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

@kurenaif

暗号化って？

暗号化

「鍵」

平文 → 暗号文

THIS IS A MESSAGE! ⇄ WKLV LV D PHVVDJH!

「鍵」

ハッシュ化

戻せない

平文 → ハッシュ化

THIS IS A MESSAGE! → $2a$10$hogehoge

対称鍵暗号と公開鍵暗号

対称鍵暗号

平文 $\xrightarrow{\text{「鍵」}}$ 暗号文

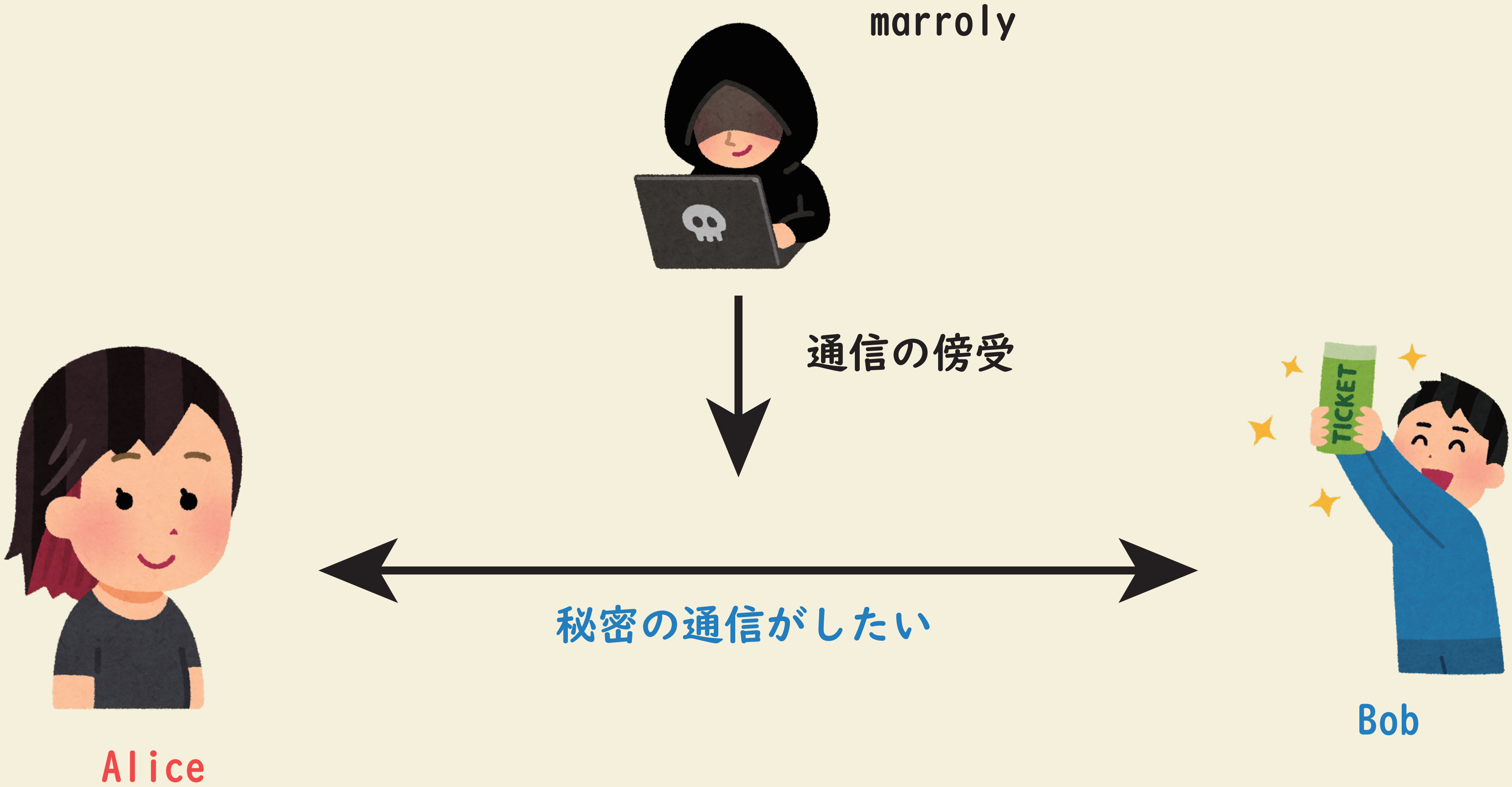
THIS IS A MESSAGE! $\xleftarrow[\text{「鍵」}]{}$ WKLV LV D PHVVDJH!

公開鍵暗号

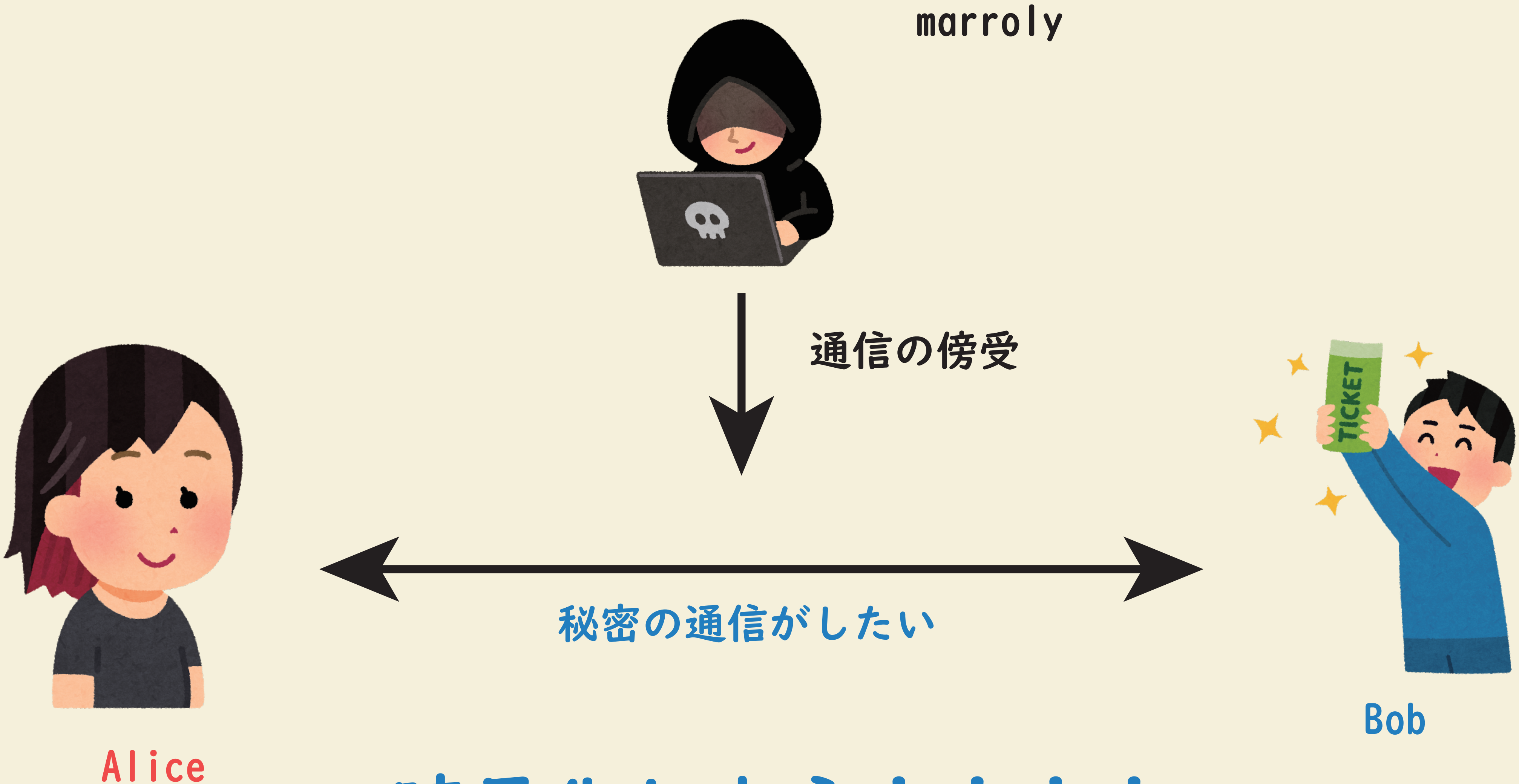
平文 $\xrightarrow{\text{暗号化専用鍵}}$ 暗号文

This is Message! $\xleftarrow[\text{復号専用鍵}]{}$ 1298579a183e75983

公開鍵暗号はなぜ必要？



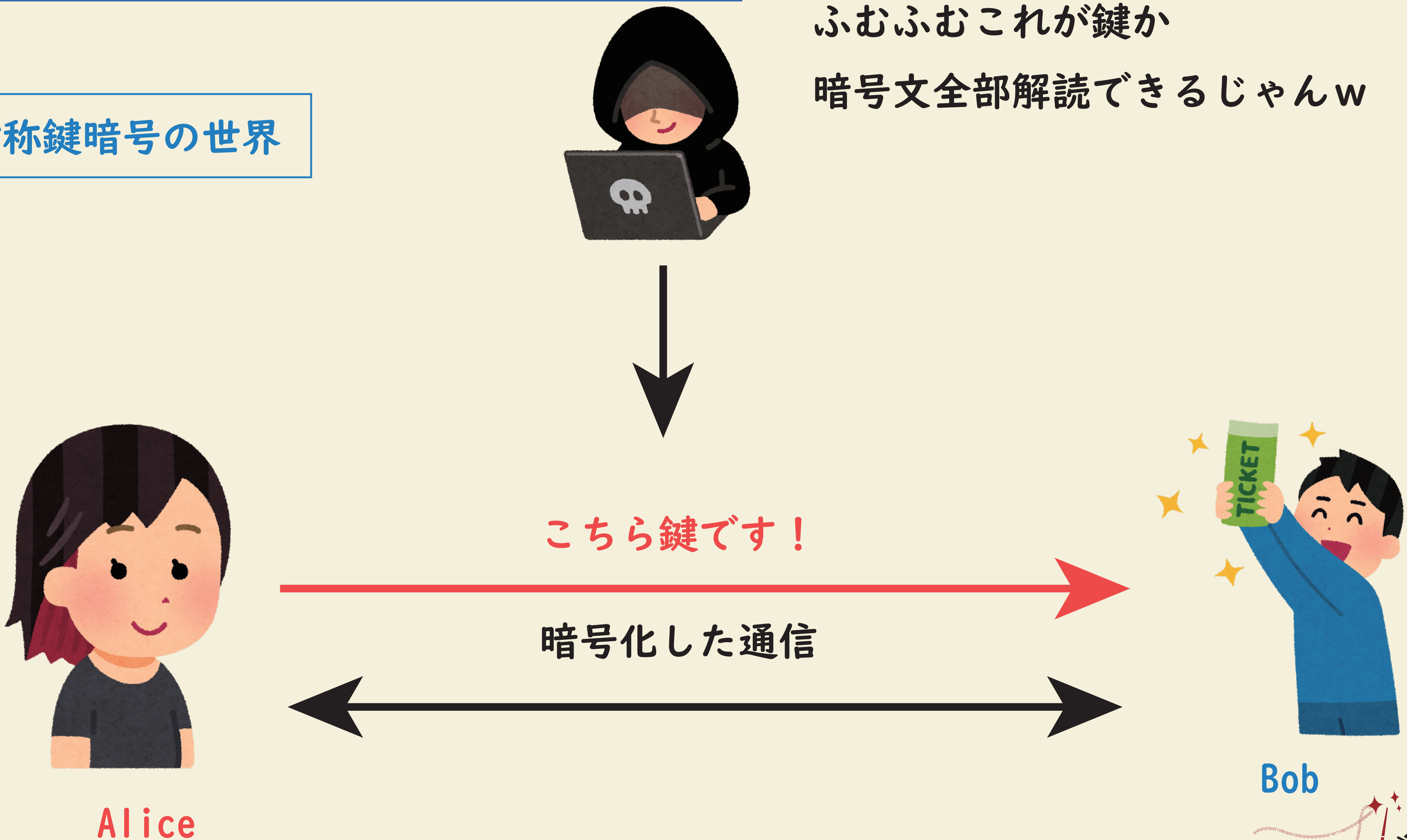
公開鍵暗号はなぜ必要？



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



もしこの世に対称鍵 … しかなけれ

…が鍵か

…読できるじゃんw

対称…

鍵を秘密裏に渡す必要がある！

（鍵配送問題）

Alice

Bob

紅の魔術工房

Atelier kurenaif

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

紅の魔術工房
Atelier kurenaif

公開鍵暗号のすごいところ

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

marroly

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」

公開鍵暗号は一方向的。

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）

Alice

暗号化専用鍵（公開鍵）

Bob

暗号化専用鍵（公開鍵）

marroly

紅の魔術工房
Atelier kurenaif

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

$n = 91$

$e = 5$

$\cancel{d = 29}$

$\cancel{m = 5}$

$c = 31$

盗聴者が得られる情報

$$31 = m^3 \mod 91$$

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、$\mathcal{O}\left(\sqrt{n}\right)$

(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、
ϕ(n) と互いに素な値に設定する
(65537 がよく使われる)

5. $d = e^{-1} \mod \phi(n)$ を求める

note:
n は
2^{1024}
2^{2048}
オーダーの数字

紅の魔術工房
Atelier kurenaif

1. ２つの素数 p, q を作る
    1. 1. 適当に乱数を生成して
    2. 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

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

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

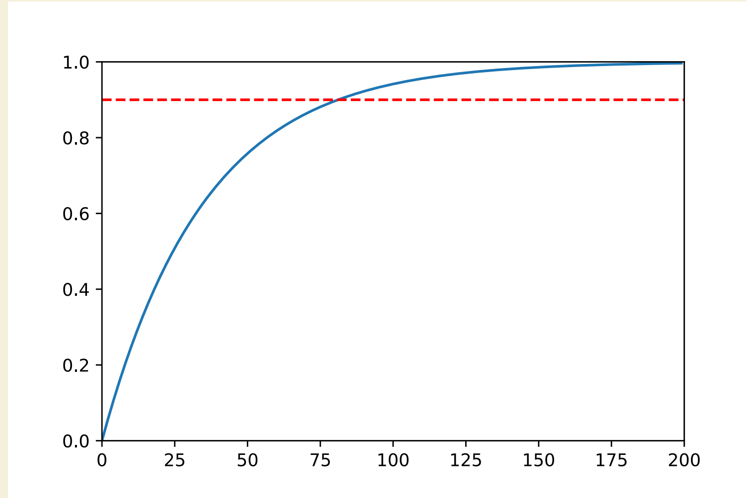
適当な乱数が素数である確率

200回くらい乱数

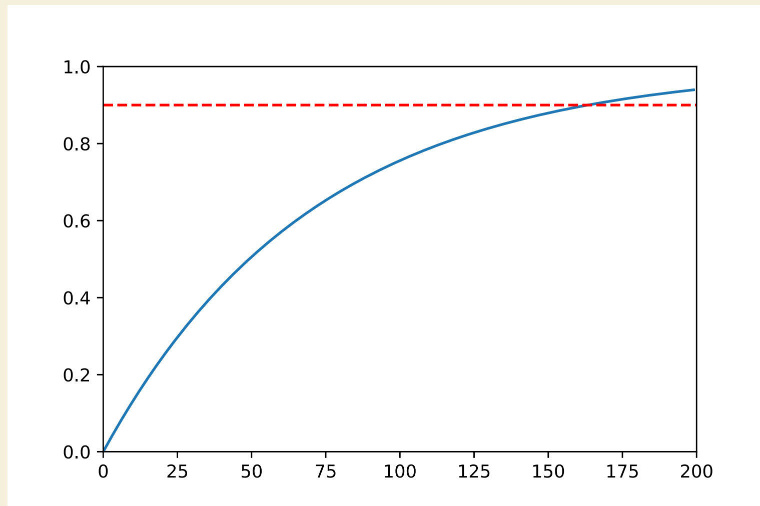
生成すれば

90%以上で素数

$$n = 2^{512}$$



$$n = 2^{1024}$$



素数になる確率

回数

大きな数の素数判定

詳細省略
ググってね

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

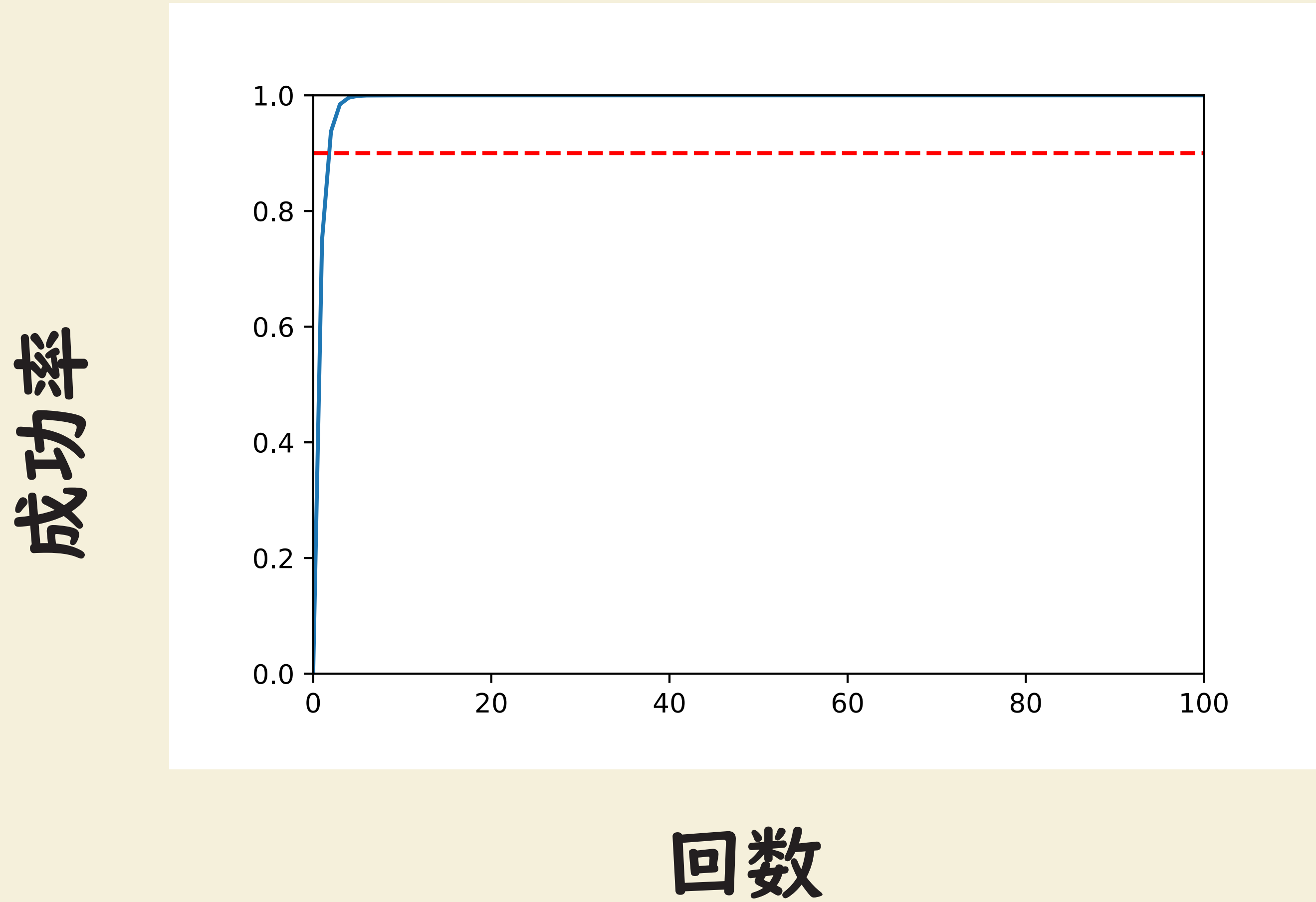
1回で3/4くらいの確率で、高速に正しく判定してくれる。

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 $\mathcal{O}(N \log\log N)$ 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



本当にこんな方法が使われてるの？

調査中
信用しないでね

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

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

素数っぽい
ランダムな数を
作るっぽい関数

```c

/*
 * Refer to FIPS 186-4 C.3.2 Enhanced Miller-Rabin Probabilistic Primality Test.
 * OR C.3.1 Miller-Rabin Probabilistic Primality Test (if enhanced is zero).
 * The Step numbers listed in the code refer to the enhanced case.
 *
 * if enhanced is set, then status returns one of the following:
 *     BN_PRIMETEST_PROBABLY_PRIME
 *     BN_PRIMETEST_COMPOSITE_WITH_FACTOR
 *     BN_PRIMETEST_COMPOSITE_NOT_POWER_OF_PRIME
 * if enhanced is zero, then status returns either
 *     BN_PRIMETEST_PROBABLY_PRIME or
 *     BN_PRIMETEST_COMPOSITE
 *
 * returns 0 if there was an error, otherwise it returns 1.
 */
int bn_miller_rabin_is_prime(const BIGNUM *w, int iterations, BN_CTX *ctx,
                             BN_GENCB *cb, int enhanced, int *status)
{
```

ミラーラビンを
呼んでるっぽい場所

source: https://github.com/openssl/openssl/blob/80b94a5adb461f94629d36db351a051b0a890856/crypto/bn/bn_prime.c

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！

$$n = p \times q$$

$e, d, \boldsymbol{n}$

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、
φ(n) と互いに素な値に設定する
(65537 がよく使われる)

5. $d = e^{-1} \mod \phi(n)$ を求める

紅の魔術工房
Atelier kurenaif

オイラーのφ関数、オイラーの定理

オイラーのφ関数

$\phi(n)$：1〜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

$1,\cancel{2},\cancel{3},\cancel{4},5,\cancel{6}$ 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

$$a^{\phi(n)} = 1 \mod n$$

nが素数のとき、

$$a^{n-1} = 1 \mod n$$ （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

note：公約数を持つのは、p の倍数か、q の倍数

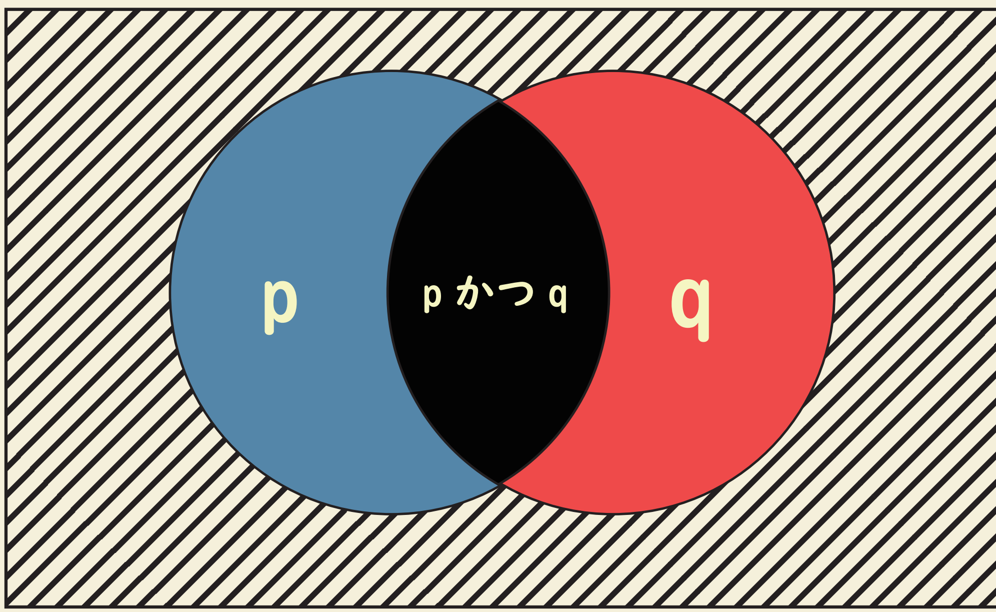
n が p*q のときの φ 関数

φ 関数は p*q と互いに素な値の個数

⇔ 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

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

1 から p*q までの整数



p*q と互いに素な整数たち

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、
φ(n) と互いに素な値に設定する
(65537 がよく使われる)

5. $d = e^{-1} \mod \phi(n)$ を求める

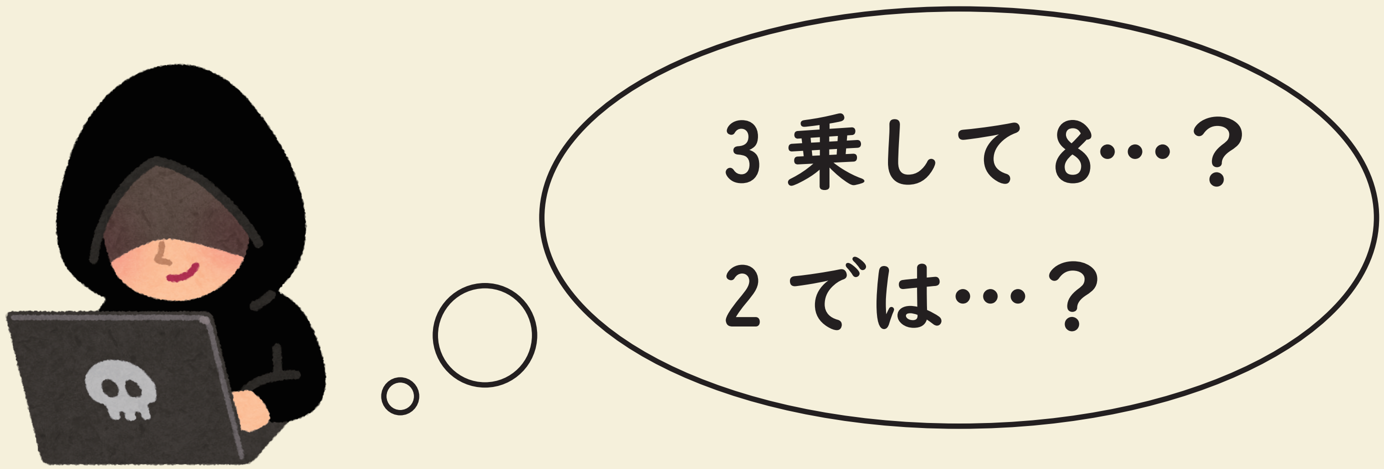
紅の魔術工房
Atelier kurenaif

e はなぜ小さすぎてはだめなのか？

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

e=3, n=91, m=2 とすると…？

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



e はなぜ小さすぎてはだめなのか？

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

e=3, n=91, m=2 とすると…？

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

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φ(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φ(n)は

互いに素でなければならない。

紅の魔術工房
Atelier kurenaif

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 0b10000000000000001 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 0b1000000000000000１ となり、立っている bit が少ないから

？？？？？？？？

繰り返し二乗法

$$3^1 \xrightarrow[\text{3*3}]{\text{二乗}} 3^2 \xrightarrow[\text{9*9}]{\text{二乗}} 3^4 \xrightarrow[\text{81*81}]{\text{二乗}} 3^8 \xrightarrow{\text{二乗}} 3^{16}$$

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 $(2^1 + 2^2 + 2^4)$

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

$$3^1 \xrightarrow[\;]{\substack{\text{二乗} \\ 3*3}} 3^2 \xrightarrow[\;]{\substack{\text{二乗} \\ 9*9}} 3^4 \xrightarrow[\;]{\substack{\text{二乗} \\ 81*81}} 3^8 \xrightarrow{\text{二乗}} 3^{16}$$

繰り返し二乗法

1. ２進数変換する：0b10110

1　0　1　1　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

$$3^{16} \xleftarrow[\text{}]{\text{二乗}\ 3*3} 3^{8} \xleftarrow{\text{二乗}\ 9*9} 3^{4} \xleftarrow{\text{二乗}\ 81*81} 3^{2} \xleftarrow{\text{二乗}} 3^{1}$$

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

$$
\begin{tikzcd}
3^{16} \arrow[drr] & 3^{8} \arrow[l, "\text{二乗}\ 3*3"'] & 3^{4} \arrow[l, "\text{二乗}\ 9*9"'] \arrow[d] & 3^{2} \arrow[l, "\text{二乗}\ 81*81"'] \arrow[dl] & 3^{1} \arrow[l, "\text{二乗}"'] \\
 & & 3^{22} & &
\end{tikzcd}
$$

3. 掛け算する

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

$$3^{16} \xleftarrow[\text{二乗}]{3*3} 3^{8} \xleftarrow[\text{二乗}]{9*9} 3^{4} \xleftarrow[\text{二乗}]{81*81} 3^{2}$$

3. 掛け算する

$$3^{22}$$

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

base^exp % 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

base^31 まで 求める

```
/* 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);
    }
}
```

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

https://github.com/python/cpython/blob/109fc2792a490ee5cd8a423e17d415fbdedec5c8/Objects/longobject.c#L4404-L4422

繰り返し二乗法

0b 10 | 00000 | 00000 | 00001

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 $c = m^e \mod n$

復号 $c^d \mod n$

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

ということは… $de = 1 \mod \phi(n) = X\phi(n) + 1$

$$c^d = (m^e)^d = m^{ed} = m^{X\phi(n)+1} = m^{X\phi(n)} \times m$$

$$= \left(m^{\phi(n)}\right)^X \times m = 1^X \times m = m \mod n$$

オイラーの定理！

あれ？eとd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はめちゃめちゃ大きい

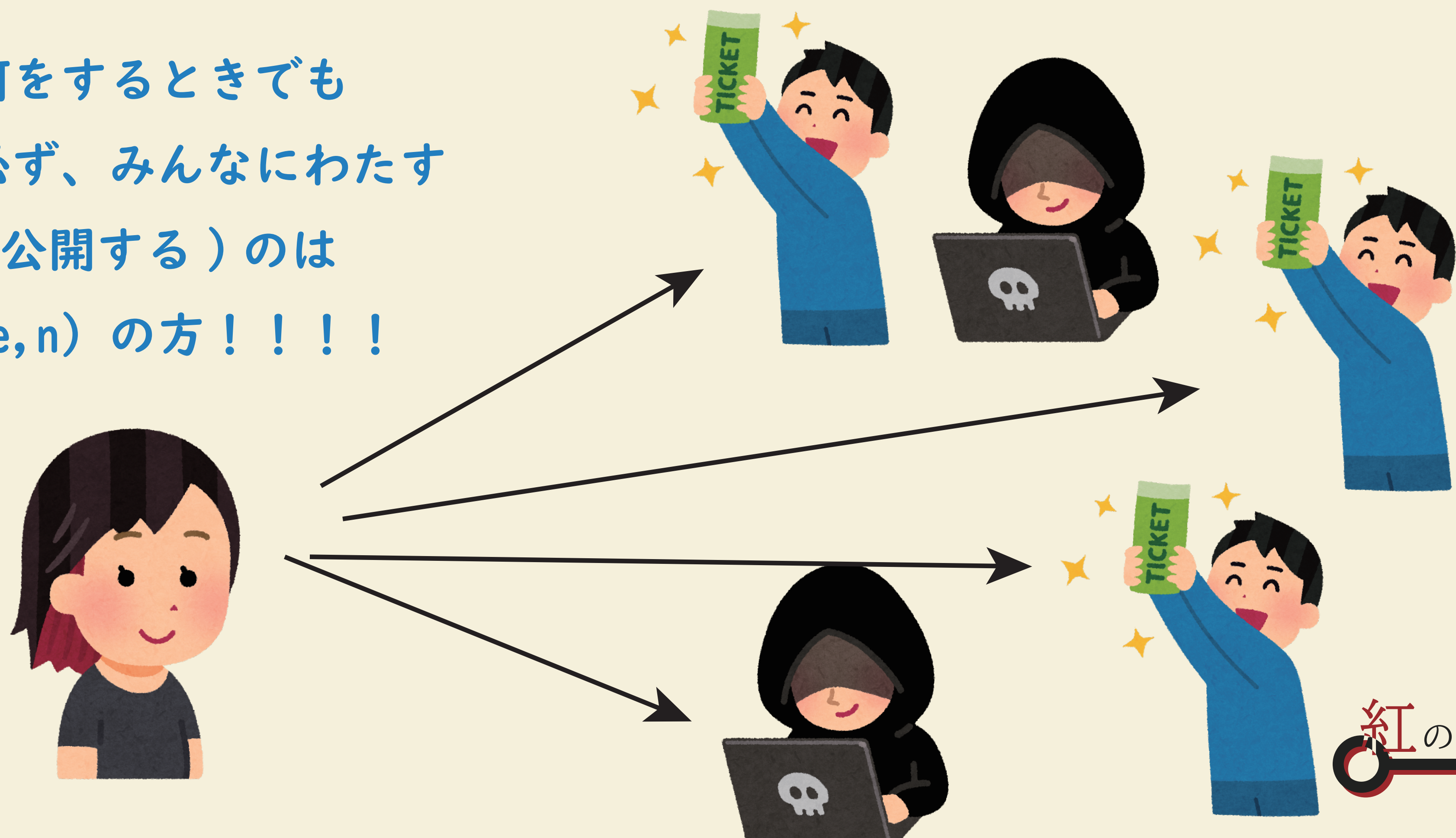
→大きな公開鍵の値で暗号化することになる

→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、 φ(n) と互いに素な値に設定する (65537 がよく使われる)	$e = 5$
5. $d = e^{-1} \mod \phi(n)$ を求める	$d = 29$ $ed = 5 \times 29 = 1 \mod \phi(n)$

$n = 91$

$e = 5$

$d = 29$

$m = 5$

$c = 31$

紅の魔術工房
Atelier kurenaif

なぜ素因数分解されると危ないのか？

n から p, q が分かると

1. ２つの素数 p, q を作る	
2. $n = p \times q$ とする	$\boldsymbol{n = 91}$
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、φ(n) と互いに素な値に設定する (65537 がよく使われる)	$e = 5$
5. $d = e^{-1} \mod \phi(n)$ を求める	5 番まで 成立してしまう！！！

RSA暗号まとめ

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 $c = m^e \mod n$ 公開鍵：(e,n)

復号 $m = c^d \mod n$ 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、秘密にするようの鍵！
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

紅の魔術工房
Atelier kurenaif

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 を復合してください。

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_2

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_5