

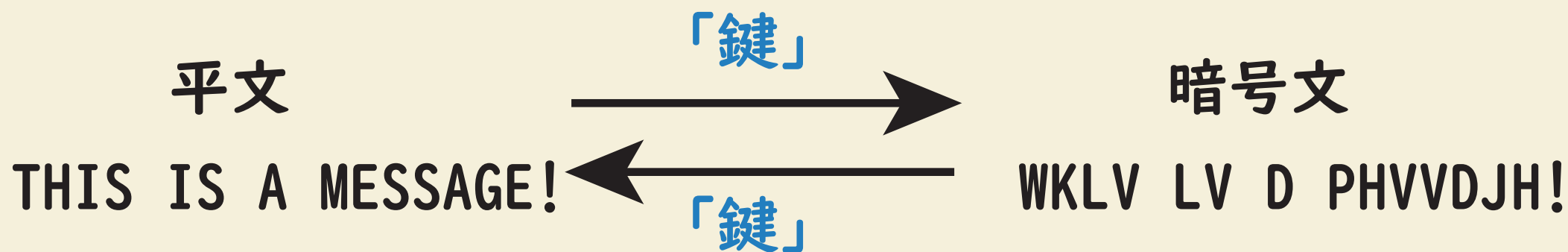
# CTF 入門 RSA 暗号

素因数分解されると危険!?

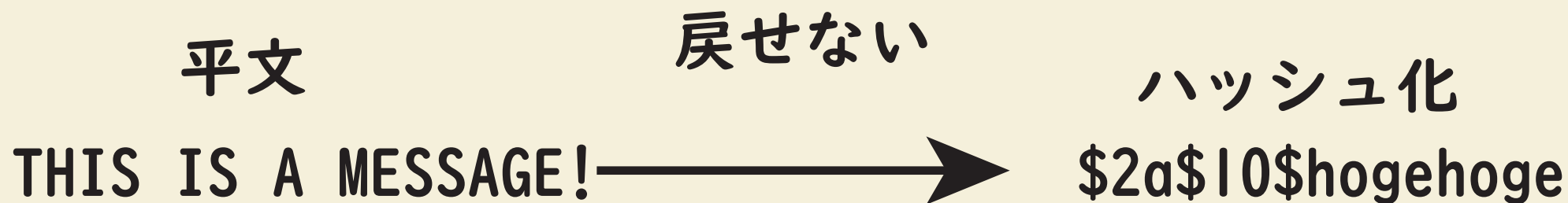
通信を守る技術と  
その弱点

暗号化って？

## 暗号化



## ハッシュ化



## 対称鍵暗号と公開鍵暗号

### 対称鍵暗号

平文

THIS IS A MESSAGE!

「鍵」

「鍵」

暗号文

WKLV LV D PHVVDJH!

### 公開鍵暗号

平文

This is Message!

暗号化専用鍵

復号専用鍵

暗号文

1298579a183e75983

## 公開鍵暗号はなぜ必要？

marroly



通信の傍受



Alice



秘密の通信がしたい



Bob

## 公開鍵暗号はなぜ必要？

marroly



通信の傍受



Alice



秘密の通信がしたい



Bob

暗号化しよう！！！！

もしこの世に対称鍵暗号しかなければ

対称鍵暗号の世界

ふむふむこれが鍵か  
暗号文全部解読できるじゃんw

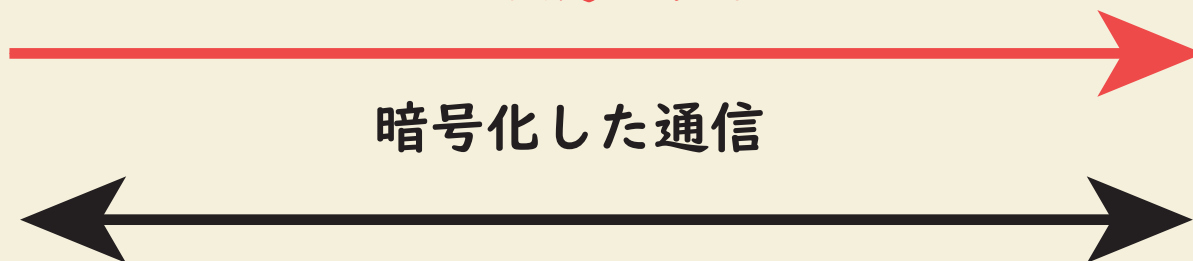


こちら鍵です！

暗号化した通信



Alice



Bob

もしこの世に対称鍵しかなければ

ば鍵か

売できるじゃんw

対称鍵問題

鍵を秘密裏に渡す必要がある！

（鍵配送問題）



Alice



Bob

## 公開鍵暗号のすごいところ

暗号化専用鍵

復号専用鍵



Alice

暗号文を作れるし、  
暗号文を解読できる

暗号化専用鍵



Bob

暗号文を作れる  
暗号文は解読できない

(盗聴)

暗号化専用鍵



marroly

暗号文は作れる  
暗号文は解読できない



Bob の暗号文は、Alice 以外に解読できない



Alice

暗号文を作れるし、  
暗号文を解読できる



Bob

暗号文を作れる  
暗号文は解読できない



marroly

暗号文は作れる  
暗号文は解読できない

## 公開鍵暗号を使った鍵交換



暗号化専用鍵

暗号化された「対称鍵」

鍵は入手できたけど  
復号用の鍵を持って  
ないから復号できない…

暗号化専用鍵

復号専用鍵



Alice

暗号化専用鍵です！

これで「対称鍵」を送ってください！



暗号化専用鍵で「対称鍵」を暗号化しました！



「対称鍵」で暗号化した通信



暗号化専用鍵



Bob

なぜ公開鍵と対称鍵を一緒に使うの？

公開鍵暗号だけで良くない？

公開鍵暗号は、対称鍵暗号に比べて「遅い」

公開鍵暗号は一方的。

公開鍵暗号はあんまりゴリゴリ使いたくない。

復号専門



Alice

暗号専門



Bob

## なぜ「公開鍵」というの？

公開する用の鍵を公開鍵

自分だけ持っておく用の鍵を秘密鍵という

暗号化専用鍵（公開鍵）

復号専用鍵（秘密鍵）

暗号化専用鍵（公開鍵）

暗号化専用鍵（公開鍵）



Alice



Bob



marroly

# 注意

「RSA 暗号を用いた電子署名」では、  
ちょっと違う感じになります。

ごちゃごちゃになるので、今日は  
説明しません。

独学をするときは、信頼できる記事で勉強してね！

## (Plain) RSA 暗号

$m$ : 平文

$c$ : 暗号文

暗号化用の鍵: ( $e$ ,  $n$ )

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

復号用の鍵: ( $d$ ,  $n$ )

この式が成り立つように  $e$ ,  $d$ ,  $n$  を頑張って作る!

## (Plain) RSA 暗号の例

$m=5$  を暗号化する

$$n = 91$$

$$e = 5$$

$$d = 29$$

$$m = 5$$

$$c = 31$$

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

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

```
>>> (5 ** e) % n
31
>>> (31 ** d) % n
5
>>>
```

## (Plain) RSA 暗号の例

$$n = 91$$

$$e = 5$$

~~$$d = 29$$~~

~~$$m = 5$$~~

$$c = 31$$



盗聴者が得られる情報

$$31 = m^3 \pmod{91}$$

ここから  $m$  を求める効率の良いアルゴリズムは知られていない。

参考：ポラード・ローの離散対数アルゴリズムという手法で、 $\mathcal{O}(\sqrt{n})$



## (Plain) RSA 暗号の例

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

ではどうやってこの式が成り立つような  
 $e, d, n$  を求めるのか？

## RSA 暗号の $e, d, n$ の求める手順

1. 2つの素数  $p, q$  を作る
2.  $n = p \times q$  とする
3.  $\phi(n) = (p - 1) \times (q - 1)$  を求める
4.  $e$  を 大きすぎず、小さすぎない、  
 $\phi(n)$  と互いに素な値に設定する  
(65537 がよく使われる)
5.  $d = e^{-1} \pmod{\phi(n)}$  を求める

note:

$n$  は

$$2^{1024}$$

$$2^{2048}$$

オーダーの数字

## 2つの素数 $p, q$ を作る

### 1. 2つの素数 $p, q$ を作る

1. 1. 適当に乱数を生成して

1. 2. それが素数かどうか判別する！

## 適当な乱数が素数である確率

### 素数定理

1 から  $n$  までの素数を  $\pi(n)$  とおくと  
 $n$  が十分大きいとき

$$\pi(n) \approx \frac{n}{\log n}$$

$n$  が素数である確率は、だいたい

$$\frac{1}{\log n}$$

## 適当な乱数が素数である確率

$n$  が素数である確率は、だいたい

$$\frac{1}{\log n}$$

$n = 2^{512}$  で 2.8% くらい

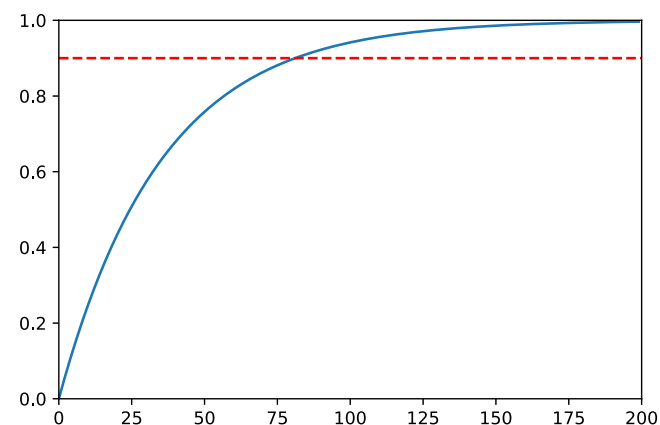
$n = 2^{1024}$  で 1.4% くらい

適当な乱数が素数である確率

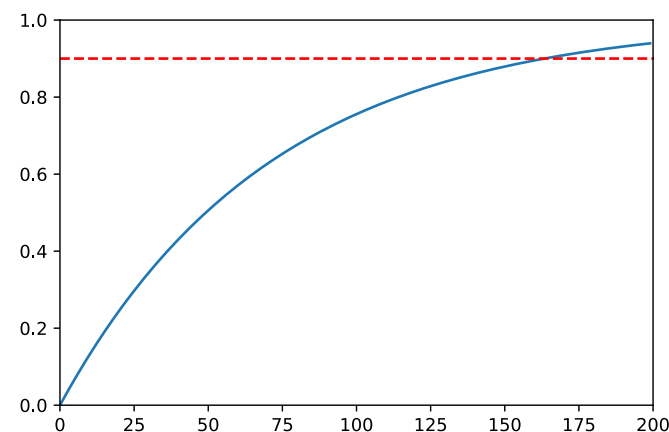
200 回くらい乱数  
生成すれば  
90% 以上で素数

素数になる確率

$$n = 2^{512}$$



$$n = 2^{1024}$$



回数

# ミラーラビンの素数判定法

というアルゴリズムを使用する

1回で  $3/4$  くらいの確率で、高速に正しく判定してくれる。

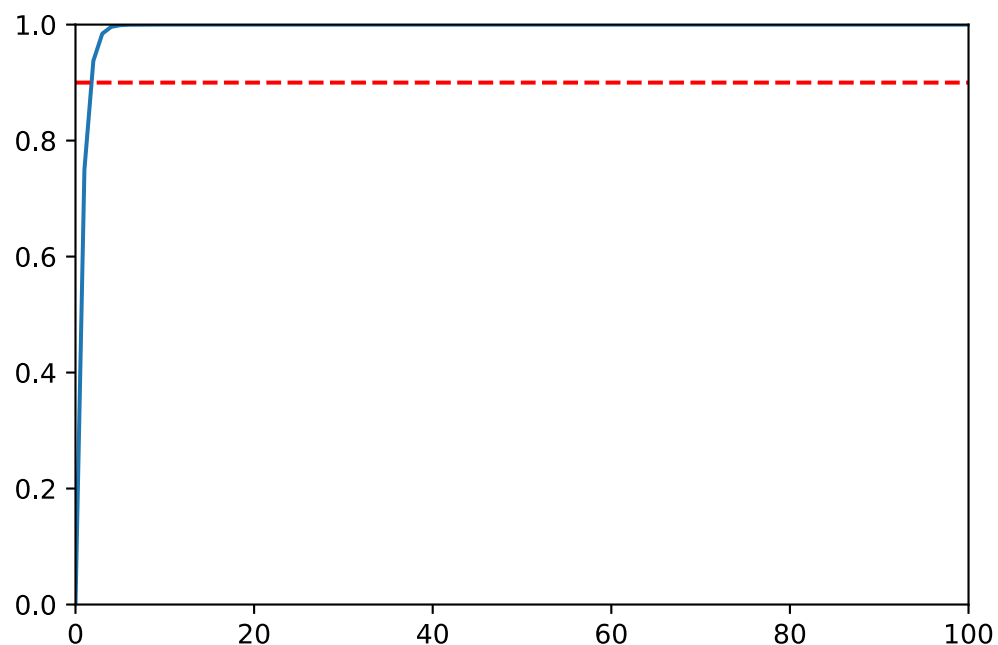
たくさん回して、高い確率で素数判定をする。

$2^{1024}$  とかに対して、エストラテネスの篩  $O(N \log \log N)$  は厳しい

## ミラーラビンの素数判定法

10 回ですでに 99.9999046% 成功する

成功率



回数



本当にこんな方法が使われてるの？

OpenSSL ではもうちょい複雑な実装になってました

openssl/crypto/bn/bn\_prime.c

```
156 /* make a random number and set the top and bottom bits */
157 if (add == NULL) {
158     if (!probable_prime(ret, bits, safe, mods, ctx))
159         goto err;
160 } else {
161     if (!probable_prime_dh(ret, bits, safe, mods, add, rem, ctx))
162         goto err;
```

```
319
320 /*
321  * Refer to FIPS 186-4 C.3.2 Enhanced Miller-Rabin Probabilistic Primality Test.
322  * OR C.3.1 Miller-Rabin Probabilistic Primality Test (if enhanced is zero).
323  * The Step numbers listed in the code refer to the enhanced case.
324  *
325  * if enhanced is set, then status returns one of the following:
326  *     BN_PRIMESTEST_PROBABLY_PRIME
327  *     BN_PRIMESTEST_COMPOSITE_WITH_FACTOR
328  *     BN_PRIMESTEST_COMPOSITE_NOT_POWER_OF_PRIME
329  * if enhanced is zero, then status returns either
330  *     BN_PRIMESTEST_PROBABLY_PRIME or
331  *     BN_PRIMESTEST_COMPOSITE
332  *
333  * returns 0 if there was an error, otherwise it returns 1.
334  */
335 int bn_miller_rabin_is_prime(const BIGNUM *w, int iterations, BN_CTX *ctx,
336                             BN_GENCB *cb, int enhanced, int *status)
337 {
```

素数っぽい

ランダムな数を

作るっぽい関数

ミラーラビンを

呼んでるっぽい場所

調査中  
信用しないでね

n を求めよう

2つの素数が求まったので、  
n を求めることができるようになった！

$$n = p \times q$$

$e, d, n$

## RSA 暗号の $e, d, n$ の求める手順

1. 2つの素数  $p, q$  を作る
2.  $n = p \times q$  とする
3.  $\phi(n) = (p - 1) \times (q - 1)$  を求める
4.  $e$  を 大きすぎず、小さすぎない、  
 $\phi(n)$  と互いに素な値に設定する  
(65537 がよく使われる)
5.  $d = e^{-1} \pmod{\phi(n)}$  を求める

## オイラーの $\phi$ 関数、オイラーの定理

### オイラーの $\phi$ 関数

$\phi(n)$  :  $1 \sim n$  までの整数で、 $n$  と互いに素  
となるものの個数

例えば...  $n=6$  のときは

$1, 2, 3, 4, 5, 6$  なので...  $\phi(n) = 2$

## オイラーのφ関数、オイラーの定理

### オイラーの定理

a と n を互いに素な整数とすると

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

n が素数のとき、

$$a^{n-1} \equiv 1 \pmod{n} \quad (\text{フェルマーの小定理})$$

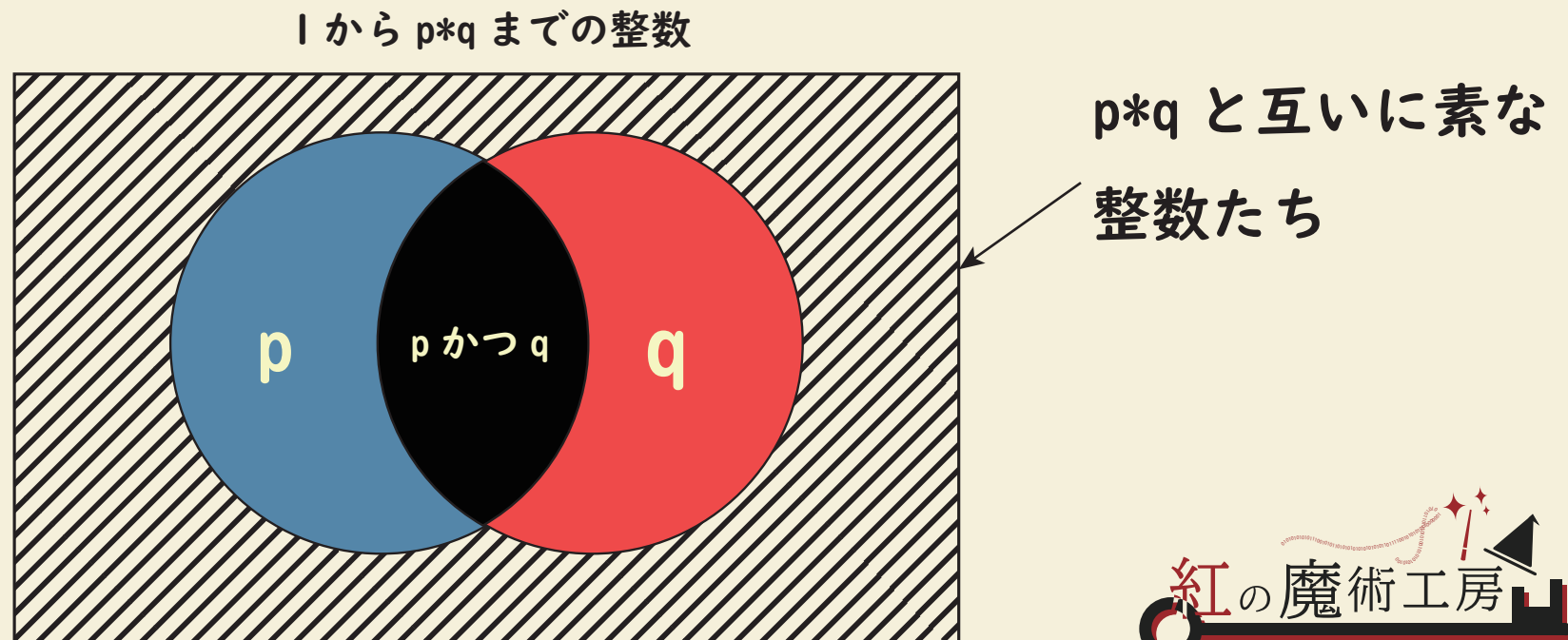
演習 : n が素数のときのオイラーのφ関数を求めてみて  
本当に↑の式が成り立つか確かえめてみよう

## n が p\*q のときの $\phi$ 関数

$\phi$  関数は p\*q と互いに素な値の個数

$\Leftrightarrow$  すべての個数 - p の倍数の個数 - q の倍数の個数 + p の倍数かつ q の倍数の個数

$$\begin{aligned}\phi(n) &= pq - q - p + 1 \\ &= (p - 1)(q - 1)\end{aligned}$$



## RSA 暗号の $e, d, n$ の求める手順

1. 2つの素数  $p, q$  を作る
2.  $n = p \times q$  とする
3.  $\phi(n) = (p - 1) \times (q - 1)$  を求める
4.  $e$  を 大きすぎず、小さすぎない、  
 $\phi(n)$  と互いに素な値に設定する  
(65537 がよく使われる)
5.  $d = e^{-1} \pmod{\phi(n)}$  を求める

e はなぜ小さすぎてはダメなのか？

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

e=3, n=91, m=2 とすると…？

$$8 = m^3 \bmod 91$$



3 乗して 8…？

2 では…？



e はなぜ小さすぎではダメなのか？

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

e=3, n=91, m=2 とすると…？

$$8 = m^3 \bmod 91$$

離散対数問題を難しくするためには  
mod でくるくるさせる必要がある。

e は何故大きすぎてはいけないのか？

次回以降  
説明

e が大きくなると、d が相対的に小さくなる

Winer' s Attack という  
攻撃が刺さってしまう

e がなぜ  $\phi(n)$  と互いに素なのか？

次のステップのこれを求めるため。

$$d = e^{-1} \mod \phi(n)$$

オイラーの定理を使った方法：今日の資料

ユークリッドの互除法を使った方法：

【CTF 入門】乱数の次の値を予測する | 【Crypto】

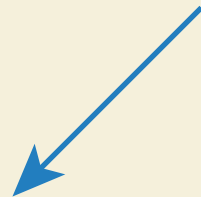
どちらの方法でも、e と  $\phi(n)$  は  
互いに素でなければならない。

## e はなぜ 65537 なのか？

1. 素数だから
2. 2進数で表したときに 0b10000000000000000000000001 となり、立っている bit が少ないから

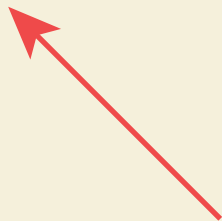
e はなぜ 65537 なのか？

逆数が求めやすそう



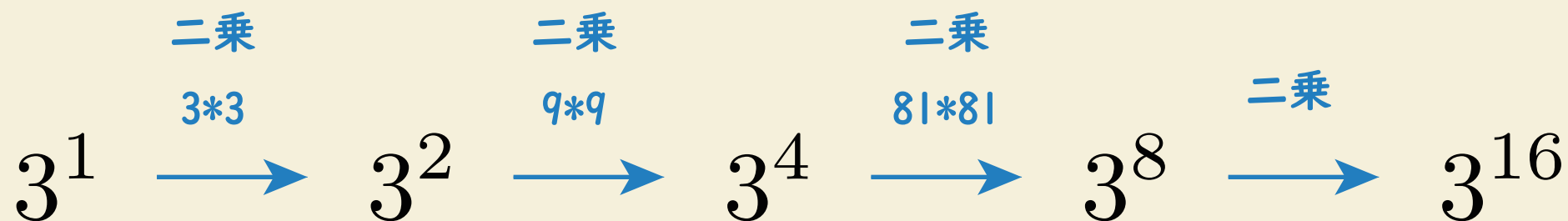
1. 素数だから

2. 2進数で表したときに 0b10000000000000000000000001 となり、  
立っている bit が少ないから



????????

## 繰り返し二乗法



$3^{16}$  を求めるのに 4 回の計算でできた！

倍々になっていくので、 $\mathcal{O}(\log N)$

## 繰り返し二乗法

$3^{22}$  はどうやって求めるのか？

22 を 2 進数表記して 0b10110 ( $2^1 + 2^2 + 2^4$ )

$$3^{22} = 3^{2^1+2^2+2^4} = 3^{2+4+16} = 3^2 \times 3^4 \times 3^{16}$$

$$\begin{array}{ccccccc} & \text{二乗} & & \text{二乗} & & \text{二乗} & \\ & 3*3 & & 9*9 & & 81*81 & \\ & \text{二乗} & & & & & \\ 3^1 & \longrightarrow & 3^2 & \longrightarrow & 3^4 & \longrightarrow & 3^8 & \longrightarrow & 3^{16} \end{array}$$

## 繰返し二乗法

1. 2進数変換する：0b10110

|

0

|

|

0



## 繰り返し二乗法

1. 2進数変換する：0b10110

1 0 1 1 0

2. bit 数分 2 乗する



## 繰り返し二乗法

1. 2進数変換する：0b10110

1 0 1 1 0

2. bit 数分 2 乗する



3. 掛け算する

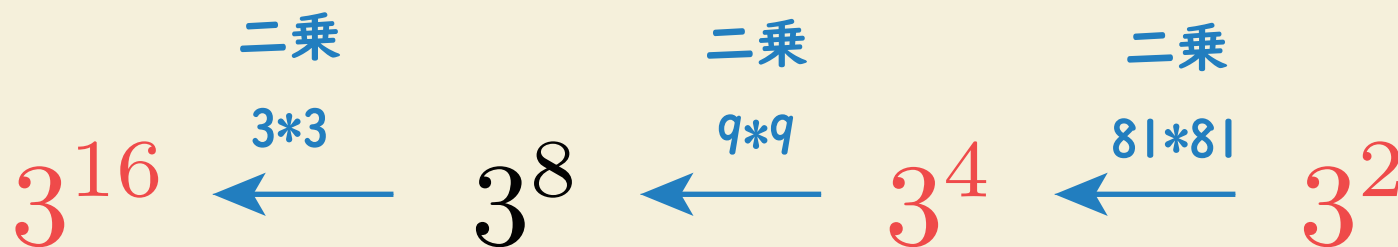
$3^{22}$

## 繰り返し二乗法

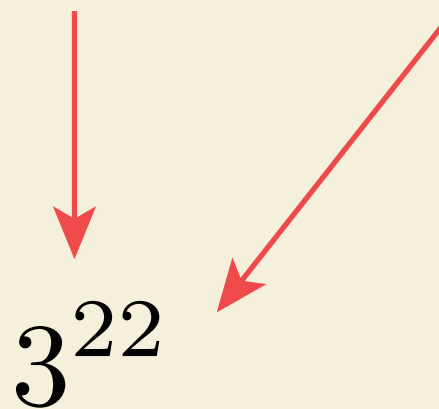
1. 2進数変換する：0b10110

|                      0                      |                      |                      0

2. bit 数分 2 乗する



3. 掛け算する



memo

2 進数表記で  
1 の数が少ないと  
最後の掛け算の  
回数が少なくなる  
から速い

## Python における pow

$\text{base}^{\text{exp}} \% \text{mod}$  を求めてくれる関数がデフォルトで存在する

**pow**(*base*, *exp*[, *mod*]) ¶

Return *base* to the power *exp*; if *mod* is present, return *base* to the power *exp*, modulo *mod* (computed more efficiently than `pow(base, exp) % mod`). The two-argument form `pow(base, exp)` is equivalent to using the power operator: `base**exp`.

ref) <https://docs.python.org/ja/3/library/functions.html>

## Python における pow

```
/* Left-to-right 5-ary exponentiation (HAC Algorithm 14.82) */
Py_INCREF(z);          /* still holds 1L */
table[0] = z;
for (i = 1; i < 32; ++i)
    MULT(table[i-1], a, table[i]);

for (i = Py_SIZE(b) - 1; i >= 0; --i) {
    const digit bi = b->ob_digit[i];

    for (j = PyLong_SHIFT - 5; j >= 0; j -= 5) {
        const int index = (bi >> j) & 0x1f;
        for (k = 0; k < 5; ++k)
            MULT(z, z, z);
        if (index)
            MULT(z, table[index], z);
    }
}
```

base<sup>31</sup> まで 求める

下位 5bit を求める

5bit 分まとめて  
掛け算する

0 のときは掛け算しない

## 繰返し二乗法

0b 10 | 000000 | 000000 | 000001

2進数で、5bit ずつ区切って、0のみの部分は掛け算しない。

Python の Pow でも速い！

e は 65537 固定でも安全？

安全。e はもともと公開する情報で、  
RSA 暗号は離散対数を求めるのが難しい  
ことが前提になっているので問題ない。

## d の求め方

d はなぜこの式でいいの？

$$d = e^{-1} \mod \phi(n)$$

暗号化したり復号化したりするときは、mod n の世界なのに！！！！

暗号化

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

復号化

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



## d の求め方

暗号化  $c = m^e \pmod n$

復号  $c^d \pmod n$

---

$$d = e^{-1} \pmod{\phi(n)}$$

$$\text{ということは... } de = 1 \pmod{\phi(n)} = X\phi(n) + 1$$

---

$$\begin{aligned} c^d &= (m^e)^d = m^{ed} = m^{X\phi(n)+1} = m^{X\phi(n)} \times m \\ &= \left(m^{\phi(n)}\right)^X \times m = 1^X \times m = m \pmod n \end{aligned}$$

オイラーの定理！

あれ？ e と d っって入れ替えても良くない？

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

$$c^e \mod n$$

$$c^e = (m^d)^e = m^{ed} = m \mod n$$

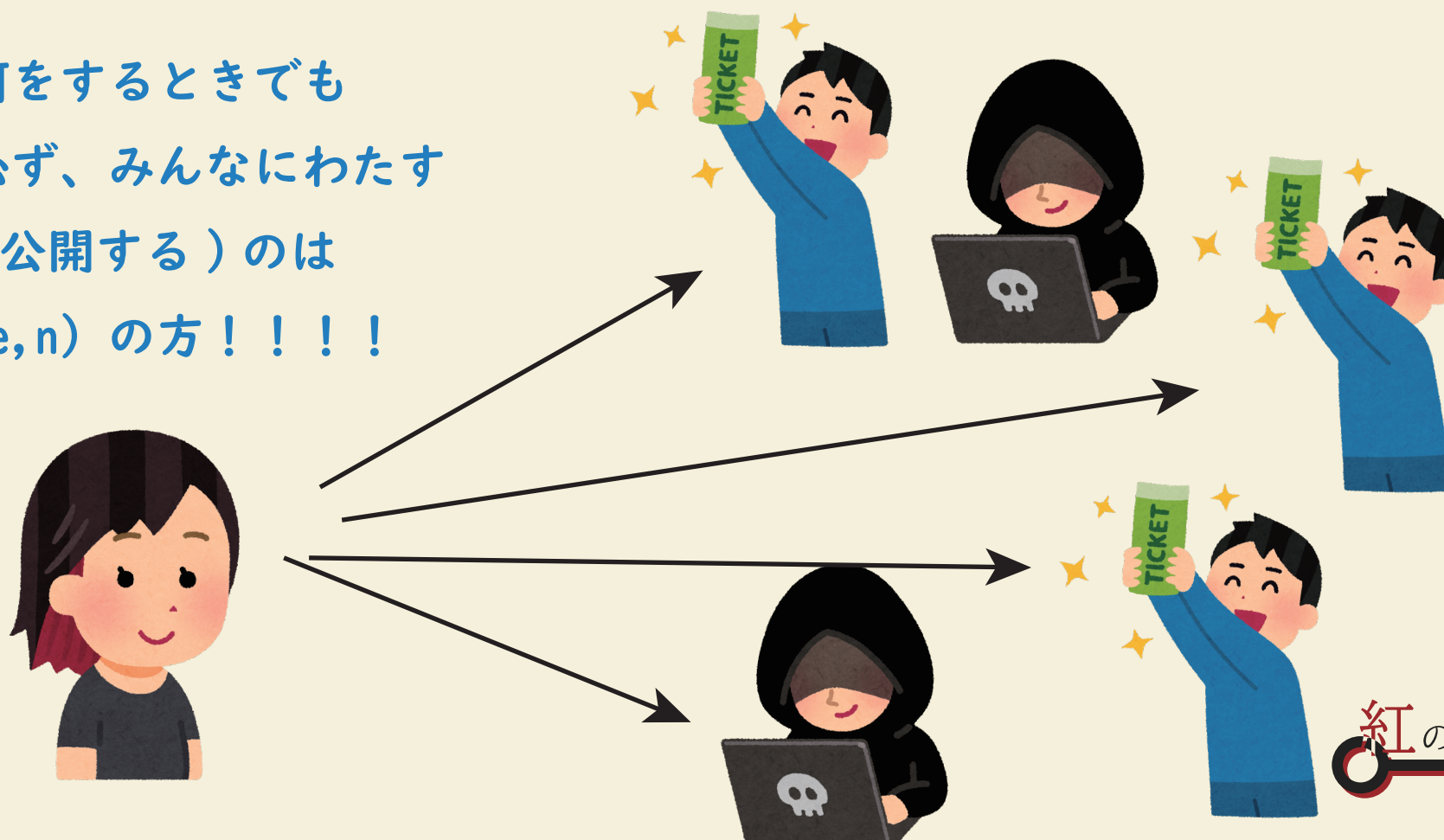
---

e に比べて、基本的に d はめっちゃめっちゃ大きい  
 → 大きな公開鍵の値で暗号化することになる  
 → Winer' s Attack が刺さる！

覚えていてほしいこと

d( 秘密鍵 ) から e( 公開鍵 ) は求められるけど  
e( 公開鍵 ) から d( 秘密鍵 ) は求められない。

何をするときでも  
必ず、みんなにわたす  
( 公開する ) のは  
( e, n ) の方 ! ! ! !



一回やってみよう！

1. 2つの素数  $p, q$  を作る

$$p = 7$$
$$q = 13$$

2.  $n = p \times q$  とする

$$n = 91$$

3.  $\phi(n) = (p - 1) \times (q - 1)$  を求める

$$\phi(n) = (p - 1)(q - 1) = 6 \times 12 = 72$$

4.  $e$  を 大きすぎず、小さすぎない、  
 $\phi(n)$  と互いに素な値に設定する  
(65537 がよく使われる)

$$e = 5$$

5.  $d = e^{-1} \pmod{\phi(n)}$  を求める

$$d = 29$$
$$ed = 5 \times 29 = 1$$

$$n = 91$$

$$e = 5$$

$$d = 29$$

$$m = 5$$

$$c = 31$$

## なぜ素因数分解されると危ないのか？

1. 2つの素数  $p, q$  を作る

2.  $n = p \times q$  とする

3.  $\phi(n) = (p-1) \times (q-1)$  を求める

4.  $e$  を 大きすぎず、小さすぎない、  
 $\phi(n)$  と互いに素な値に設定する  
(65537 がよく使われる)

5.  $d = e^{-1} \pmod{\phi(n)}$  を求める

$n$  から  $p, q$  が分かると

$$n = 91$$

$$e = 5$$

5 番まで  
成立してしまう!!!

# RSA 暗号は、意外と簡単に作れる！

今日紹介したのは PlainRSA といって、最近は使われてない  
よく使われてるのは  
RSA-OAEP というもうちょい複雑なやつ

暗号化	$c = m^e \bmod n$	公開鍵：(e, n)
復号	$m = c^d \bmod n$	秘密鍵：(d, n)

## RSA 暗号まとめ

RSA 暗号は、 **$n$  の素因数分解が難しい**ということと  
**離散対数を求めるのが難しい**仮定のもとで、成立  
している

公開鍵は、公開する用の鍵！

秘密鍵は、秘密にするような鍵！

入れ替えちゃだめ！

## 素数生成に関する問題

99 個の 9 を含む素数を一つ生成してください。

<https://github.com/kurenaif/kurenaifCTF/tree/master/ninety-nine>



## RSA 暗号の作り方の復習

$p, q, n, e, c$  が与えられるので、平文  $m$  を複合してください。

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_basic](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_basic)

## RSA 暗号の典型 CTF 問題

難易度順とは限らないよ！

CTF では自分で全部実装する必要はないから、難しそうだったら  
世の中にあるツールを使って解いてみてね！

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_1](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_1)

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_2](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_2)

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_3](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_3)

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_4](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_4)

[https://github.com/kurenaif/kurenaifCTF/tree/master/rsa\\_5](https://github.com/kurenaif/kurenaifCTF/tree/master/rsa_5)