[Capture The Flag]

NAMA TIM : [Ainge CTF]

Minggu, 31 Oktober 2021

Ketua Tim	
1.	Wiwit Rifa'i
Member	
1.	Afrizal Fikri
2.	Luqman Arifin Siswanto

Table of Content

Crypto

Shuvvler

Diberikan dua buah file: *flag.html.enc* dan *chall.py*.

flag.html.enc adalah file hasil enkripsi yang dilakukan oleh *chall.py* dengan substitusi dan shift karakter-karakter pada *flag.html*. Untuk melakukan dekripsi, kita cukup perlu membuat fungsi inverse dari substitusi dan shift yang digunakan, dan membalikan prosesnya dari belakang. Lalu, kita juga bisa bruteforce key yang digunakan karena banyaknya key yang mungkin sedikit.

Berikut adalah script solver yang digunakan.

File: solver.py

```
#!/usr/bin/env python3
import random, string, sys

chars = string.ascii_lowercase + string.ascii_uppercase + string.digits +
string.punctuation

def inv_func1(s, key):
    r = ''
    for i in range(len(s)):
        if s[i] in chars:
            if   s[i] in string.ascii_lowercase: z = string.ascii_lowercase
            elif s[i] in string.ascii_uppercase: z = string.ascii_uppercase
            elif s[i] in string.digits:          z = string.digits
            else:                                z = string.punctuation
            r += z[(z.index(s[i]) + len(z) - key) % len(z)]
        else:
            r += s[i]
    return r

def inv_func2(s):
    return s.translate(s.maketrans(chars[13:] + chars[:13], chars))

def decrypt(s, key):
    r = s
    for _ in range(8):
        r = inv_func2(r)
        r = inv_func1(r, key)
    return r

def main():
    fns = sys.argv[1:]
    for fn in fns:
        try:
            x = open(fn + '.enc', 'r').read()
            f = open(fn, 'w')
            key = 0
            s = ''
            while True:
                s = x
                s = decrypt(x, key)
```

```
                if 'html' in s:
                    print(key, 'found!!')
                    break
                else:
                    print(key, 'not found')
                key += 1
            f.write(s)
            f.close()
        except:
            continue

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print(f'Usage: {sys.argv[0]} <file>')
        print(f'Example: {sys.argv[0]} flag.html')
        sys.exit()
    main()
```

Flag: **CJ2021{Substitut3_4nd_sh1ft_555}**

RSASS

Diberikan 2 buah file: *chall.py* dan *output.txt*.

output.txt berisi hasil output dari *chall.py* yang menyatakan beberapa hasil enkripsi RSA dari *flag.txt* dengan nilai eksponen e yang berbeda. Karena nilai GCD dari semua nilai e yang digunakan bernilai 1 yaitu GCD(e_1, e_2, …, e_k) = 1, maka dengan bezout's identity (linear diophantine equation) dijamin terdapat nilai- nilai (x_1, x_2, …, x_k) sedemikian sehingga $e_1 * x_1 + e_2 * x_2 + \ldots + e_k * x_k = GCD(e_1, e_2, \ldots, e_k)$. Dengan begitu, kita bisa membuat eksponen dari cipher menjadi 1 yang menyatakan nilai flag yang asli.

Kita dapat menggunakan algoritma extended euclid untuk mencari nilai-nilai dari (x1, x2) sedemikian sehingga $e_1 * x_1 + e_2 * x_2 = GCD(e_1, e_2)$. Lalu, kita proses hasil tersebut dengan eksponen-eksponen lain hingga GCD dari semua eksponen menjadi 1 dan kita dapatkan flag aslinya.

Berikut adalah script solver yang digunakan.

File: solver.py

```
from Crypto.Util.number import *
import sys
sys.setrecursionlimit(5000)

def ext_euclid(a, b):
  if a == 0:
    return b, 0, 1
  d, x, y = ext_euclid(b % a, a)
  return d, y - (b // a) * x, x

def modinv(x, m):
  d, x, y = ext_euclid(x, m)
  assert d == 1
  return x % m

def attack(e1, c1, e2, c2, n):
  d, x, y = ext_euclid(e1, e2)
  if x < 0:
    c1 = modinv(c1, n)
    x = -x
  if y < 0:
    c2 = modinv(c2, n)
    y = -y
  return d, pow(c1, x, n) * pow(c2, y, n) % n

base = 1337
points = []
mul = 1
with open('output.txt', 'r') as f:
  n = int(f.readline().strip())
  for line in f.readlines():
    p, m = map(int, line.strip()[1:-1].split(', '))
```

```
    p += base
    mul *= p
    points.append((p, m))

d = mul // points[0][0]
m = points[0][1]
for p, en in points[1:]:
  d, m = attack(d, m, mul // p, en, n)

print(long_to_bytes(m).decode())
```

Flag: **CJ2021{Wir_wollen__dass_jeder_hier_im_Raum_mitmacht__ja_ea8f5dc}**

Baby Next

Diberikan 2 buah file: *chall.py* dan *output.txt*.

output.txt berisi nilai n dan hasil enkripsi flag dengan RSA. Pembangkitan nilai n dilakukan dengan membalikkan bit-bit selain dari 2 bit pada most significant bit (MSB) dari sebuah prima, lalu mencari prima selanjutnya yang cukup dekat dengan nilai tersebut, dan nilai n merupakan hasil dari perkalian 2 prima tersebut.

Akibat dari proses tersebut, 2 prima yang dihasilkan tersebut memiliki sifat yang bisa dieksploitasi yaitu jumlah kedua prima yang bisa di estimasi yaitu $p + q = 2^{512} + 2^{256} + x$ atau $p + q = 2^{513} + x$ untuk suatu nilai x yang tidak terlalu besar. Sehingga, kita bisa melakukan bruteforce untuk mencari nilai x dan melakukan faktorisasi fermat.

Berikut adalah script solver yang digunakan.

File: solver.py

```
from sympy import integer_nthroot
from Crypto.Util.number import *

n =
999117102530646641689175613927307347748802569956357425800031824410902409777880995
667281303979741605242856383436653140386004139801788083730927901203309177823796476
774781061130058348109143664186676932737133722561127310278622829267912089703265470
28034342408090623409495840320518060489050699358411485867698775327
c =
548387746168208944872882071136466722963387846170858760718916177116753339936383787
808524397732282642578791700105358721516875220858477439158489426424951826743968035
500286752060209152812187528464228264540313278678096333107751687597197185204739828
29736399275265775059744057417184036965152456278717799755158576900

def fermat_factorize(key, offset=0):
    limit=1000000
    a =
0x18000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000 // 2
    max = a + limit
    solutions = []
    while a < max:
        b2 = a*a - key
        if b2 >= 0:
            b = integer_nthroot(b2,2)[0]
            if b*b == b2:
                return a + b, a - b
        a += 1
    return -1, -1

p, q = fermat_factorize(n)

e = 0x65537
```

```
euler = (p - 1) * (q - 1)
d = inverse(e, euler)
m = pow(c, d, n)

print(long_to_bytes(m))
```

Flag: **CJ2021{ahh__another_p_plus_q_RSA_challenge_8367}**

Burve

Diberikan 2 buah file: *chall.py* dan *output*.

output berisi nilai enkripsi dari sebuah *text.txt* menggunakan semacam enkripsi yang menggunakan Elliptic Curve. Setiap karakter pada *text.txt* dienkripsi menjadi sebuah point pada EC yaitu (key + i^2 + t^2) * g dengan g adalah sebuah base point pada EC, key adalah key yang tidak diketahui, i adalah posisi karakter pada text, dan t adalah nilai karakter tersebut.

Kita tidak bisa menginverse dari sebuah point p untuk mencari nilai x sedemikian sehingga p = x * g, namun kita tetap masih bisa mencari selisih dari 2 point, yaitu jika p = x * g dan q = y * g maka kita bisa mencoba menambah p dengan g terus menerus sampai nilainya sama dengan q, atau sebaliknya dengan menambah q dengan g terus menerus sampai nilainya sama dengan p. Dengan begitu, kita bisa mencari nilai selisih dari x-y tersebut.

Dengan cara tersebut, kita bisa mengestimasi nilai perbedaan antar karakter pada text, lalu kita bisa menentukan semua karakter jika kita sudah yakin dengan nilai dari suatu karakter.

Berikut adalah script solver yang digunakan.

File: solver.py

```
#!/usr/bin/env python3
from libnum.ecc import Curve as Burve

burve = Burve(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc,
0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b,
0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
boint = (0x7cf27b188d034f7e8a52380304b51ac3c08969e277f21b35a60b48fc47669978,
0x07775510db8ed040293d9ac69f7430dbba7dade63ce982299e04b79d227873d1)
assert burve.check(boint)
burve.g = boint

out = open('output', 'rb').read()
sz = len(out)


diffs = dict()

for i in range(256):
    for j in range(256):
        d = j * j - i * i
        if d not in diffs:
            diffs[d] = [(j, i)]
        else:
            diffs[d].append((j, i))

print(len(diffs))

text = []
base = None
```

```
def check(p, q, diff):
    if diff < 0:
        q = burve.add(q, burve.generate(-diff))
    else:
        p = burve.add(p, burve.generate(diff))
    return p == q
f = open('text.txt', 'w')

for i in range(sz // 32):
    x = int.from_bytes(out[32 * i:32 * (i+1)], 'big')
    p = list(tuple(burve.check_x(x)))
    k = None
    if not base:
        base = p[1]
        k = 36
    else:
        for cur in range(128):
            t = cur * cur - text[0] * text[0]
            d = i * i + t
            for c in p:
                if check(base, c, d):
                    k = cur
                    break
            if k:
                break
    if k:
        print(i, 'found', k)
        text.append(k)
        f.write(chr(k))
    else:
        print(i, 'not found')

result = ''.join(map(chr, text))
print(result)
```

Flag: **CJ2021{new_b4se_b0int_on_the_Burve_6bcb0e8d0538}**

Glückstag

Diberikan 1 file *server.py* dan sebuah service `nc 159.223.87.165 33301`.

Berdasarkan *server.py*, diketahui bahwa service tersebut akan menerima passphrase dan akan menggunakannya sebagai bagian dari parameter untuk enkripsi flag dengan AES. Berikut adalah beberapa hal yang terjadi dalam enkripsi:

1. Key dibangkitkan berdasarkan prefix + passphrase untuk sebuah konstan prefix yang rahasia.
2. IV dibangkitkan dengan menambahkan random pad di depan dan di belakan dari prefix.
3. Panjang dari prefix + passphrase mempengaruhi apakah key akan diswap dengan IV, dan apakah IV akan diswap dengan block pertama message.
4. Enkripsi sebuah blok dilakukan dengan melakukan xor blok tersebut dengan hasil enkripsi AES ECB dari hasil cipher pada blok sebelumnya (atau IV jika ini adalah blok pertama).

Hal pertama yang ingin kita lakukan adalah mencari panjang dari prefix dan nilai dari prefix itu sendiri.

Kita bisa melakukan bruteforce terhadap panjang dari passphrase, sedemikian sehingga kita ingin agar key dan IV tidak diswap, namun IV akan diswap dengan block pertama flag. Tujuannya adalah agar blok pertama berisi prefix yang dixor dengan hasil enkripsi dari IV yang bernilai konstan. Dengan begitu, jika posisi prefix-nya sama pada blok tersebut, maka nilai-nilai pada posisi prefix tersebut akan tetap sama, sedangkan nilai-nilai sisanya yang merupakan padding akan berubah-ubah.

Kita bisa mengumpulkan sejumlah cipher flag untuk passphrase yang sama tersebut. Lalu kita cek apakah ada contiguous substring yang sama pada blok pertama antar 2 cipher dengan panjang > 2 (karena panjang prefix > 2). Jika ditemukan, berarti posisi prefix dari 2 cipher tersebut terletak pada posisi yang sama, dan kita juga bisa tau panjang dari prefix tersebut.

Setelah itu, kita kumpulkan cipher-cipher dengan posisi prefix yang berbeda-beda. Lalu, kita bisa mengetahui nilai xor dari antar karakter pada prefix. Sisanya adalah kita perlu bruteforce salah satu karakter dari prefix sehingga karakter sisanya bisa mengikuti berdasarkan nilai xor antar karakter tersebut. Dengan begitu, terdapat 256 kemungkinan nilai prefix.

Untuk mengetahui nilai prefix mana yang benar, kita bisa mencoba menggunakan masing-masing prefix dan mencoba untuk mendekripsi nilai IV yang sekarang berisi blok pertama dari flag yang berisi CJ2021. Jika ditemukan, maka kita berhasil mendapatkan blok pertama dari flag. Lalu, blok-blok sisanya bisa kita dapatkan dengan mendekripsi blok-blok yang lain dengan menggunakan prefix dan passphrase yang tepat.

Berikut adalah script solver yang digunakan.

File: solver.py

```
from Crypto.Cipher import AES
from Crypto.Hash import MD5
import binascii, os, random, sys
from Crypto.Util.number import *
from pwn import *

context.log_level = 'error'

def get_enc(passphrase):
```

```
  r = remote('159.223.87.165', 33301)
  r.sendlineafter(b'>> ', passphrase)
  r.recvuntil(b'<< ')
  res = r.recvline()
  return res.strip()

xor = lambda a, b: bytes([i ^ j for i, j in zip(a, b)])

def longest_common(a, b):
  best = 0
  pos = 0
  len_a = len(a)
  len_b = len(b)
  for i in range(len(a)):
    k = 0
    while i + k < len_a and i + k < len_b and a[i+k] == b[i+k]:
      k += 1
    if best < k:
      best = k
      pos = i
  return pos, best

pref_all = dict()
origins = dict()
pref_len = 0

for i in range(4):
  passphrase = b'a' * (i + 1)
  print('Trying passphrase:', passphrase)
  zero = b'\00' * 16
  entries = []
  limit = 40

  while len(pref_all) <= 16 - pref_len and (limit > 0 or pref_len > 0):
    limit -= 1
    cur = bytes.fromhex(get_enc(passphrase).decode())
    for enc in entries:
      pos, common = longest_common(cur[:16], enc[:16])
      if common > 2:
        if pref_len == 0:
          pref_len = common
        if pref_len == common:
          print('- Found prefix: at', pos, '=', enc[pos : pos + common])
          pref_all[pos] = cur[pos : pos + common]
          origins[pos] = cur
    entries.append(cur)
  if pref_len > 0:
    break
```

```
pref_xor = xor(pref_all[0][1:], pref_all[1][:-1])

for a in range(256):
  c = a
  prefix = [c]
  for x in pref_xor:
    c ^= x
    prefix.append(c)
  prefix = bytes(prefix)

  crcc = int.to_bytes(binascii.crc32(prefix + passphrase), 4, 'big')
  key = MD5.new(prefix + crcc).digest()
  cipher = AES.new(key, AES.MODE_ECB)
  block = [-1 for _ in range(16)]
  for k, v in origins.items():
    for i in range(len(prefix)):
      c = prefix[i] ^ v[i + k]
      block[i + k] = c
  block = bytes(block)

  result = cipher.decrypt(block)
  if b'CJ' in result:
    flag = result
    result = bytearray(origins[0])
    for i in range(len(result)-2 * AES.block_size, -1, -AES.block_size):
        result[i + AES.block_size:i + 2*AES.block_size] = xor(
            result[i + AES.block_size:i + 2*AES.block_size],
            cipher.encrypt(result[i:i + AES.block_size])
        )
    result = result[AES.block_size:]
    flag += result
    print('Flag:', flag.decode())
```

Flag: **CJ2021{sowry_we_dont_speak_CBC_here_0f6b3ee991c}**

Reversing

crac

Kita hanya diberikan sebuah binary executable *crac* yang menerima sebuah input string dan menghasilkan output “Correct” (yang ini akan kita ketahui setelah menganalisis binary) atau “Incorrect”. Step pertama yang kita lakukan yaitu decompile binary memakai tools Ghidra. Dari situ kita mendapatkan isi dari fungsi main seperti di bawah.

main function

```
undefined8 main(void) {
  ...

  local_10 = *(long *)(in_FS_OFFSET + 0x28);

  // Verification table
  local_1c8[0] = 0x77cde91a;
  local_1c8[1] = 0xccd2022f;
  local_1c8[2] = 0x608aeb79;
  local_1c8[3] = 0xfac5bd9d;
  local_1b8 = 0x8e7ea6e7;
  local_1b4 = 0x2d7a767a;
  local_1b0 = 0xb371dfac;
  local_1ac = 0x712ab43b;
  local_1a8 = 0x35d048d6;
  local_1a4 = 0xf634d202;
  local_1a0 = 0x9a34568c;
  local_19c = 0xa904ff9a;
  local_198 = 0x83e7c438;
  local_194 = 0x620b9d61;
  local_190 = 0x5773afa9;
  local_18c = 0x664458b7;
  local_188 = 0xa43dd41;
  local_184 = 0xcee971dc;
  local_180 = 0xa9cc367a;
  local_17c = 0x6345ffb2;
  local_178 = 0x3ae7e1f9;
  local_174 = 0x2c90f1f8;
  local_170 = 0x3b619b06;
  local_16c = 0x2690dd53;
  local_168 = 0x1bf8ef9b;
  local_164 = 0x2ac998a9;
  local_160 = 0x2031aaf1;
  local_15c = 0x69e0f85;
  local_158 = 0xaa9d940f;
  local_154 = 0x3712e66b;
  local_150 = 0x45e38f79;
```

```
  local_14c = 0x3b4ba1c1;
  local_148 = 0x196f7aea;
  local_144 = 0xf4523f88;
  local_140 = 0xadcc08c0;
  local_13c = 0xaab631e6;
  local_138 = 0x6f5bc681;
  local_134 = 0x7e2c6f8f;
  local_130 = 0x256b08cc;
  local_12c = 0xe5cf91f8;
  local_128 = 0xe2b9bb1e;
  local_124 = 0xa75f4fb9;
  local_120 = 0x65182448;
  local_11c = 0x48948c8;

  puts("crac?");
  __isoc99_scanf(&DAT_00100b7a,local_118);
  sVar2 = strlen(local_118);
  if ((int)sVar2 == 0x2c) {  // 44
    local_1d8 = 0;
    while (local_1d8 < 0x2c) {
      local_1d0 = 0;
      local_1d4 = 0;
      while (local_1d4 <= local_1d8) {
        iVar1 = crac(local_118 + local_1d4,1,local_118);
        local_1d0 = local_1d0 + iVar1;
        local_1d4 = local_1d4 + 1;
      }
      if (local_1c8[local_1d8] != local_1d0) {
        puts("Incorrect!");
        goto LAB_00100ad0;
      }
      local_1d8 = local_1d8 + 1;
    }
    puts("Correct!");
  }
  else {
    puts("Incorrect!");
  }
LAB_00100ad0:
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Beberapa poin yang bisa dilihat di sini:

1. string input harus memiliki panjang 44
2. string akan dihitung menggunakan fungsi *crac*
3. fungsi *crac* dipanggil dengan argumen kedua yang selalu sama, yaitu 1
4. hasil dari pemanggilan *crac* pada sebuah prefix akan dijumlahkan dan dicek menggunakan tabel verifikasi yang di-initiate sebelumnya
5. jika hasil ada yang tidak cocok maka akan mengeluarkan "Incorrect", selain itu maka mengeluarkan "Correct"
6. tidak ada pemanggilan pada local file maupun remote socket

Berdasarkan poin 6 di atas, bisa kita tebak bahwa flag ada dalam string itu sendiri. Maka, kita buat script C untuk mensimulasikan call function dan mengetes apakah pengecekan pada prefix flag "CJ2021{" berjalan. Hasilnya ternyata benar sehingga bisa dipastikan input itu sendiri adalah flagnya dan tinggal kita cari isi flagnya.

Kemudian kita buka isi fungsi crac dan kita temukan akses ke crc32table sehingga bisa ditebak bahwa ini adalah fungsi menghitung crc32. Kita coba mencari sebagian isi crc32table di G**gle dan mendapati bahwa crc32 bergantung pada polinom yang digunakan. Dari situ kita juga dapat isi lengkap tabel crc32 yang polinomnya sama dengan yang kita gunakan, yaitu 0x04C11DB7 di situs berikut.

crac function

```
ulong crac(char *s,int len)

{
  char *ptr = s;
  int i = 0;
  uint crc = 0xffffffff;

  while (i < len) {
    crc = *(uint *)(crc32table + (ulong)(((int)*ptr ^ crc >> 0x18) & 0xff) * 4) ^
              crc << 8;
    i = i + 1;
    ptr = ptr + 1;
  }
  return (ulong)crc;
}
```

Perhatikan poin observasi 3 di atas. Karena fungsi hanya dipanggil dengan length 1, maka bagian pengulangan dapat dihapus. Sehingga fungsi *crac* hanya menghitung crc dari satu karakter. Pada poin 4 juga juga terlihat hasil bahwa tabel adalah *prefix sum* crc untuk tiap huruf. Kita bisa membuat tabel *reverse index* untuk nilai crc tiap karakter yang mungkin (alfanumerik plus underscore) dan mencari karakter yang cocok sesuai nilai crc dari tabel. Nilai crc dapat didapat dengan mudah dengan mengurangkan *sum* dengan *previous sum* (index sebelumnya). Snippet code dapat dilihat di bawah.

rev.cpp

```cpp
#include <cstdio>
#include <cstring>
#include <map>

using namespace std;

unsigned tab[] = {
  0x77cde91a,0xccd2022f,0x608aeb79,0xfac5bd9d,0x8e7ea6e7,
  0x2d7a767a,0xb371dfac,0x712ab43b,0x35d048d6,0xf634d202,
  0x9a34568c,0xa904ff9a,0x83e7c438,0x620b9d61,0x5773afa9,
  0x664458b7,0xa43dd41,0xcee971dc,0xa9cc367a,0x6345ffb2,
  0x3ae7e1f9,0x2c90f1f8,0x3b619b06,0x2690dd53,0x1bf8ef9b,
  0x2ac998a9,0x2031aaf1,0x69e0f85,0xaa9d940f,0x3712e66b,
  0x45e38f79,0x3b4ba1c1,0x196f7aea,0xf4523f88,0xadcc08c0,
  0xaab631e6,0x6f5bc681,0x7e2c6f8f,0x256b08cc,0xe5cf91f8,
  0xe2b9bb1e,0xa75f4fb9,0x65182448,0x48948c8
};

// source https://github.com/mariusae/pingrf/blob/master/libpump/crc32.c
unsigned crc32[] = {
      0x00000000, 0x04C11DB7, 0x09823B6E, 0x0D4326D9,
  0x130476DC, 0x17C56B6B, 0x1A864DB2, 0x1E475005,
  0x2608EDB8, 0x22C9F00F, 0x2F8AD6D6, 0x2B4BCB61,
  0x350C9B64, 0x31CD86D3, 0x3C8EA00A, 0x384FBDBD,
  0x4C11DB70, 0x48D0C6C7, 0x4593E01E, 0x4152FDA9,
      0x5F15ADAC, 0x5BD4B01B, 0x569796C2, 0x52568B75,
      0x6A1936C8, 0x6ED82B7F, 0x639B0DA6, 0x675A1011,
      0x791D4014, 0x7DDC5DA3, 0x709F7B7A, 0x745E66CD,
      0x9823B6E0, 0x9CE2AB57, 0x91A18D8E, 0x95609039,
      0x8B27C03C, 0x8FE6DD8B, 0x82A5FB52, 0x8664E6E5,
      0xBE2B5B58, 0xBAEA46EF, 0xB7A96036, 0xB3687D81,
      0xAD2F2D84, 0xA9EE3033, 0xA4AD16EA, 0xA06C0B5D,
      0xD4326D90, 0xD0F37027, 0xDDB056FE, 0xD9714B49,
      0xC7361B4C, 0xC3F706FB, 0xCEB42022, 0xCA753D95,
      0xF23A8028, 0xF6FB9D9F, 0xFBB8BB46, 0xFF79A6F1,
      0xE13EF6F4, 0xE5FFEB43, 0xE8BCCD9A, 0xEC7DD02D,
      0x34867077, 0x30476DC0, 0x3D044B19, 0x39C556AE,
      0x278206AB, 0x23431B1C, 0x2E003DC5, 0x2AC12072,
      0x128E9DCF, 0x164F8078, 0x1B0CA6A1, 0x1FCDBB16,
      0x018AEB13, 0x054BF6A4, 0x0808D07D, 0x0CC9CDCA,
      0x7897AB07, 0x7C56B6B0, 0x71159069, 0x75D48DDE,
      0x6B93DDDB, 0x6F52C06C, 0x6211E6B5, 0x66D0FB02,
```

```
    0x5E9F46BF, 0x5A5E5B08, 0x571D7DD1, 0x53DC6066,
    0x4D9B3063, 0x495A2DD4, 0x44190B0D, 0x40D816BA,
    0xACA5C697, 0xA864DB20, 0xA527FDF9, 0xA1E6E04E,
    0xBFA1B04B, 0xBB60ADFC, 0xB6238B25, 0xB2E29692,
    0x8AAD2B2F, 0x8E6C3698, 0x832F1041, 0x87EE0DF6,
    0x99A95DF3, 0x9D684044, 0x902B669D, 0x94EA7B2A,
    0xE0B41DE7, 0xE4750050, 0xE9362689, 0xEDF73B3E,
    0xF3B06B3B, 0xF771768C, 0xFA325055, 0xFEF34DE2,
    0xC6BCF05F, 0xC27DEDE8, 0xCF3ECB31, 0xCBFFD686,
    0xD5B88683, 0xD1799B34, 0xDC3ABDED, 0xD8FBA05A,
    0x690CE0EE, 0x6DCDFD59, 0x608EDB80, 0x644FC637,
    0x7A089632, 0x7EC98B85, 0x738AAD5C, 0x774BB0EB,
    0x4F040D56, 0x4BC510E1, 0x46863638, 0x42472B8F,
    0x5C007B8A, 0x58C1663D, 0x558240E4, 0x51435D53,
    0x251D3B9E, 0x21DC2629, 0x2C9F00F0, 0x285E1D47,
    0x36194D42, 0x32D850F5, 0x3F9B762C, 0x3B5A6B9B,
    0x0315D626, 0x07D4CB91, 0x0A97ED48, 0x0E56F0FF,
    0x1011A0FA, 0x14D0BD4D, 0x19939B94, 0x1D528623,
    0xF12F560E, 0xF5EE4BB9, 0xF8AD6D60, 0xFC6C70D7,
    0xE22B20D2, 0xE6EA3D65, 0xEBA91BBC, 0xEF68060B,
    0xD727BBB6, 0xD3E6A601, 0xDEA580D8, 0xDA649D6F,
    0xC423CD6A, 0xC0E2D0DD, 0xCDA1F604, 0xC960EBB3,
    0xBD3E8D7E, 0xB9FF90C9, 0xB4BCB610, 0xB07DABA7,
    0xAE3AFBA2, 0xAAFBE615, 0xA7B8C0CC, 0xA379DD7B,
    0x9B3660C6, 0x9FF77D71, 0x92B45BA8, 0x9675461F,
    0x8832161A, 0x8CF30BAD, 0x81B02D74, 0x857130C3,
    0x5D8A9099, 0x594B8D2E, 0x5408ABF7, 0x50C9B640,
    0x4E8EE645, 0x4A4FFBF2, 0x470CDD2B, 0x43CDC09C,
    0x7B827D21, 0x7F436096, 0x7200464F, 0x76C15BF8,
    0x68860BFD, 0x6C47164A, 0x61043093, 0x65C52D24,
    0x119B4BE9, 0x155A565E, 0x18197087, 0x1CD86D30,
    0x029F3D35, 0x065E2082, 0x0B1D065B, 0x0FDC1BEC,
    0x3793A651, 0x3352BBE6, 0x3E119D3F, 0x3AD08088,
    0x2497D08D, 0x2056CD3A, 0x2D15EBE3, 0x29D4F654,
    0xC5A92679, 0xC1683BCE, 0xCC2B1D17, 0xC8EA00A0,
    0xD6AD50A5, 0xD26C4D12, 0xDF2F6BCB, 0xDBEE767C,
    0xE3A1CBC1, 0xE760D676, 0xEA23F0AF, 0xEEE2ED18,
    0xF0A5BD1D, 0xF464A0AA, 0xF9278673, 0xFDE69BC4,
    0x89B8FD09, 0x8D79E0BE, 0x803AC667, 0x84FBDBD0,
    0x9ABC8BD5, 0x9E7D9662, 0x933EB0BB, 0x97FFAD0C,
    0xAFB010B1, 0xAB710D06, 0xA6322BDF, 0xA2F33668,
    0xBCB4666D, 0xB8757BDA, 0xB5365D03, 0xB1F740B4
};
```

```
unsigned crc32_calc(char data) {
      return (0xffffffff << 8) ^ crc32[((0xffffffff >> 24) ^ data) & 255];
}

int main() {
  char S[46];
  unsigned sum = 0, i = 0;
  S[0x2c] = '\0';
  char K[] = "qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890_";
  int m = strlen(K);

  map<unsigned, char> mp;   // reverse index
  for (auto c: K) {
    mp[crc32_calc(c)] = c;
  }

  while (i < 0x2c) {      // 44
    S[i] = mp[tab[i] - sum];
    sum = tab[i];
    i++;
  }

  printf("%s\n", S);
  return 0;
}
```

Flag: **CJ2021{this_one_should_be_easy_enough_right}**