) = (e*d - 1)/k

Solve for p,q from φ(N) (using quadratic)

Rebuild private key (n,e,d,p,q)

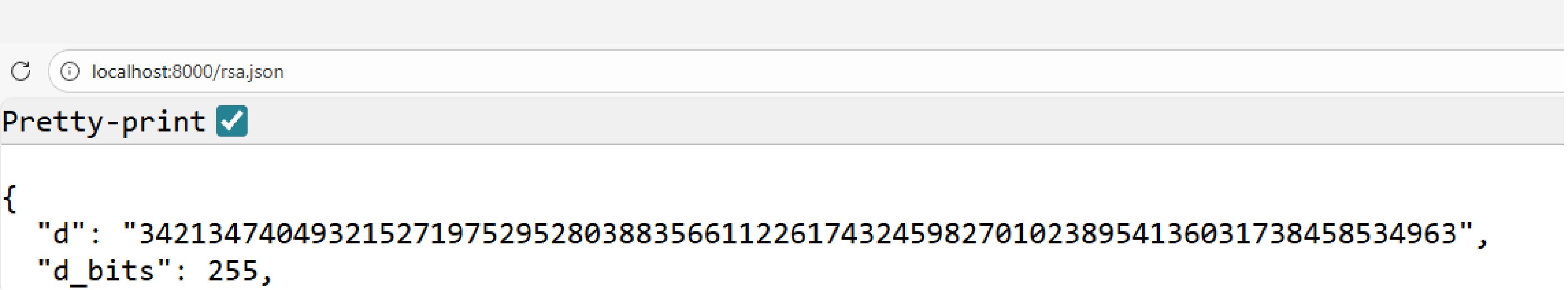
Extract RSA p (N,e)

⇒



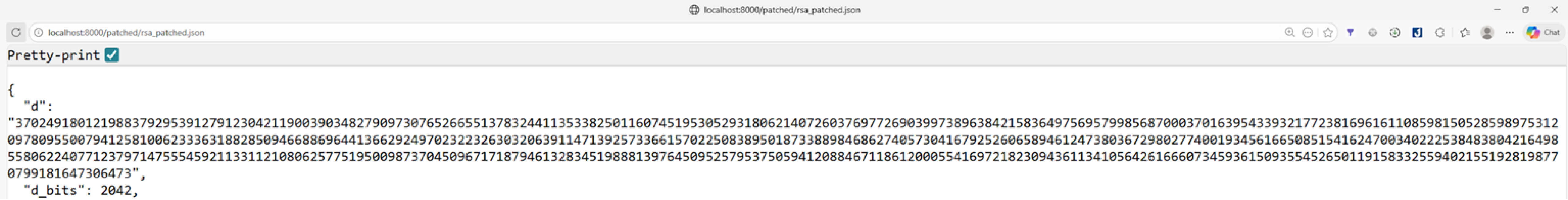
```
2025-12-26 09:09:49,556 - INFO -  Lab Vuln - Wiener's Attack on RSA with small d
2025-12-26 09:09:49,577 - INFO - publickey: -----BEGIN PUBLIC KEY-----
MIIBHzANBgkqhkiG9w0BAQEFAAOCAQwAMIIBBwKBgQC8rWJU6Ards7Btb1Soxj9s
UwvHbC1lneSeaZFVCD+PNXAh0n4dYOxBAvIUq2wdxtD68LlnDALfSoCllporb+lF
heKXH/yausIYH4qeC0pIEDsNUDXy6PwTtChyI3tsQTJWcZYyOs77oKpAJYCxUARX
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=4316583676471345194251073209086713251627077810322154417314820189307471159162906821040614379
6395465974504913117808459012601392324817259299131001095322678251058510068930531340618656085321968581086232905936538745706652688837224492444407662922981219364873779440
691299254249695785392166736881280411444835444735213
2025-12-26 09:09:49,578 - INFO - [*] Running Wiener's attack ...
[illegible]
        2025-12-26 09:09:49,593 - INFO - [*] Forged JWT:
YBeCpU10/ [illegible]
xKwq8 [illegible]
        [illegible] - INFO - [*] Verified local signature OK
        [illegible] - INFO - [*] Trying admin endpoint (via cookie): http://http_server_c:8000/admin
        [illegible] - 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
```

- D bits trước khi vá:



```
{
  "d": "34213474049321527197529528038835661122617432459827010238954136031738458534963",
  "d_bits": 255,
```

- D bits sau khi vá:



```
{
  "d":
"370249180121988379295391279123042119003903482790973076526655137832441135338250116074519530529318062140726037697726903997389638421583649756957998568700037016395433932177238169616110859815052859897531 2
0978095500794125810062333631882850946688696441366292497023223263032063911471392573366157022508389501873388984686274057304167925260658946124738036729802774001934561665085154162470034022253848380421 6498
558062240771237971475554592113311210806257751950098737045096717187946132834519888139764509525795375059412088467118612000554169721823094361134105642616660734593615093554526501191583325594021551928198 77
0799181647306473",
  "d_bits": 2042,
```

Fault attack: mở rộng

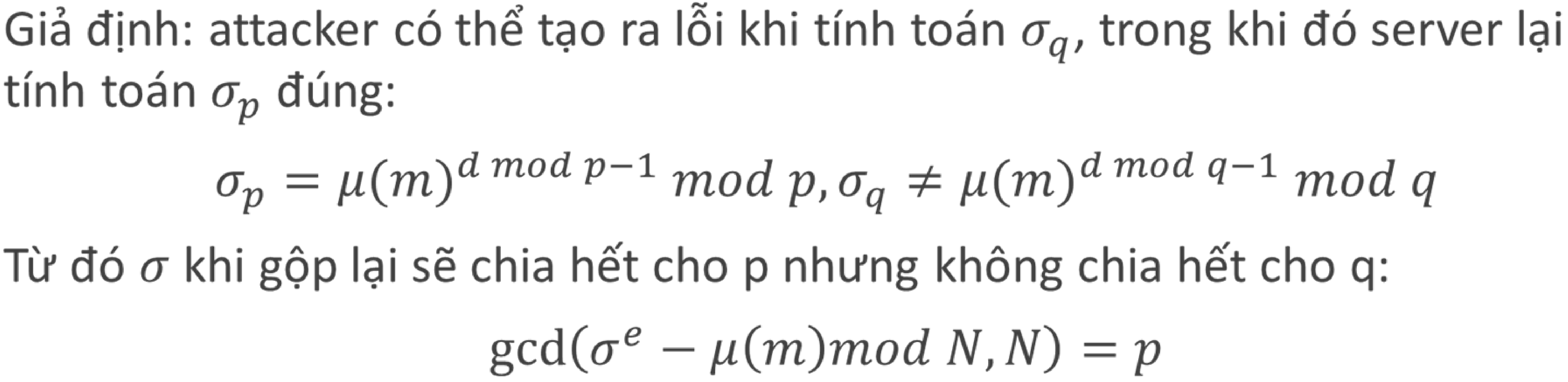
- Khi triển khai RSA, định lý thặng dư Trung Hoa được dung để tăng tốc độ kí và giải mã (nhanh hơn khoảng 4 lần)

- 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:

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

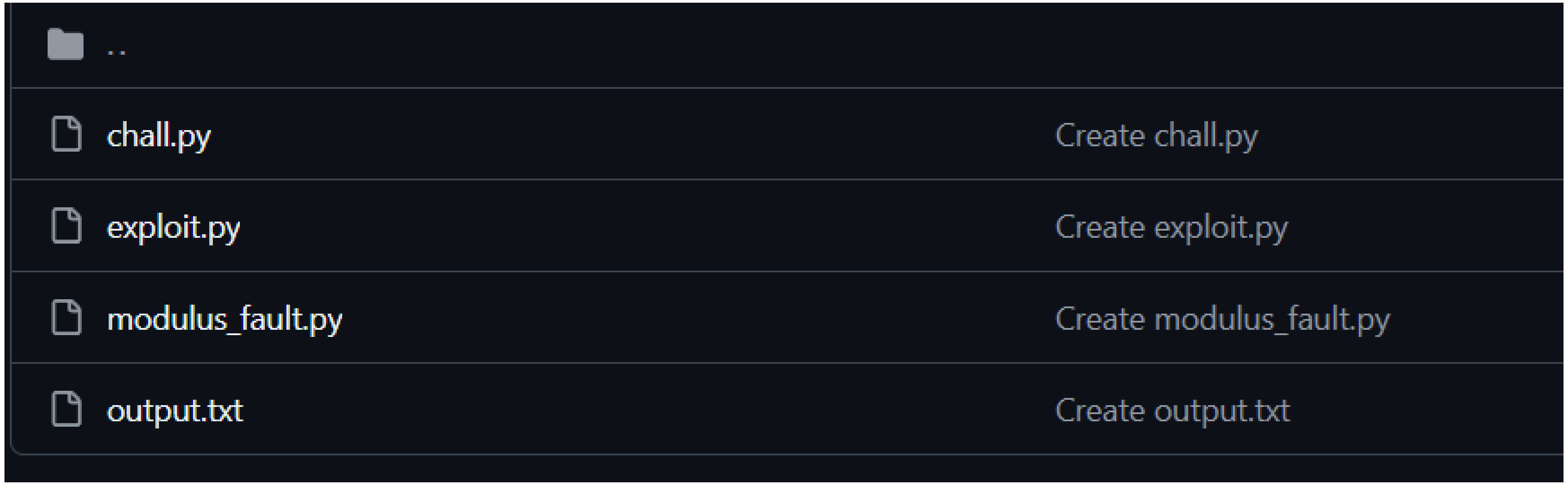
Từ đó σ khi gộp lại sẽ chia hết cho p nhưng không chia hết cho q:

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

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

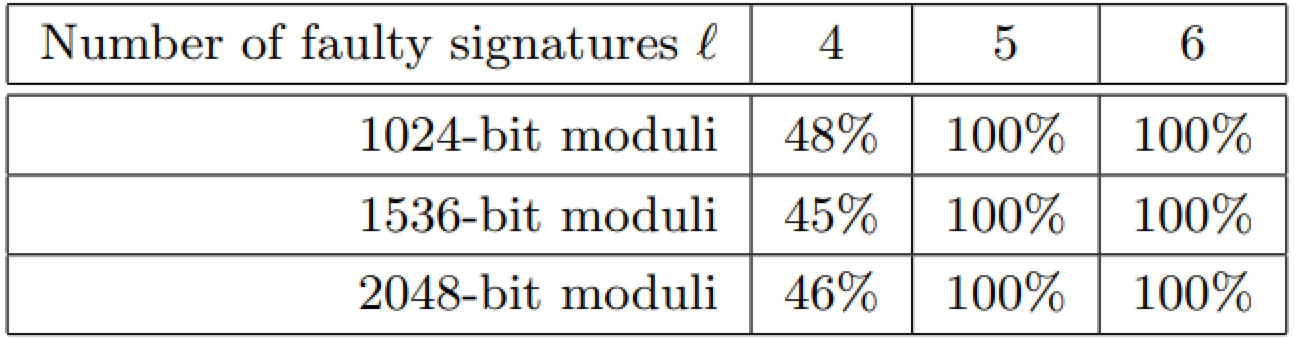
$$\rightarrow \; v = \sigma_p \cdot \alpha + \sigma_q \cdot \beta \bmod N \cdot N'$$

$$\rightarrow v = \sigma_p \cdot \alpha + \sigma_q \cdot \beta$$

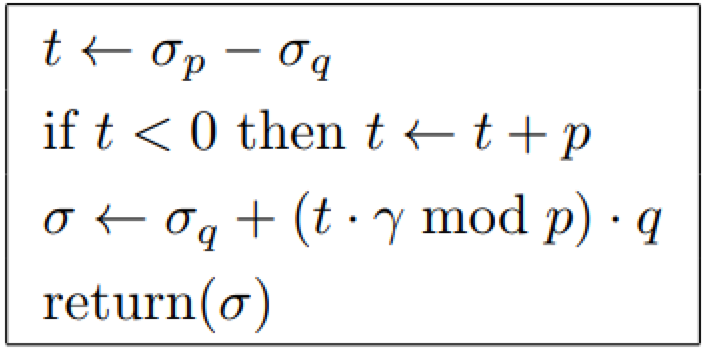


..

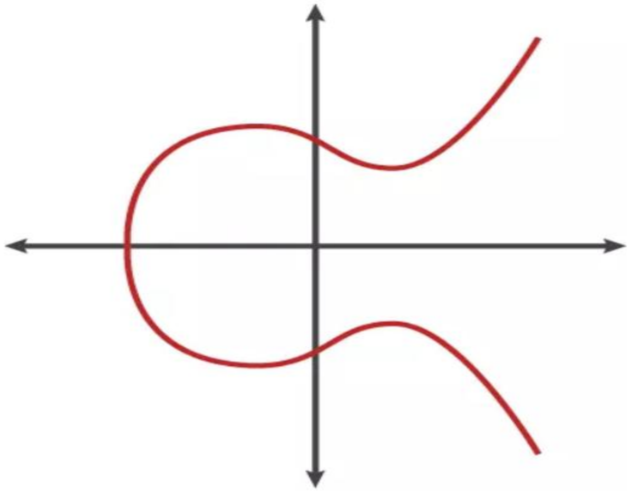
chall.py	Create chall.py
exploit.py	Create exploit.py
modulus_fault.py	Create modulus_fault.py
output.txt	Create output.txt

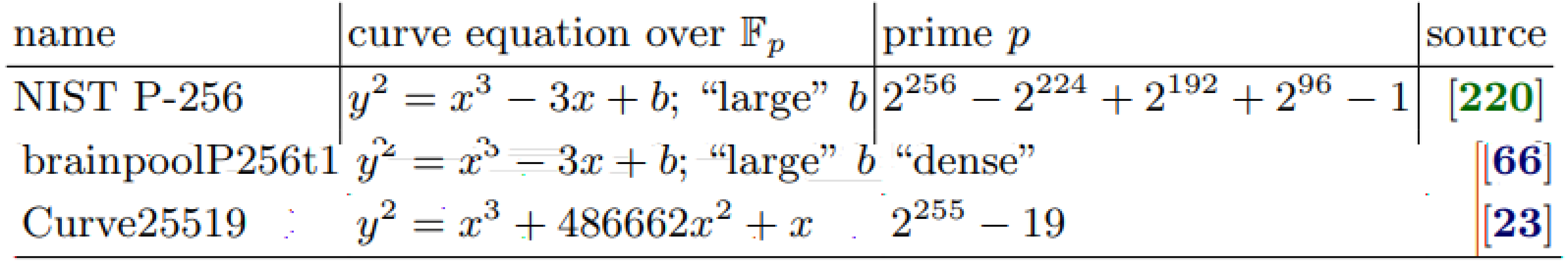


Number of faulty signatures ℓ	4	5	6
1024-bit moduli	48%	100%	100%
1536-bit moduli	45%	100%	100%
2048-bit moduli	46%	100%	100%

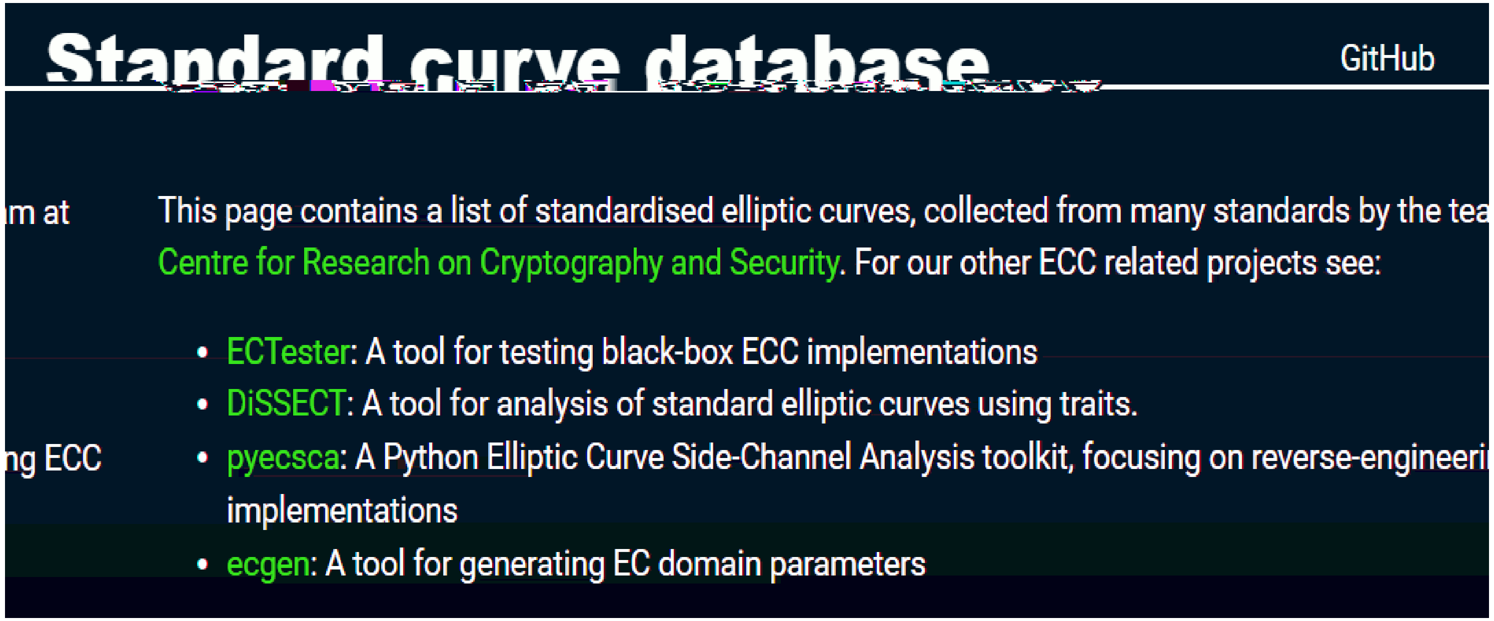


```
$t \leftarrow \sigma_p - \sigma_q$
if $t < 0$ then $t \leftarrow t + p$
$\sigma \leftarrow \sigma_q + (t \cdot \gamma \bmod p) \cdot q$
return($\sigma$)
```





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]



Standard curve database

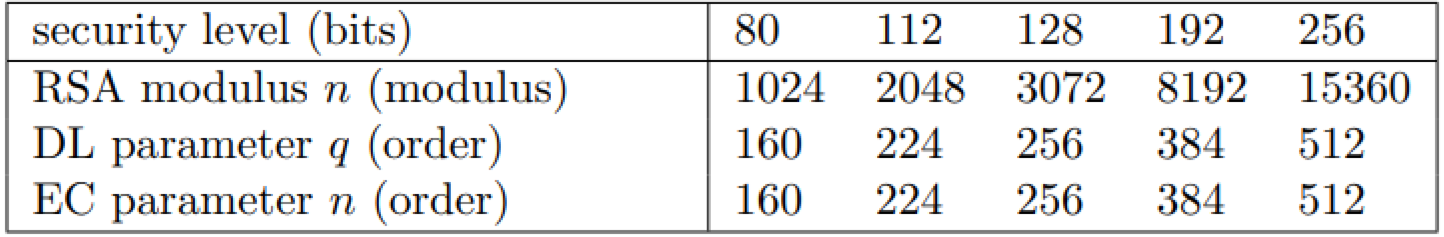
m at

ng ECC

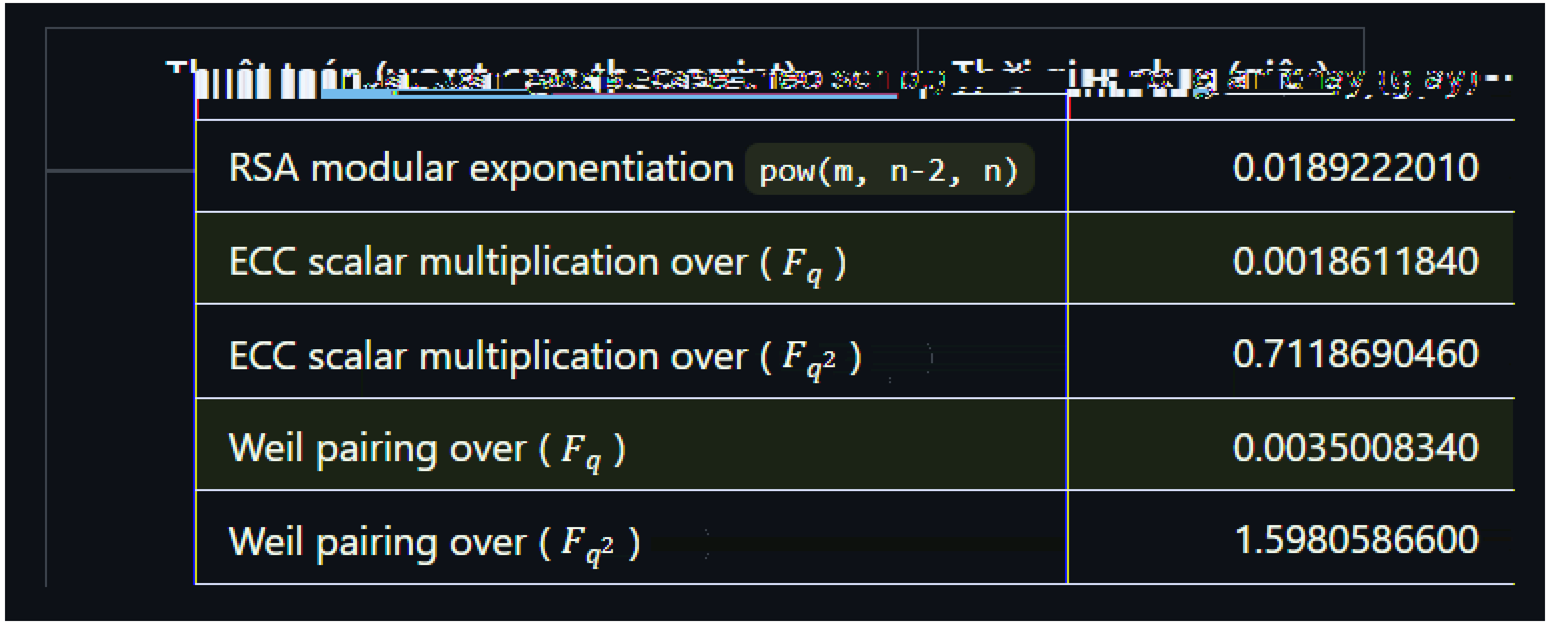
This page contains a list of standardised elliptic curves, collected from many standards by the tea
Centre for Research on Cryptography and Security. For our other ECC related projects see:

- ECTester: A tool for testing black-box ECC implementations
- DiSSECT: A tool for analysis of standard elliptic curves using traits.
- pyecsca: A Python Elliptic Curve Side-Channel Analysis toolkit, focusing on reverse-engineeri
  implementations
- ecgen: A tool for generating EC domain parameters

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



Thuật toán [illegible]	[illegible]
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

```
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
```

openssl_rsa.py Create openssl_rsa.py

py_rsa.py Create py_rsa.py