SECCON Beginners CTF 2021

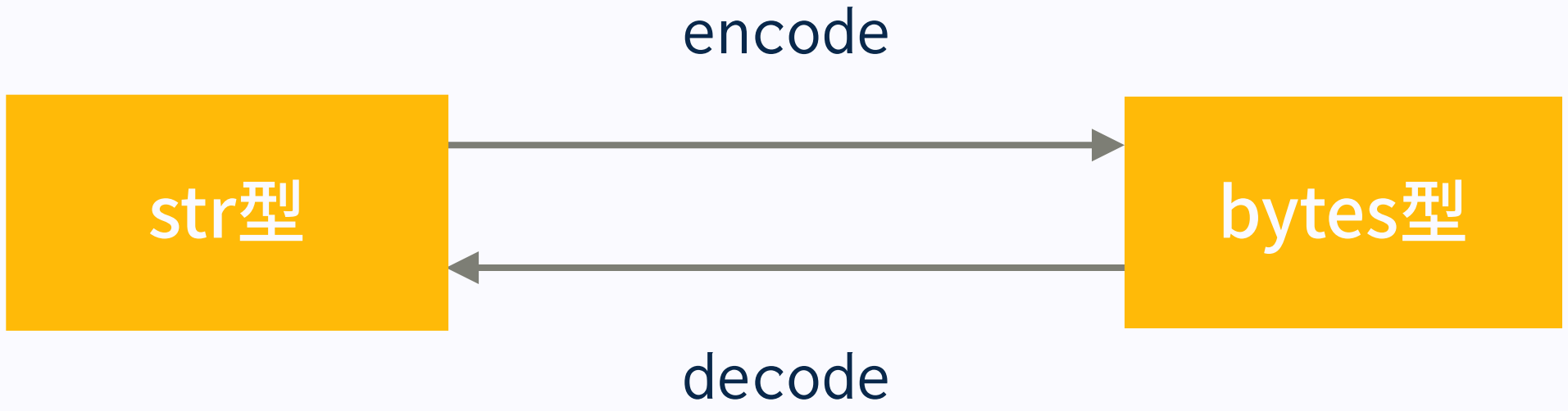
Crypto 問題の解説

濵田幸希 (HK_ilohas)

文字列と数値の相互変換

str型とbytes型の変換

文字列（str型）を数値（int型）にする前に，
str型をbytes型に変換する必要がある．



str型とbytes型の変換

```
>>> flag = "Hello, World!"
>>> flag.encode()
b'Hello, World!'
```

str型

bytes型

```
>>> flag = b"Hello, World!"
>>> flag.decode()
'Hello, World!
```

bytes型

str型

bytes型とint型の変換

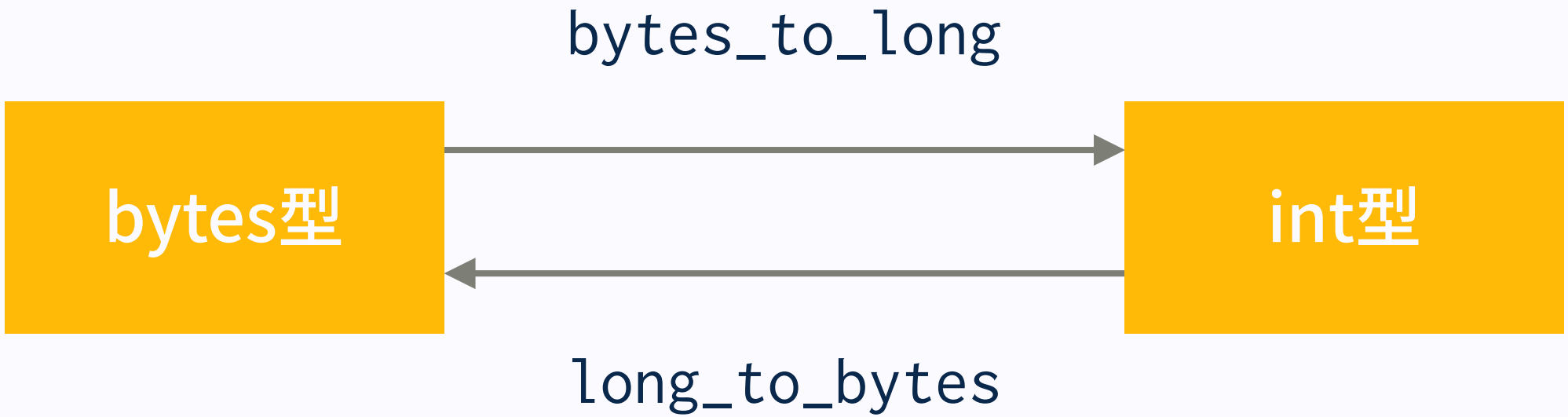
bytes型のままだと計算ができないのでint型に変換したい．
そこで，pycryptodomeという暗号系のライブラリを使用する．
https://pypi.org/project/pycryptodome/

```
$ pip3 install pycryptodome
```

bytes型とint型の変換

`bytes_to_long()`: bytes型をint型に変換する関数

`long_to_bytes()`: int型をbytes型に変換する関数



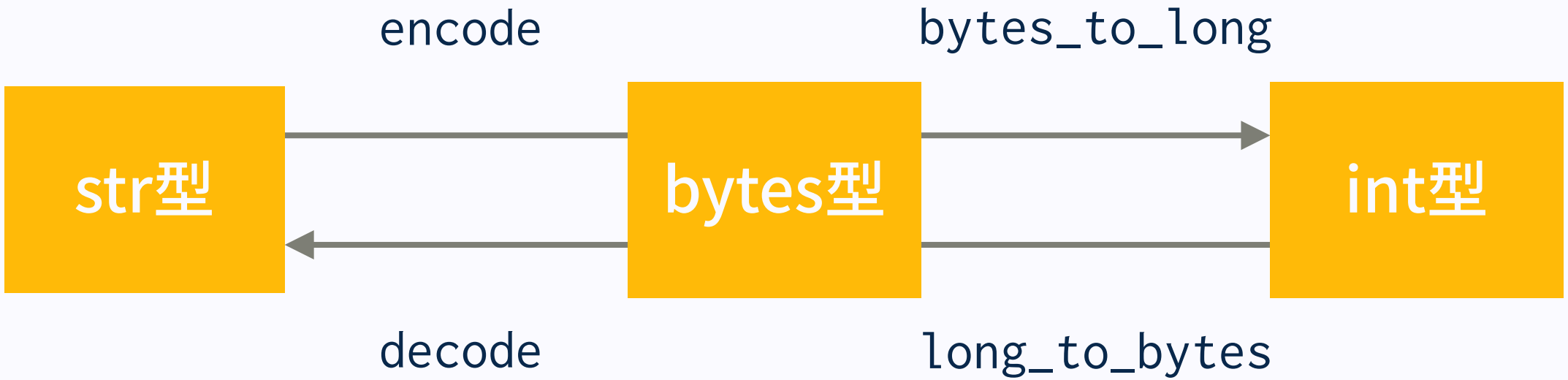
bytes型とint型の変換

```
>>> from Crypto.Util.number import *
>>> flag = “Hello, World!”
```

```
>>> bytes_to_long(flag.encode())
573581676307385491820377514908 9
```

```
>>> long_to_bytes(573581676307385491820377514908 9)
b'Hello, World!'
```

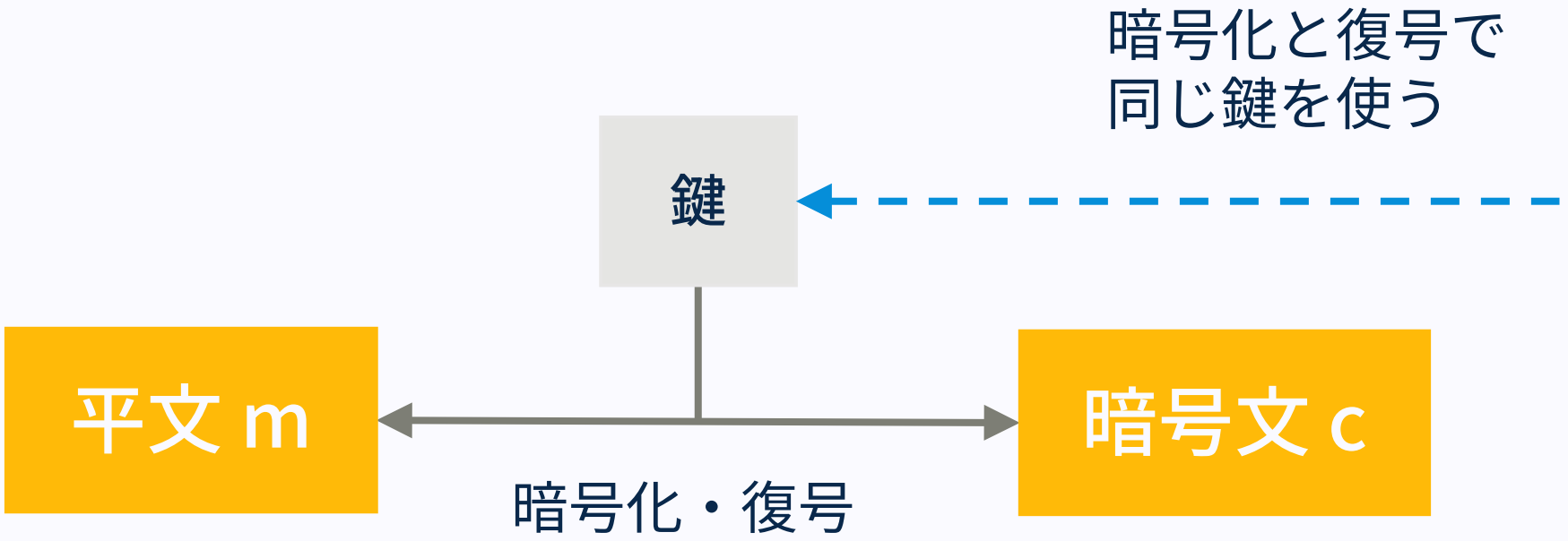
変換のまとめ



RSA暗号の仕組み

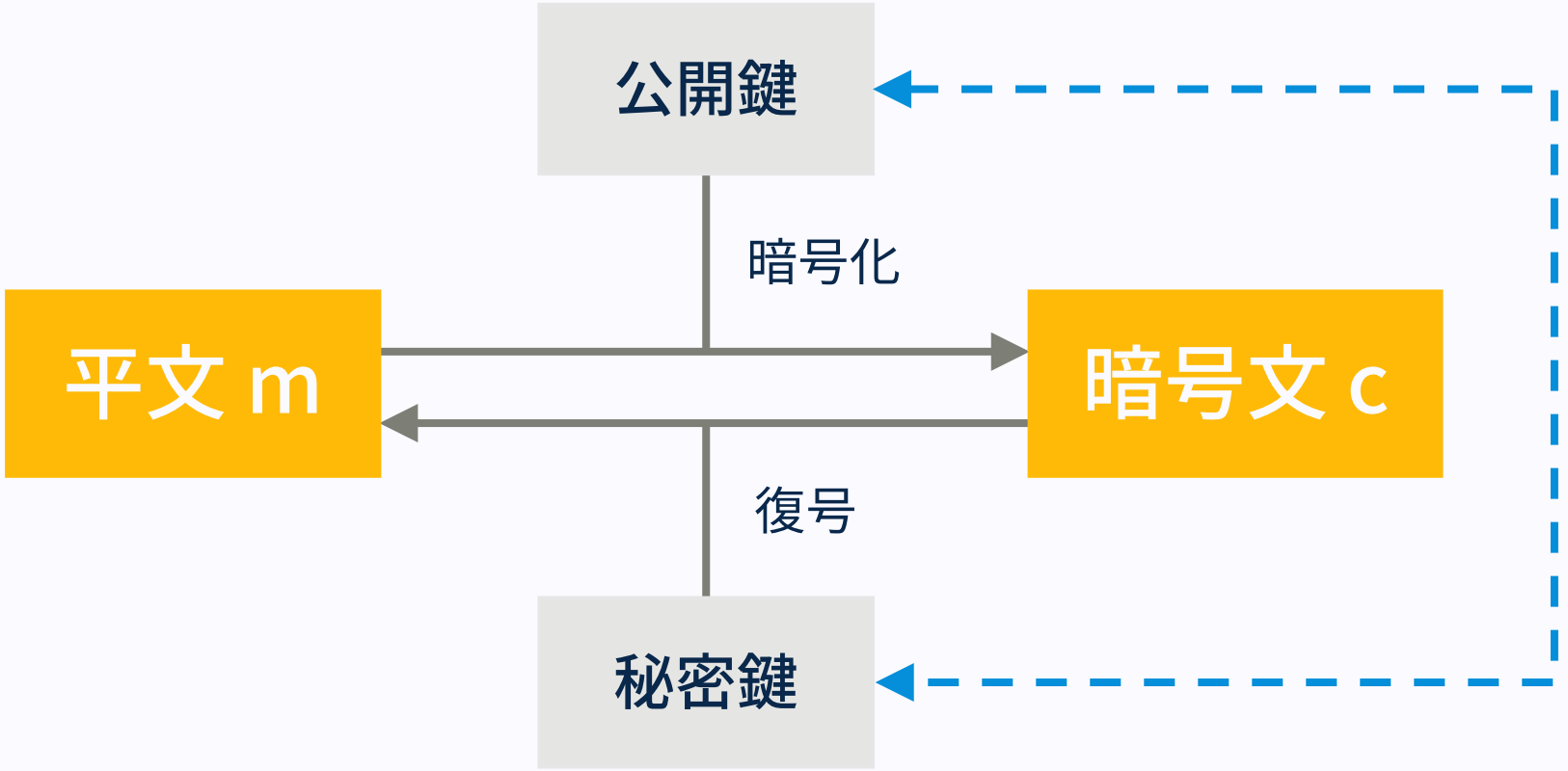
共通鍵暗号

ブロック暗号 (AES, DES) やストリーム暗号 (RC4) など



公開鍵暗号

RSA, ElGamal, 楕円ElGamalなど



暗号化と復号で
異なる鍵を使う

RSA暗号

平文 m，暗号文 c，公開鍵 (e, n)，秘密鍵 (d, n)

- 暗号化

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

- 復号

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

鍵の作り方

① 大きな2つの素数 p, q を生成する

乱数生成＋素数判定（ミラーラビンテスト）を繰り返す.
具体的には 1024 ～ 2048 bit 程度.

② $\mathrm{n} = pq$ を計算する

③ l を計算する

a と b の最大公約数 (Greatest Common Divisor),
最小公約数 (Least Common Multiple) を $\gcd(a,b), \mathrm{lcm}(a,b)$ とする.

$$l = \mathrm{lcm}(p-1, q-1) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)}$$

鍵の作り方

- ③の補足

資料によっては，$\mathrm{lcm}(p-1, q-1)$ を

$$\phi(n) = (p-1)(q-1)$$

としている場合がある．しかし，どちらを採用してもRSA暗号は成立する．

一般には $\mathrm{lcm}(p-1, q-1)$ のほうが $(p-1)(q-1)$ よりも小さくなるので，

$\mathrm{lcm}(p-1, q-1)$ が使われている．

詳しく知りたい人は「オイラーのトーシェント関数」で検索．

鍵の作り方

④ e を計算する

$1 < e < l$ かつ $\gcd(e, l) = 1$ を満たす e を求める．

大きすぎても小さすぎてもダメ．

$e = 2^{16} + 1 = 65537$ が推奨されている．

⑤ d を計算する

$1 < d < l$ かつ $ed \equiv 1 \pmod{l}$ を満たす d を求める．

これは拡張ユークリッドの互除法で見つけられる．

RSA暗号の関連資料

- 結城浩，『暗号技術入門 第3版 秘密の国のアリス』
- IPUSIRON，『暗号技術のすべて』
- 光成滋生，『クラウドを支えるこれからの暗号技術』
  https://herumi.github.io/ango/
- kurenaif，【CTF入門】RSA暗号を実装/解読する 1 【Crypto】(訂正はコメント参照！)
  https://www.youtube.com/watch?v=HKDyUELFdXs
- sonickun，RSA暗号運用でやってはいけない n のこと #ssmjp
  https://www.slideshare.net/sonickun/rsa-n-ssmjp
- ふるつき，CTF crypto 逆引き
  https://furutsuki.hatenablog.com/entry/2021/03/16/095021

RSA暗号の関連資料

- ももいろテクノロジー，plain RSAに対する攻撃手法を実装してみる
  http://inaz2.hatenablog.com/entry/2016/01/15/011138
- ももいろテクノロジー，SageMathを使ってCoppersmith's Attackをやってみる
  http://inaz2.hatenablog.com/entry/2016/01/20/022936
- A painter and a black cat，CTF Crypto
  https://raintrees.net/projects/a-painter-and-a-black-cat/wiki/CTF_Crypto
- 0xDktb，Summary of Crypto in CTF(RSA)
  https://0xdktb.top/2020/02/28/Summary-of-Crypto-in-CTF-RSA/
- RsaCtfTool
  https://github.com/Ganapati/RsaCtfTool

[Beginner] simple_RSA

ファイル構成

- `problem.py`

  Pythonのプログラム

- `output.txt`

  `problem.py`の実行結果

output.txt

公開鍵と暗号文が書かれている.

```
n = 176866718424003935747305120342001285213369195697359727916766O…

e = 3

c = 213791751530017111508691084168363024686878057337971319880256 9…
```

problem.py

```python
from Crypto.Util.number import *
from flag import flag

flag = bytes_to_long(flag.encode("utf-8"))

p = getPrime(1024)
q = getPrime(1024)
n = p * q
e = 3

assert 2046 < n.bit_length()
assert 375 == flag.bit_length()

print("n =", n)
print("e =", e)
print("c =", pow(flag, e, n))
```

実際には存在しない
ダミーのライブラリ

e が小さすぎる

n は 2046 bit より大きい

`flag` は 375 bit

Low Public-Exponent Attack

$m^e < n$ のとき，c の e 乗根を取ることで m が求まる．

◆ **理由**

$m^e < n$ のとき， m^e は $\mathrm{mod}\, n$ の影響を受けず，

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

となるから．

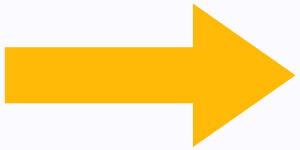
【例】$7 \mod 19 = 7$

Low Public-Exponent Attack

この問題では $e = 3$， n は 2047 bit 以上，`flag` は 375 bit なので，

$$\mathrm{flag}^e < n$$

だと予想できる．



Low Public-Exponent Attackでflagが出る！

方針

① output.txtから e, n, c の値を取ってくる

② c の e 乗根を計算する

③ 出てきた数値を文字列に変換する

Pythonでe乗根を計算する方法

gmpy2というライブラリを使用する.

インストールするためにはGMPの開発環境が必要.
Ubuntuだとaptを使うと上手くやってくれる.

```
$ sudo apt update
$ sudo apt install python3-gmpy2
```

Windows

① pipが対応しているwhlのバージョンを確認する

```
>>> from pip._internal.utils.compatibility_tags import get_supported
>>> print(get_supported())
```

② gmpy2のwhlファイルをダウンロードする

https://www.lfd.uci.edu/~gohlke/pythonlibs/#gmpy

```
$ pip install ./gmpy2-2.0.8-cp38-cp38-win_amd64.whl
```

iroot

```
>>> import gmpy2
>>> gmpy2.iroot(1024, 2)
(mpz(32), True)
>>> gmpy2.iroot(1024, 2)[0]
mpz(32)
```

返り値はタプル型なので
[0]を付けて取り出す.

```
>>> gmpy2.root(1024, 2)
mpfr('32.0')
```

irootとrootは別物.
rootは実数なので
精度が落ちる.

[Beginner] Logical_SEESAW

ファイル構成

- `problem.py`

  Pythonのプログラム

- `output.txt`

  `problem.py`の実行結果

output.txt

暗号文が16個書かれている.

```
cipher = ['11000010111…', '11000110110…', '11000010110…',
           …
          '11000110110…', '11000010101…', '11000110101…']
```

problem.py

```
from Crypto.Util.number import *
from random import random, getrandbits
from flag import flag

flag = bytes_to_long(flag.encode("utf-8"))
length = flag.bit_length()
key = getrandbits(length)
while not length == key.bit_length():
    key = getrandbits(length)

flag = list(bin(flag)[2:])
key = list(bin(key)[2:])
```

flagのビット長

flagと同じビット長のkeyを生成

flagとkeyを2進数の
リストに変換する

problem.py

```
cipher_L = []

for _ in range(16):
    cipher = flag[:]
    m = 0.5

    for i in range(length):
        n = random()
        if n > m:
            cipher[i] = str(eval(cipher[i] + "&" + key[i]))

    cipher_L.append("".join(cipher))


print("cipher =", cipher_L)
```

16個の暗号文を生成

50%の確率で
flag[i] & key[i]

暗号文のパターン

```
cipher[k][i] = flag[i]
               または flag[i] & key[i]
```

$$0 \leq k \leq 15$$
$$0 \leq i < \text{length}$$

flag[i]	key[i]	flag[i] & key[i]
0	0	0
0	1	0
1	0	0
1	1	1

→ cipher[k][i]が“1”なら，
flag[i]は”1”

暗号文のパターン

```
cipher[k][i] = flag[i]
               または flag[i] & key[i]
```

$$0 \leq k \leq 15$$
$$0 \leq i < \text{length}$$

flag[i]	key[i]	flag[i] & key[i]
0	0	0
0	1	0
1	0	0
1	1	1

すべてのkについて，
cipher[k][i]が“0”なら，
flag[i]は”0”

暗号文のパターン

```
cipher[k][i] = flag[i]
               または flag[i] & key[i]
```

$$0 \leq k \leq 15$$
$$0 \leq i < \text{length}$$

flag[i]	key[i]	flag[i] & key[i]
0	0	0
0	1	0
1	0	0
1	1	1

この場合もcipher[k][i]は“0”になるが，このパターンが16回連続で発生する確率はほぼゼロ

16回すべてAND演算がされる確率

- 1回だけ判定してAND演算がされる確率

$$\frac{1}{2}$$

- 16回すべてAND演算がされる確率

$$\left(\frac{1}{2}\right)^{16} = \frac{1}{65536} \cong 0.0015\ \%$$

滅多に起こらないから，
この場合は考えなくてもいい

方針

① output.txtからcipherを取ってくる

② 16個の暗号文の各ビットについて
一つでも“1”があれば，flagの対応するビットを”1”
すべて”0”なら，flagの対応するビットを”0”

③ flagを10進数に変換する

④ flagを文字列に変換する

[Easy] GFM

ファイル構成

- `problem.sage`

  SageMathのプログラム

- `output.txt`

  `problem.sage`の実行結果

SageMathとは

「Sageは，代数学，幾何学，数論，暗号理論，数値解析，および関連諸分野の研究と教育を支援する，フリーなオープンソース数学ソフトウェアである．」（公式ページより）

Pythonをベースに開発されているので，文法はPythonと大体同じ．

- オンラインの実行環境
  https://sagecell.sagemath.org/
- 公式ページ
  https://www.sagemath.org/
- 公式チュートリアル
  https://doc.sagemath.org/html/ja/tutorial/index.html

problem.sage

```
FLAG = b'<censored>'

SIZE = 8
p = random_prime(2 ^ 128)
MS = MatrixSpace(GF(p), SIZE)


key = MS.random_element()
while key.rank() != SIZE:
    key = MS.random_element()
```

位数 p の有限体上の数を
要素に持つ 8×8 の行列空間 MS

$\mathrm{rank}(key) = 8$
$\rightarrow key$ は正則行列（逆行列を持つ）

有限体

加算・減算・乗算・除算の四則演算が定義され，
閉じている有限集合を**有限体**といい，$GF(p)$ と表す．
発見者のガロアにちなんで，**ガロア体** (Galois Field) とも呼ぶ．

※「閉じている」… すべての演算結果が集合内に存在すること．

p は有限体に含まれる要素の数であり，**位数**と呼ばれる．

$$\underbrace{\{0, 1, 2, \ldots, p-2, p-1\}}_{p\text{ 個}}$$

problem.sage

```
M = copy(MS.zero())
for i in range(SIZE):
    for j in range(SIZE):
        n = i * SIZE + j
        if n < len(FLAG):
            M[i, j] = FLAG[n]
        else:
            M[i, j] = GF(p).random_element()

enc = key * M * key

print('p:', p)
print('key:', key)
print('enc:', enc)
```

行列 M にFLAGを1文字ずつ格納する

行列 M の空部分には乱数を格納する

output.txtに出力されている値

Mを取り出す

FLAGは M に格納されているので，enc から M を取り出したい．
key は逆行列 key^{-1} を持つので，

$$enc = key \times M \times key$$

$$M = key^{-1} \times enc \times key^{-1}$$

SageMathで逆行列を求めるためには，inverse()を使えばいい．

```
key_inv = key.inverse()
```

方針

① output.txtから p, key, enc の値を取ってくる

② key の逆行列 key^{-1} を計算する

③ enc の左右に key^{-1} をかけて M を取り出す

④ M からFLAGを取り出す

[Medium] Imaginary

ファイル構成

- app.py

  サーバ側のプログラム

- test.py

  配布されたapp.pyをローカル環境で動かせるように
  勝手に改造したプログラム

  ※ フラグを書いているので，中身は見ない方がいいかも

ローカル環境でサーバを動かす方法

test.pyを実行するだけ.
止めるときは"ctrl+c"を押す.

```
$ python3 test.py
Start server at localhost:1337
```

アプリの動作

ncで接続すると，コマンド一覧が表示される．

```
$ nc localhost 1337
Welcome to Secret IMAGINARY NUMBER Store!
1. Save a number
2. Show numbers
3. Import numbers
4. Export numbers
0. Exit
>
```

アプリの動作

1. 複素数を保存する

```
> 1
Real part> 1
Imaginary part> 3
```

2. 保存した複素数を表示する

```
> 2
------------------------------------------------
1 + 3i: (1, 3)
------------------------------------------------
```

アプリの動作

4. 保存した複素数のデータをjsonに変換し，
   それをAESのECBモードで暗号化する

```
> 4
{"1 + 3i": [1, 3]}
Exported:
0b63e3e740430cb46bf2a51f4cb29edf273788035f227f77074feb3fc52d6717
```

アプリの動作

3. AESのECBモードで暗号化されたjsonを受け取り，
   それを復号して変数に代入する

```
> 3
Exported String>
0b63e3e740430cb46bf2a51f4cb29edf273788035f227f77074feb3fc52d6717
Imported.
----------------------------------------------
1 + 3i: (1, 3)
----------------------------------------------
```

app.py

```
        while True:
            num = self._menu()
            if num == 1:
                self._save()
            elif num == 2:
                self._show()
            elif num == 3:
                self._import()
            elif num == 4:
                self._export()
            elif num == 5:
                self._secret()
            else:
                break
```

表示されていないコマンド5

FLAGの出力条件

self.numbersに'1337i'が含まれていたらFLAGを出力する.

```
    def _secret(self):
        if '1337i' in self.numbers:
            self.request.sendall(b'Congratulations!¥n')
            self.request.sendall(f'The flag is {flag}¥n'.encode())
```

self.numbers

keyに複素数，valueに実部・虚部のリストを持つ辞書．

```
        self.numbers = {}
```

```
        name = f'{re} + {im}i'
        self.numbers[name] = [re, im]
```

```
>>> numbers = {}
>>> re = 1
>>> im = 2
>>> name = f'{re} + {im}i'
>>> numbers[name] = [re, im]
>>> numbers
{'1 + 2i': [1, 2]}
```

self.numbers

実部に0を入れても，0 + 1337iと保存されてしまう．

```
> 1
Real part> 0
Imaginary part> 1337
1. Save a number
2. Show numbers
3. Import numbers
4. Export numbers
0. Exit
> 2
------------------------------------------------
0 + 1337i: (0, 1337)
------------------------------------------------
```

データの改ざん

そこで，Import / Export の機能に注目する．

① Exportでデータを出力する．
　このデータはAESのECBモードで暗号化されている．

② データを改ざんして，keyを’1337i’にする．

③ Importで改ざんしたデータを入力する．

AES

ブロック暗号（共通鍵暗号）の一種．

平文を1ブロック128ビットで分割し，それを暗号化する．
平文を128ビットで割り切れない場合は，乱数を追加して
対応する (Padding)．

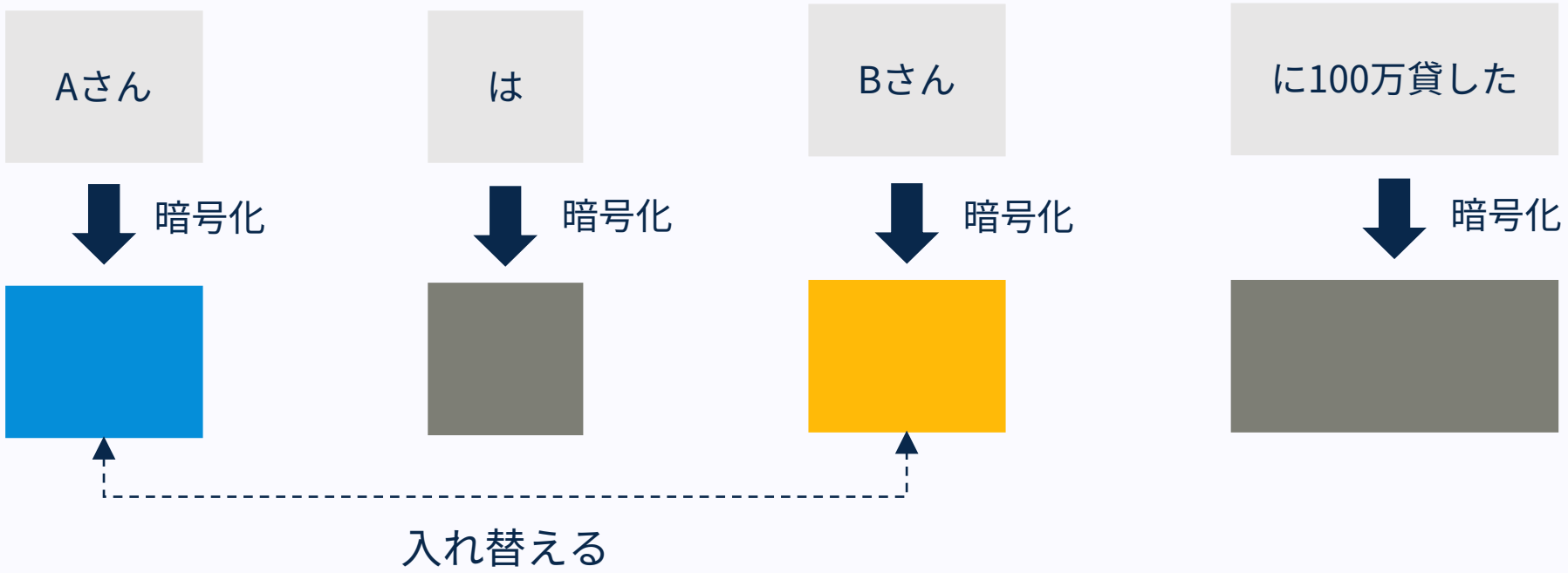
ECBモード

複数のブロックがある場合は暗号化を繰り返して行うことになる．
このときの繰り返し方をモードという．

最も単純な方法は，平文のブロックをそのまま暗号化することである．
これをECBモードという．

ECBモードの問題点

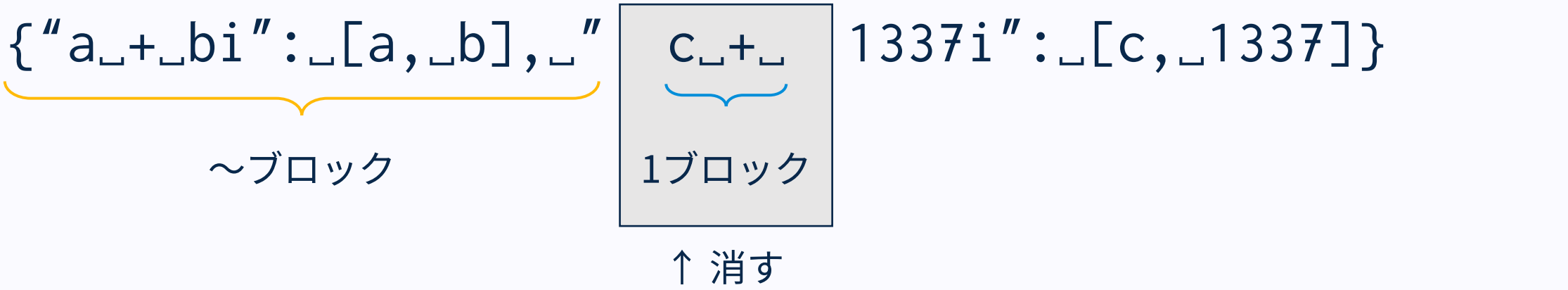
ECBモードでは，暗号文のブロックを入れ替えたり，削除したりしても復号ができてしまう．



「BさんはAさんに100万貸した」に改ざんできる

改ざんの方針

半角文字は8ビットなので，1ブロックは128÷8で16文字となる．
データのkeyに’1337i’が出てくるように上手く調整する．



{“a␣+␣bi”:␣[a,␣b],␣”1337i”:␣[c,␣1337]}

改ざん用データの作成

{“a␣+␣bi”:␣[a,␣b],␣”とc␣+␣を作る.

```
$ python3 data_1.py
{'1000 + 1000i': [1000, 1000], '
x: 1000
bytes: 32
```

```
$ python3 data_2.py
1000000000000 +
y: 1000000000000
bytes: 16
```

暗号文の入手

Exportを使って，暗号文を入手する．

```
> 4
{"1000 + 1000i": [1000, 1000], "1000000000000 + 1337i":
[1000000000000, 1337]}
Exported:
71295e3edd417b3a18be48eb67b129508c7a47ce644b2402003bdd4e8c50510167d8
c505c310236544b96267806d5fed097af578edba148ead016696be713cff95c68a47
aabef5095c32c686008db357
```

不要なブロックの削除

100000000000␣+␣のブロックを削除し，
必要なブロックだけを取り出す．

```
$ python3 clip.py
71295e3edd417b3a18be48eb67b129508c7a47ce644b2402003bdd4e8c505101097a
f578edba148ead016696be713cff95c68a47aabef5095c32c686008db357
```

FLAG

改ざんしたデータをImportし，5でFLAGを表示する．

```
> 5
Congratulations!
The flag is
ctf4b{yeah_you_are_a_member_of_imaginary_number_club}
```

Cryptoの作問者Writeup

- @ushigai_sub，SECCON Beginners CTF2021 作問者Writeup
  https://qiita.com/ushigai_sub/items/8c63fb566f19ac097bc5
- rex.gs，SECCON Beginners CTF 2021 Crypto 解説
  https://rex.gs/2021/05/seccon-beginners-ctf-2021-crypto-%E8%A7%A3%E8%AA%AC/

LLLの関連資料

- 國廣 昇，格子理論を用いた暗号解読の最近の研究動向
  https://www.jstage.jst.go.jp/article/essfr/5/1/5_1_42/_pdf
- うさぎ小屋，PlaidCTF CTF 2015: Lazy
  https://kimiyuki.net/writeups/ctf-2015-plaidctf-2015-lazy/
- katagaitai workshop #7 crypto ナップサック暗号と低密度攻撃
  https://www.slideshare.net/trmr105/katagaitai-workshop-7-crypto
- Xornet，LLLでCrypto問題を解く
  https://project-euphoria.dev/blog/10-lll-for-crypto/
- ptr-yudai，LLLで殴る【yoshi-camp 2020 Spring備忘録】
  https://ptr-yudai.hatenablog.com/entry/2020/03/12/223338

p-8RSAの関連資料

- y011d4.log，p - 1 ≡ 0 (mod e) のときの RSA 復号方法
  https://y011d4.netlify.app/20201026-not-coprime-e-phi/
- How to compute m value from RSA if phi(n) is not relative prime with the e?
  https://crypto.stackexchange.com/questions/81949/how-to-compute-m-value-from-rsa-if-phin-is-not-relative-prime-with-the-e
- 0xDktb，Summary of Crypto in CTF(RSA)
  https://0xdktb.top/2020/02/28/Summary-of-Crypto-in-CTF-RSA/
- RSA Risk: When e and PHI share the same factor (and are not co-prime)
  https://asecuritysite.com/encryption/rsa_prob