

RSA暗号運用で
やってはいけない

 n のこと

#ssmjp 2017/02

sonickun



sonickun (@y_hag)

- 大学院生（4月から就職）
- #ssmjpで過去に2度発表しました
 - 「ダークネットの話」 #ssmjp 2015/02
<http://www.slideshare.net/sonickun/ss-44926963>
 - 「進化するWebトラッキングの話」 #ssmjp 2015/07
<http://www.slideshare.net/sonickun/web-51132299>





RSA暗号

インターネット上で最もよく用いられている暗号方式のひとつ（SSL/TLS、SSH、IPsecなど）

- 数学的に安全性が示されている
- 運用を間違えると暗号が解読される可能性がある

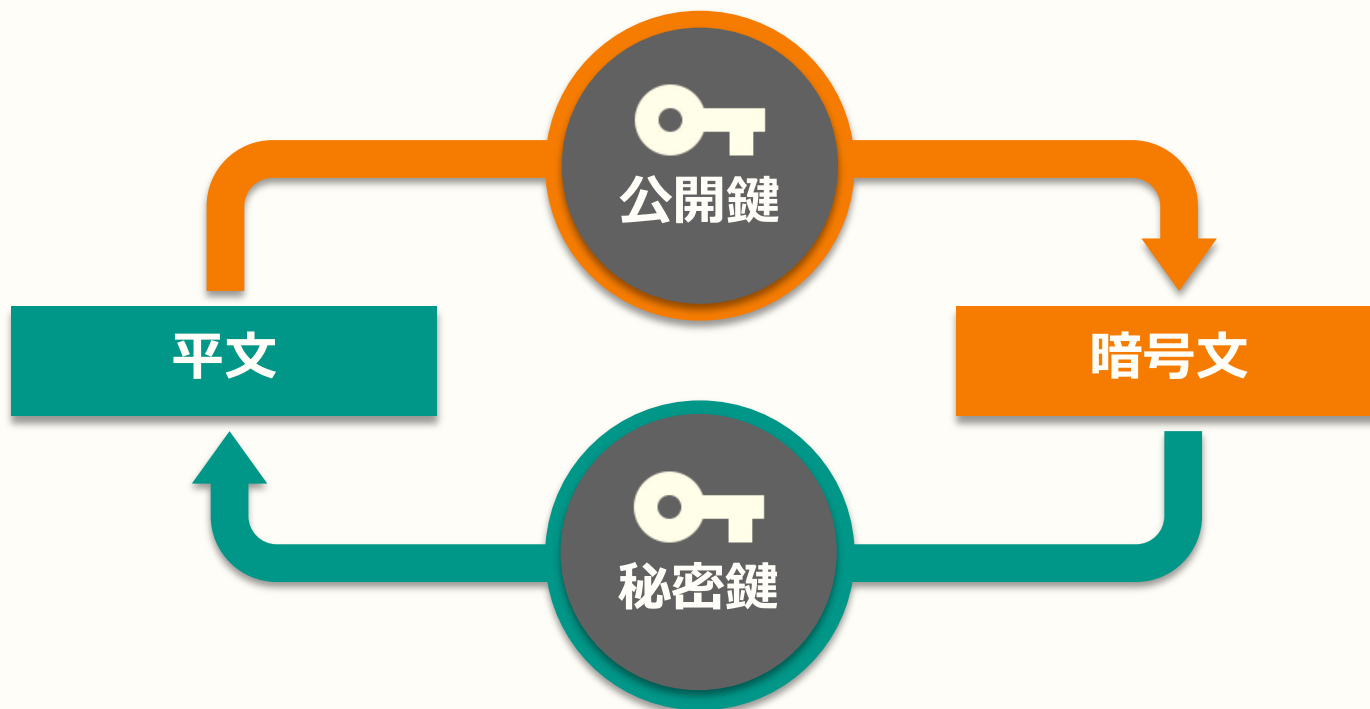


**RSA暗号の運用において
気を付けるべきことを紹介します**

RSA暗号とは



- Ron **R**ivest、Adi **S**hamir、Leonard **A**dleman によって発明された公開鍵暗号方式
- 桁数が大きな合成数の素因数分解が困難であることを安全性の根拠としている

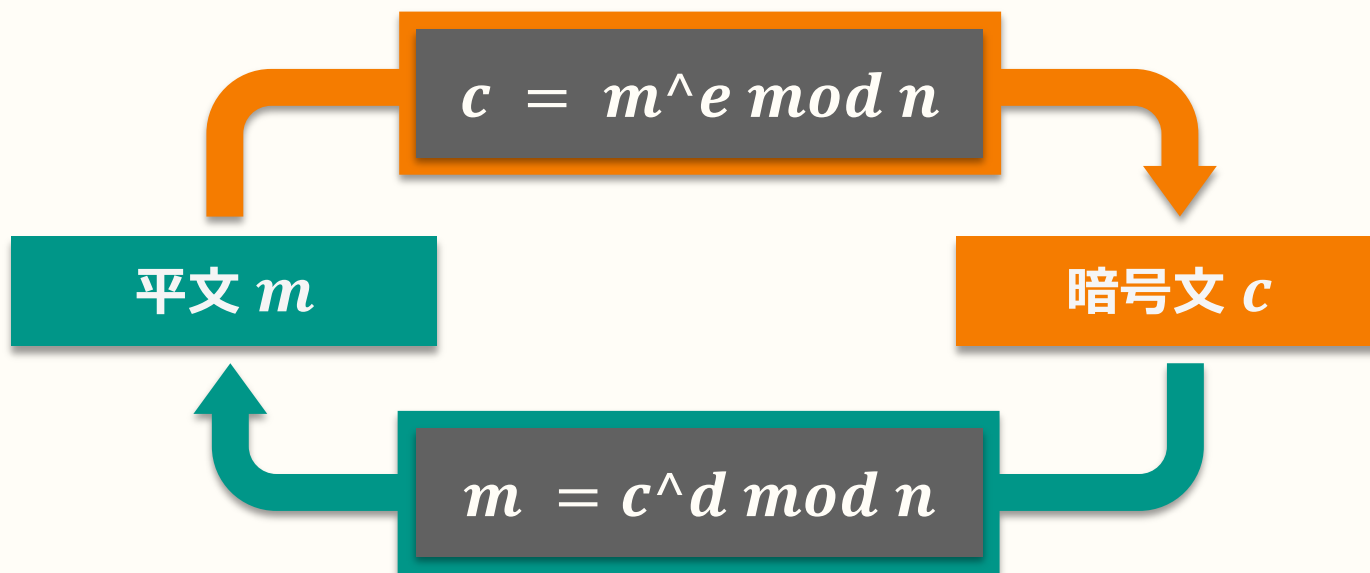


RSA暗号のアルゴリズム



鍵生成

- 素数 p, q を選ぶ
- $n = p * q, \varphi(n) = (p - 1) * (q - 1)$
- $\varphi(n)$ と互いに素となるような e を選ぶ
- $d * e \equiv 1 \pmod{\varphi(n)}$ となる最小の d を求める
- n, e を公開鍵, p, q, d を秘密鍵とする



RSA暗号運用で
やってはいけない

n のこと



攻撃が成立する厳密な条件までは言及しません

たまたに現実的でないシチュエーションも出てきます

(暗号学者ががんばって考えたので許して)

もちろん悪用厳禁で

RSA暗号運用でやってはいけない n のこと

その 1

公開鍵 n のビット数（鍵長）が 小さくてはいけない



- n が小さいと容易に素因数分解できてしまう
- 最新の計算機環境では**768bit**の整数の素因数分解に成功している (RSA Factoring Challenge)
- 2017年時点で1024bit以下は非推奨 (NIST)

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 2

p, q の片方が小さい値になってはいけない



- n の大きさが十分であっても、片方の素数が小さいと、 n を小さい素数から順に割っていくと素因数分解できてしまう

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

暗号化: $c = m^e \bmod n$ 復号: $m = c^d \bmod n$

RSA暗号運用でやってはいけない n のこと

その 3

近い値の素数 p, q を使ってはいけない



- フェルマー法で素因数分解可能
- p, q の値が近いと、それらの値は n の平方根の周辺に限定される

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 4

素数 p, q を有名な素数（メルセンヌ素数など）
にしてはいけない



- $2^n - 1$ (n は自然数) で表される自然数をメルセンヌ数といい、そのうち素数のものをメルセンヌ素数という
- 現在見つかっているメルセンヌ素数は**49個**のみ

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 5

公開鍵 n の生成時に 同じ素数を使いまわしてはいけない



- $n = p * q$, $n' = p * r$ のとき、 n と n' の公約数を計算することで p が求まる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 6

e の値が小さすぎてはいけない



- **Low Public Exponent Attack** が適用可能
- e と n が共に小さく ($e = 3$ など)、 $m^e < n$ のとき、 c の e 乗根を計算することで m が求まる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 7

e の値が大きすぎてはいけない



- **Wiener's Attack** が適用可能
- e が大きいと相対的に d が小さくなることを利用して e と n から秘密鍵が求まる
- e の値は 65537 (0x10001) が選ばれるのが一般的

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 8

同一の平文を異なる e で暗号化した
暗号文を与えてはいけない



- **Common Modulus Attack** が適用可能
- m, n が共通で e が異なる e, c の組があるとき、**拡張ユークリッド互除法**を用いて平文 m を計算できる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 9

同一の平文を異なる n で暗号化した
暗号文を与えてはいけない



- Håstad's Broadcast Attack が適用可能
- 同一の m を異なる n で暗号化した c が e 個 得られたとき、**中国人剰余定理**を用いて m が求まる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 10

任意の暗号文を復号した結果の
偶奇 (下位1bit) を知られてはいけない



- **LSB Decryption Oracle Attack** が適用可能
- c に対して m の偶奇がわかるとき、**二分探索**によって m が求まる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 11

上位ビットが共通する二つの平文に対する 暗号文を知られてはいけない



- Franklin-Reiter Related Message Attack が適用可能
- 二つの平文 $m_1, m_2 = a * m_1 + b$ と、それぞれの暗号文が得られるとき、 m_1 を導出可能

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 12

素数 p の上位ビットまたは下位ビットが
知られてはいけない



- **Coppersmith's Attack** が適用可能
- Coppersmithの定理を用いて素数 p の一部の情報 (p のビット数の1/2程度) から p を特定できる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 13

秘密鍵 d の下位ビットが知られてはいけない



- **Coppersmith's Attack (Partial Key Exposure Attack)** が適用可能
- Coppersmithの定理を用いて秘密鍵 d の一部の情報 (d のビット数の1/4程度) から d を特定できる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

RSA暗号運用でやってはいけない n のこと

その 14

平文 m の上位ビットまたは下位ビットが
知られてはいけない



- **Coppersmith's Attack** が適用可能
- Coppersmithの定理を用いて平文 m の一部の情報 (m のビット数の $1 - 1/e$ 程度) から m を特定できる

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

$$\text{暗号化: } c = m^e \bmod n \quad \text{復号: } m = c^d \bmod n$$

以上！

※ ホントは他にもあります

RSA-CRT Fault Attack

適応的選択暗号文攻撃

Coppersmith's Short Pad Attack

Boneh-Durfee Attack

乱数生成器の偏りを利用した攻撃

など...

公開鍵 n のビット数(鍵長)が小さくてはいけ
ない・ p, q の片方が小さい値になっ
てはいけ
ない・ 近い値の素数 p, q を使っ
てはいけ
ない・ 素数 p, q を有名な素数(メル
センヌ素数など)にしてはいけ
ない・ 公開鍵 n の生成時に同じ素
数を使いまわしてはいけ
ない・ e の値が小さすぎてはいけ
ない・ e の値が大きすぎてはいけ
ない・ 同一の平文を異なる e で暗
号化した暗号文を与えてはいけ
ない

RSA暗号運用でやってはいけ

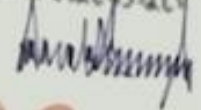
14 のこと

- ・ 同一の平文を異なる n で暗号化した暗号文を与えてはいけ
ない・ 任意の暗号文を復号した結果の偶奇 (下位1bit) を知られては
いけ
ない・ 上位ビットが共通する二つの平文に対する暗号文を知ら
れてはいけ
ない・ 素数 p の上位ビットまたは下位ビットが知ら
れてはいけ
ない・ 秘密鍵 d の下位ビットが知られてはいけ
ない・ 平文 m の上位ビットまたは下位ビットが知られてはいけ
ない

RSA暗号の 運用は難しい??

THE WHITE HOUSE
WASHINGTON

公開鍵 n のビット数(長さ)が小さくは
いけない・ p , q の片方が小さい値になっ
てはいけない・近い値の素数 p , q を使っ
てはいけない・素数 p , q を有名な素数
(メルセンヌ素数など)にしてはいけない・
公開鍵 n の生成時に同じ素数を使いまわ
してはいけない・ e の値が小さすぎではい
けない・ e の値が大きすぎではいけない・
同一の平文を異なる e で暗号化した暗号
文を与えてはいけない・同一の平文を異な
る n で暗号化した暗号文を与えてはいけ
ない・任意の暗号文を復号した結果の偶奇
(下位1bit)を知られてはいけない・上位
ビットが共通する二つの平文に対する暗号
文を知られてはいけない・素数 p の上位
ビットまたは下位ビットが知られてはいけ
ない・秘密鍵 d の下位ビットが知られて
はいけない・平文 m の上位ビットまたは
下位ビットが知られてはいけない



OpenSSL で安全な鍵を作ろう



OpenSSL

- ✓ インターネット上で標準的に利用される暗号プロトコルを実装したオープンソースのライブラリ
- ✓ 疑似乱数生成器を用いて安全なRSA鍵を生成してくれる

コマンド例

秘密鍵の生成

```
$ openssl genrsa 2048 > private-key.pem
```

公開鍵の生成

```
$ openssl rsa -pubout < private-key.pem > public-key.pem
```



■ こまめにアップデートを

- OpenSSLの脆弱性は頻繁に見つかる（2016年で34件*）
- 少し昔のバージョンではデフォルトの鍵長が短い

■ 疑似乱数生成器の seed (種) は予測不可能なものに

- UNIX系であれば `/dev/random` または `/dev/urandom`
- **EGD** (UNIX) や**EGADS** (UNIX+Windows) などのエントロピー収集ツール
- 望ましくない seed 例：時刻、特定のファイル、スクリーンショット画像、キーストロークなど

*Openssl : CVE security vulnerabilities, versions and detailed reports
http://www.cvedetails.com/product/383/Openssl-Openssl.html?vendor_id=217



- RSA暗号の運用の際には気を付けるべきことがある
- OpenSSLを正しく使えば安全な鍵が作れる



つくってみた

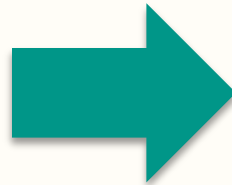
- **Cryptools - RSA実装 + 攻撃 Pythonライブラリ**
<https://github.com/sonickun/cryptools>

- **CTF暗号問題Writeupまとめ**
<https://github.com/sonickun/ctf-crypto-writeups>

さいごに...

QUIZ!!

次を示すRSA鍵は安全といえるでしょうか？



```
% openssl rsa -text -pubin < public-key.pem
```

```
Public-Key: (1024 bit)
```

```
Modulus:
```

```
00:9c:2f:65:05:89:91:20:90:6e:5a:fb:d7:55:c9:
2f:ec:42:9f:ba:19:44:66:f0:6a:ae:48:4f:a3:3c:
ab:a7:20:20:5e:94:ce:9b:f5:aa:52:72:24:91:6d:
18:52:ae:07:91:5f:bc:6a:3a:52:04:58:57:e0:a1:
22:4c:72:a3:60:c0:1c:0c:ef:38:8f:16:93:a7:46:
d5:af:bf:31:8c:0a:bf:02:76:61:ac:ab:54:e0:29:
0d:fa:21:c3:61:6a:49:82:10:e2:57:81:21:d7:c2:
38:77:42:93:31:d4:28:d7:56:b9:57:eb:41:ec:ab:
1e:aa:d8:70:18:c6:ea:34:45
```

```
Exponent:
```

```
46:6a:16:9e:8c:14:ac:89:f3:9b:5b:03:57:ef:fc:
3e:21:39:f9:b1:9e:28:c1:e2:99:f1:8b:54:95:2a:
07:a9:32:ba:5c:a9:f4:b9:3b:3e:aa:5a:12:c4:85:
69:81:ee:1a:31:a5:b4:7a:00:68:ff:08:1f:a3:c8:
c2:c5:46:fe:aa:36:19:fd:6e:c7:dd:71:c9:a2:e7:
5f:13:01:ec:93:5f:7a:5b:74:4a:73:df:34:d2:1c:
47:59:2e:14:90:74:a3:cc:ef:74:9e:ce:47:5e:3b:
6b:0c:8e:ec:ac:7c:55:29:0f:f1:48:e9:a2:9d:b8:
48:0c:fe:2a:57:80:12:75
```

```
writing RSA key
```

```
-----BEGIN PUBLIC KEY-----
```

```
MIIBHZAjBgkqhkiG9w0BAQEFAAOCQAQwAMIIBBwKBgQCCL2UFIzE
gkG5a+9dVys/s
```

```
Qp+6GURm8GquSE+jPKunICBe1M6b9apSciSRbRhSrgerRX7xq01I
EWFfgoSJMcqNg
```

```
wBwM7ziPFpOnRtWvvzGMCr8CdmGsQ1TgKQ36IcNhakmCE0JXgSH
Xwjh3QpMx1CjX
```

```
Vr1X60Hsqx6q2HAYxuo0RQKBgEZqFp6MFKyJ85tbA1fv/D4hOfm
xniJB4pnxi1SV
```

```
KgepMrpcqfS5Oz6qWhLEhWmB7hoxpbR6AGj/CB+jyMLFRv6qNhn
9bsfdccmi518T
```

```
AeyTX3pbdEpz3zTSHEdZLhSQdKPM73Sezkde02sMjuysfFUpD/F
I6aKduEgM/ipX
```

```
gBJ1
```

```
-----END PUBLIC KEY-----
```

Secure

or

Vulnerable ?

暗号文

0x1cfc3c2be23b692c3627c0fd7ad2f6b
c829c1d488107eaa6c76f2ed81d0cdd7
e16ee2794f1569efa4eb6b9526e98ef0
196d3a7d2e22aad9d4c8b6c603ac568
77db1f92d7b5885324f2fb2d2000e892
3402861ceb31f4ef63c7a2c950160717
d7195fb7fd4de794fd0b116e06ca5bff1
f964d79e1276291d2bb1e403371b971
eb

```
± % openssl rsa -text -pubin < public-key.pem
```

```
Public-Key: (1024 bit)
```

```
Modulus:
```

```
00:9c:2f:65:05:89:91:20:90:6e:5a:fb:d7:55:c9:
2f:ec:42:9f:ba:19:44:66:f0:6a:ae:48:4f:a3:3c:
ab:a7:20:20:5e:94:ce:9b:f5:aa:52:72:24:91:6d:
18:52:ae:07:91:5f:bc:6a:3a:52:04:58:57:e0:a1:
22:4c:72:a3:60:c0:1c:0c:ef:38:8f:16:93:a7:46:
d5:af:bf:31:8c:0a:bf:02:76:61:ac:ab:54:e0:29:
0d:fa:21:c3:61:6a:49:82:10:e2:57:81:21:d7:c2:
38:77:42:93:31:d4:28:d7:56:b9:57:eb:41:ec:ab:
1e:aa:d8:70:18:c6:ea:34:45
```

```
Exponent:
```

```
46:6a:16:9e:8c:14:ac:89:f3:9b:5b:03:57:ef:fc:
3e:21:39:f9:b1:9e:28:c1:e2:99:f1:8b:54:95:2a:
07:a9:32:ba:5c:a9:f4:b9:3b:3e:aa:5a:12:c4:85:
69:81:ee:1a:31:a5:b4:7a:00:68:ff:08:1f:a3:c8:
c2:c5:46:fe:aa:36:19:fd:6e:c7:dd:71:c9:a2:e7:
5f:13:01:ec:93:5f:7a:5b:74:4a:73:df:34:d2:1c:
47:59:2e:14:90:74:a3:cc:ef:74:9e:ce:47:5e:3b:
6b:0c:8e:ec:ac:7c:55:29:0f:f1:48:e9:a2:9d:b8:
48:0c:fe:2a:57:80:12:75
```

```
writing RSA key
```

```
-----BEGIN PUBLIC KEY-----
```

```
MIIBHZAQBgkqhkiG9w0BAQEFAAOCQAQwAMIIBBwKBgQCCL2UFIzE
gkG5a+9dVys/s
```

```
Qp+6GURm8GquSE+jPKunICBe1M6b9apSciSRbRhSrgerRX7xq01I
```

```
EWffgoSJMcqNg
```

```
wBwM7ziPFpOnRtWvvzGMCr8CdmGsQ1TgKQ36IcNhakmCE0JXgSH
```

```
Xwjh3QpMx1CjX
```

```
Vr1X60Hsqx6q2HAYxuo0RQKBgEZqFp6MFKyJ85tbA1fv/D4h0fm
```

```
xn1jB4pnxi1SV
```

```
KgepMrpcqfS50z6qWhLEhWmB7hoxpbR6AGj/CB+jyMLFRv6qNhn
```

```
9bsfdccmi518T
```

```
AeyTX3pbdEpz3zTSHEdZLhSQdKPM73Sezkde02sMjuysfFUpD/F
```

```
I6aKduEgM/ipX
```

```
gBJ1
```

```
-----END PUBLIC KEY-----
```

Secure

or

Vulnerable ?

e の値が極端に大きい

→ **Wiener's Attack**

適当な k に対して k/d が e/n で
近似できるため、 e/n を連分数
展開して得られる近似分数の
中から d を見つけれられる

$d = 30273e11cbe5ae0cf9054376c7645$
 $2f5ef9642c4a0d485fbe6ae6e808ff0e011$

暗号文

0x1cfc3c2be23b692c3627c0fd7ad2f6b
c829c1d488107eaa6c76f2ed81d0cdd7
e16ee2794f1569efa4eb6b9526e98ef0
196d3a7d2e22aad9d4c8b6c603ac568
77db1f92d7b5885324f2fb2d2000e892
3402861ceb31f4ef63c7a2c950160717
d7195fb7fd4de794fd0b116e06ca5bff1
f964d79e1276291d2bb1e403371b971
eb

```
% openssl rsa -text -pubin < public-key.pem
```

```
Public-Key: (1024 bit)
```

```
Modulus:
```

```
00:9c:2f:65:05:89:91:20:90:6e:5a:fb:d7:55:c9:
2f:ec:42:9f:ba:19:44:66:f0:6a:ae:48:4f:a3:3c:
ab:a7:20:20:5e:94:ce:9b:f5:aa:52:72:24:91:6d:
18:52:ae:07:91:5f:bc:6a:3a:52:04:58:57:e0:a1:
22:4c:72:a3:60:c0:1c:0c:ef:38:8f:16:93:a7:46:
d5:af:bf:31:8c:0a:bf:02:76:61:ac:ab:54:e0:29:
0d:fa:21:c3:61:6a:49:82:10:e2:57:81:21:d7:c2:
38:77:42:93:31:d4:28:d7:56:b9:57:eb:41:ec:ab:
1e:aa:d8:70:18:c6:ea:34:45
```

```
Exponent:
```

```
46:6a:16:9e:8c:14:ac:89:f3:9b:5b:03:57:ef:fc:
3e:21:39:f9:b1:9e:28:c1:e2:99:f1:8b:54:95:2a:
07:a9:32:ba:5c:a9:f4:b9:3b:3e:aa:5a:12:c4:85:
69:81:ee:1a:31:a5:b4:7a:00:68:ff:08:1f:a3:c8:
c2:c5:46:fe:aa:36:19:fd:6e:c7:dd:71:c9:a2:e7:
5f:13:01:ec:93:5f:7a:5b:74:4a:73:df:34:d2:1c:
47:59:2e:14:90:74:a3:cc:ef:74:9e:ce:47:5e:3b:
6b:0c:8e:ec:ac:7c:55:29:0f:f1:48:e9:a2:9d:b8:
48:0c:fe:2a:57:80:12:75
```

```
writing RSA key
```

```
-----BEGIN PUBLIC KEY-----
```

```
MIIBHZAQBgkqhkiG9w0BAQEFAAOCQAQwAMIIBBwKBgQCCL2UFIzE
gkG5a+9dVys/s
```

```
Qp+6GURm8GquSE+jPKunICBe1M6b9apSciSRbRhSrgerRX7xq01I
EWFfgoSJMcqNg
```

```
wBwM7ziPFpOnRtWvvzGMCr8CdmGsQ1TgKQ36IcNhakmCEOJXgSH
Xwjh3QpMx1CjX
```

```
Vr1X60Hsqx6q2HAYxuo0RQKBgEZqFp6MFKyJ85tbA1fv/D4hOfm
xnijB4pnxi1SV
```

```
KgepMrpcqfS50z6qWhLEhWmB7hoxpbR6AGj/CB+jyMLFRv6qNhn
9bsfdccmi518T
```

```
AeyTX3pbdEpz3zTSHEdZLhSQdKPM73Sezkde02sMjuysfFUpD/F
I6aKduEgm/ipX
```

```
gBJ1
```

```
-----END PUBLIC KEY-----
```

Secure

or

Vulnerable ?

e の値が極端に大きい

→ **Wiener's Attack**

適当な k に対して k/d が e/n で
近似できるため、 e/n を連分数
展開して得られる近似分数の
中から d を見つけられる

$d = 30273e11cbe5ae0cf9054376c7645$
 $2f5ef9642c4a0d485fbe6ae6e808ff0e011$

平文

"Thank you !"