How 2 Random

Hank Chen

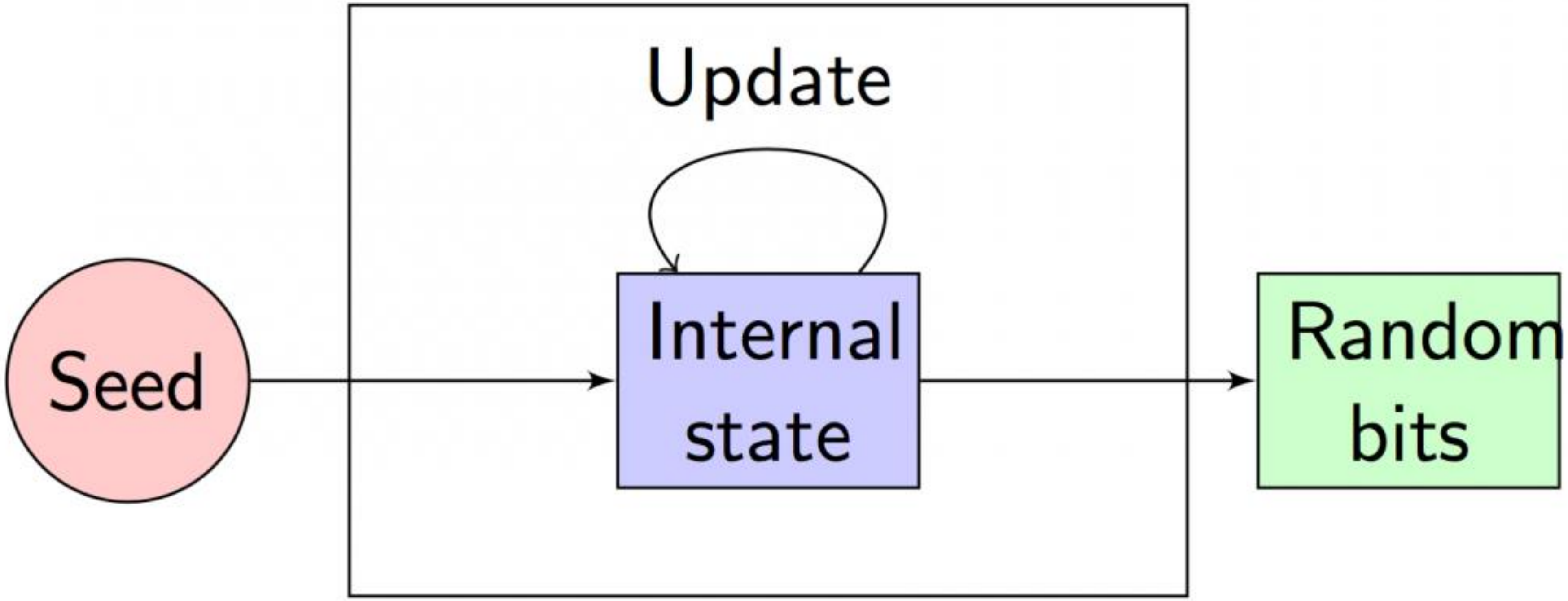
Outline

- What Is PRNG?
- Is Random Enough?
- How 2 Random?

What Is PRNG?

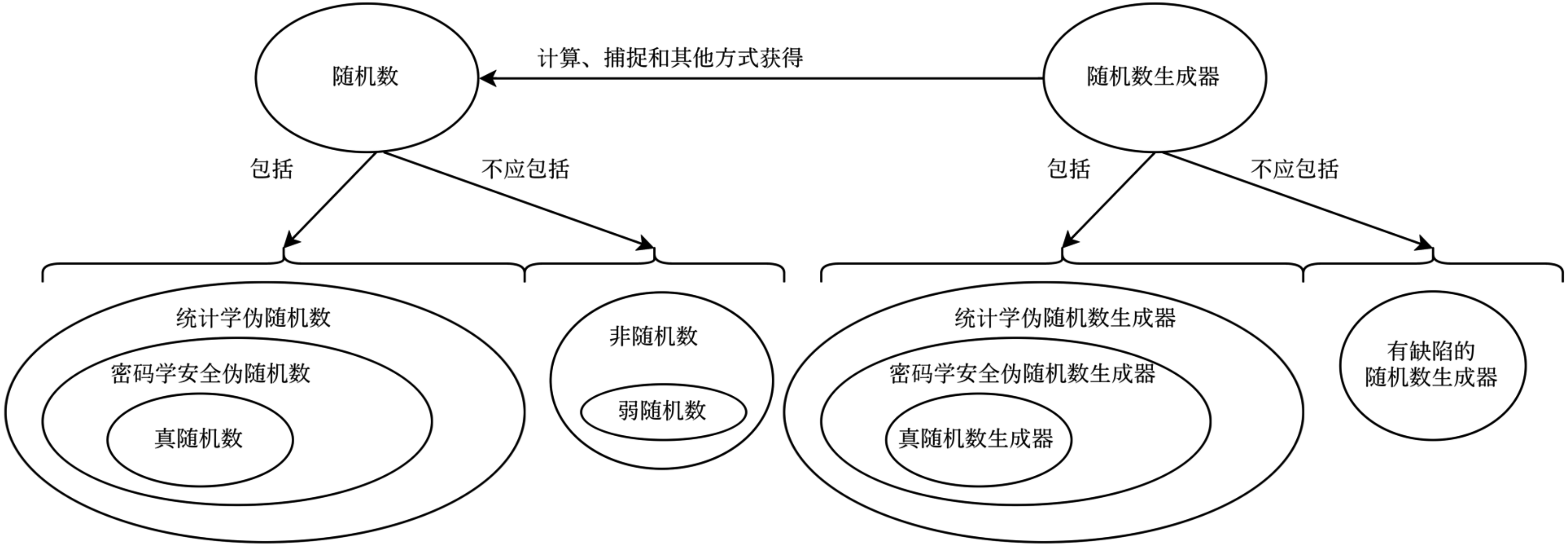
PRNG (亂數產生器)

Definition (fixed security parameter version): A (t, ϵ)-PRNG is a function



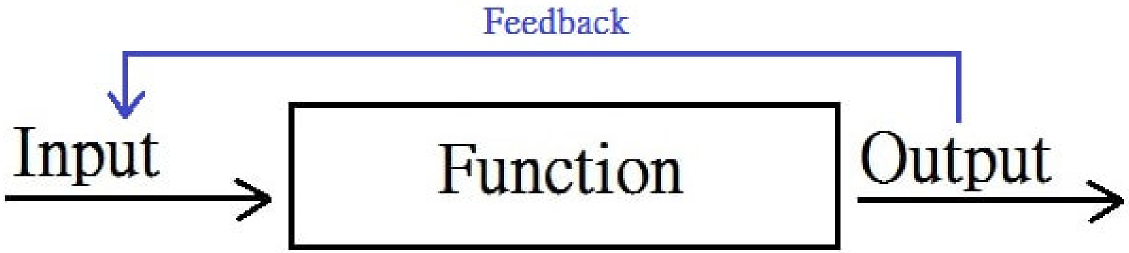
Classification

- CSPRNG (Cryptographic Secure Pseudo Random Number Generator)
- TRNG (True Random Number Generator)



CSPRNG (密碼學安全亂數產生器)

- One way function
  - 過去簡單，反推很難
  - Eg. Discrete Logrithm Problem
- Cipher based
  - CTR mode, IV = seed
  - Stream cipher, IV = seed
- Hash based
  - x_{i+1}= x_i+1; o_{i+1}=$H(x_i)$



TRNG (真•亂數產生器)

- Definition
  - 不是電腦程式來生成亂數的裝置
  - eg. 石英共振、噪音
  - 用來生成PRNG的seed
- Product
  - TPM (Trusted Platform module)

System Random

- Application (Linux)
  - /dev/urandom
  - /dev/random
  - system call getrandom()
    - /dev/random : 512 bytes
    - /dev/urandom : 33554431 bytes
  - /dev/hwrng

Is Random Enough?

Algorithm

- LCG
- XOR shift
- LFSR
- MT19937

Linear congruential generator (LCG)

定義

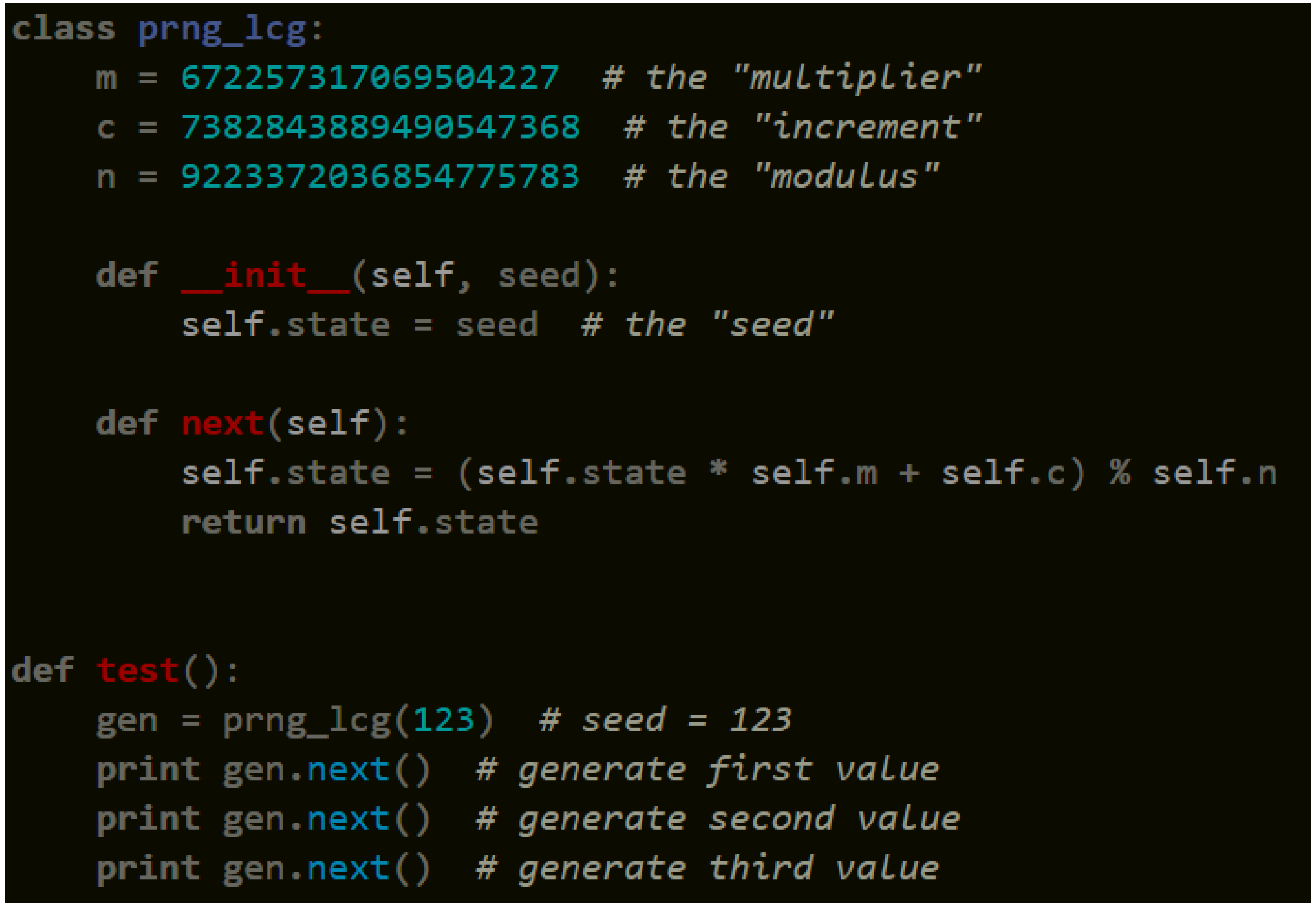
- $x_{i+1} = ax_i + b \mod m$

變形

- Raw： $x_{i-1} = a^{-1}\,(x_i - b) \mod m$
- 保留 Low order bits：估上下界
- 保留 High order bits：LLL reduction

Lab1 – Crack LCG

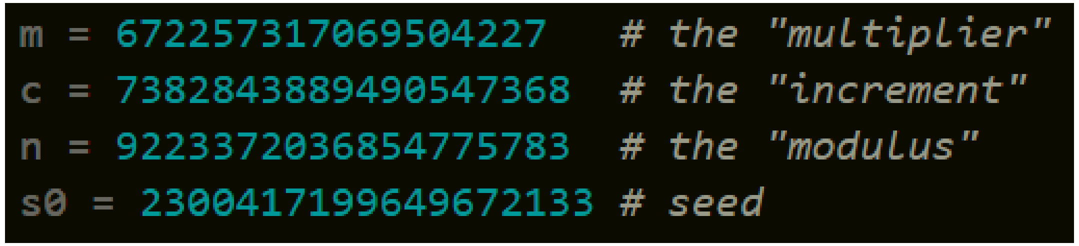
Crack LCG



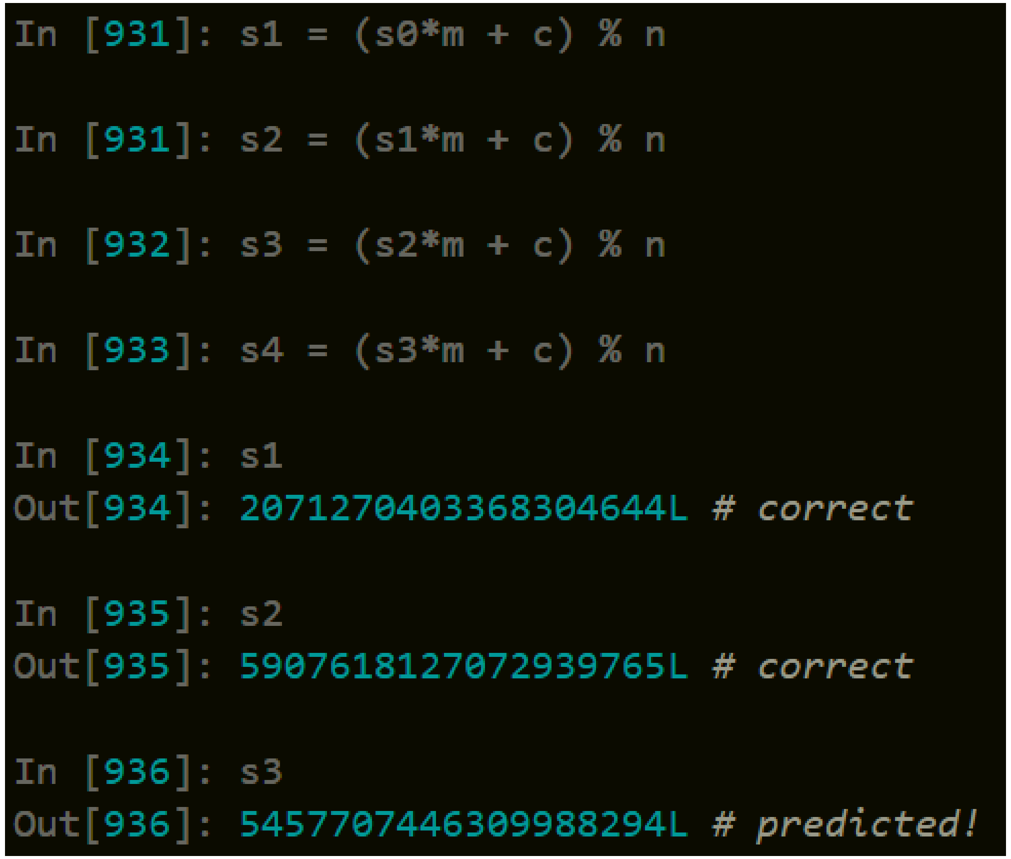
```
class prng_lcg:
    m = 672257317069504227   # the "multiplier"
    c = 7382843889490547368  # the "increment"
    n = 9223372036854775783  # the "modulus"

    def __init__(self, seed):
        self.state = seed  # the "seed"

    def next(self):
        self.state = (self.state * self.m + self.c) % self.n
        return self.state


def test():
    gen = prng_lcg(123)  # seed = 123
    print gen.next()  # generate first value
    print gen.next()  # generate second value
    print gen.next()  # generate third value
```

Crack LCG – Level 1

- Everything Known

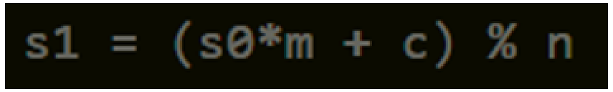


```
m = 672257317069504227   # the "multiplier"
c = 7382843889490547368  # the "increment"
n = 9223372036854775783  # the "modulus"
s0 = 2300417199649672133 # seed
```



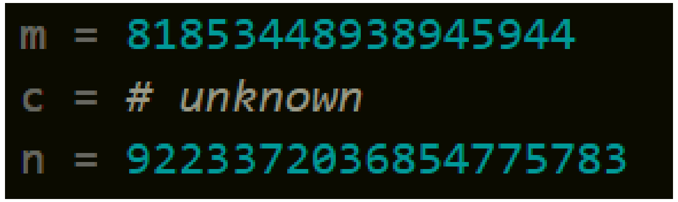
```
In [931]: s1 = (s0*m + c) % n

In [931]: s2 = (s1*m + c) % n

In [932]: s3 = (s2*m + c) % n

In [933]: s4 = (s3*m + c) % n

In [934]: s1
Out[934]: 2071270403368304644L # correct

In [935]: s2
Out[935]: 5907618127072939765L # correct

In [936]: s3
Out[936]: 5457707446309988294L # predicted!
```

Crack LCG – Level 2

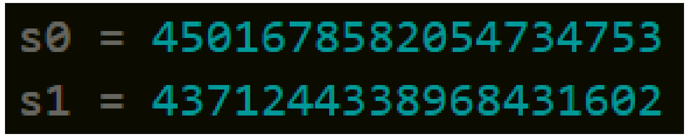


```
s1 = (s0*m + c) % n
```

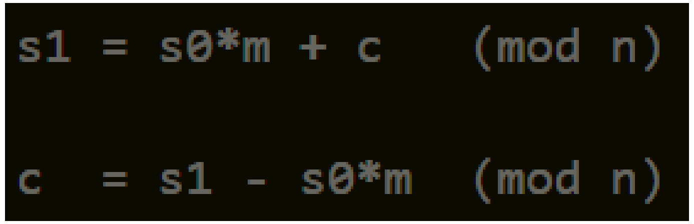
- Unknown increment



```
m = 81853448938945944
c = # unknown
n = 9223372036854775783
```

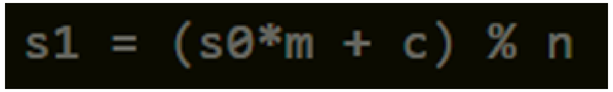


```
s0 = 4501678582054734753
s1 = 4371244338968431602
```



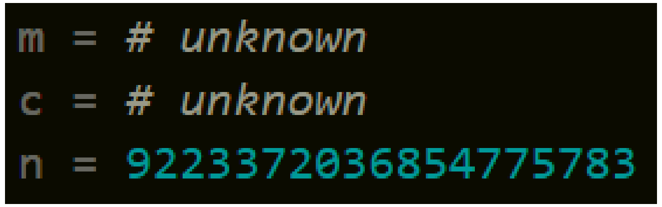
```
s1 = s0*m + c    (mod n)

c  = s1 - s0*m   (mod n)
```

Crack LCG – Level 3

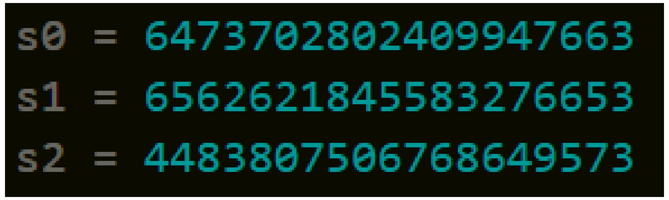


```
s1 = (s0*m + c) % n
```

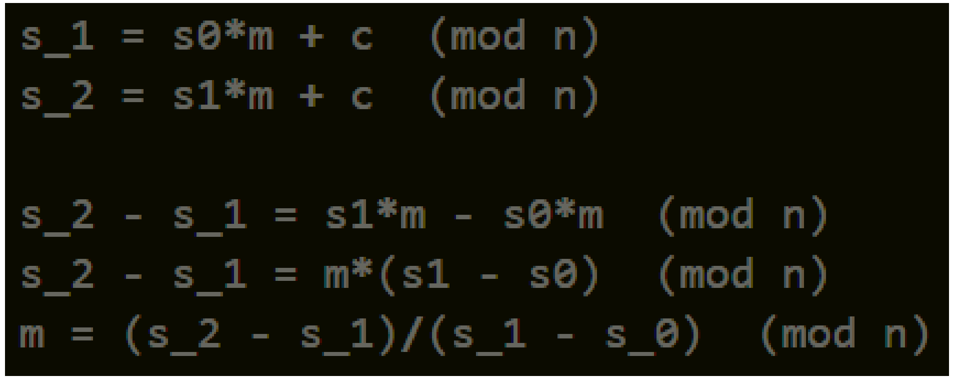
- unknown increment and multiplier



```
m = # unknown
c = # unknown
n = 9223372036854775783
```



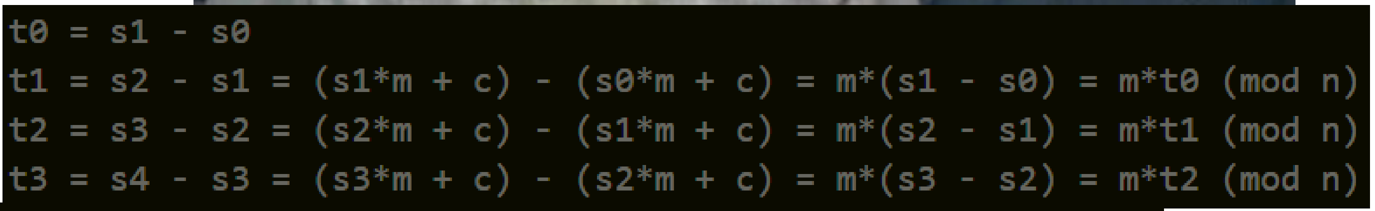
```
s0 = 6473702802409947663
s1 = 6562621845583276653
s2 = 4483807506768649573
```



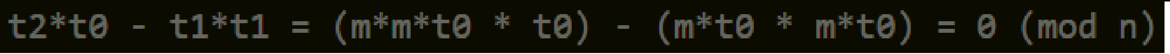
```
s_1 = s0*m + c  (mod n)
s_2 = s1*m + c  (mod n)

s_2 - s_1 = s1*m - s0*m  (mod n)
s_2 - s_1 = m*(s1 - s0)  (mod n)
m = (s_2 - s_1)/(s_1 - s_0)  (mod n)
```

Crack LCG – Level 4

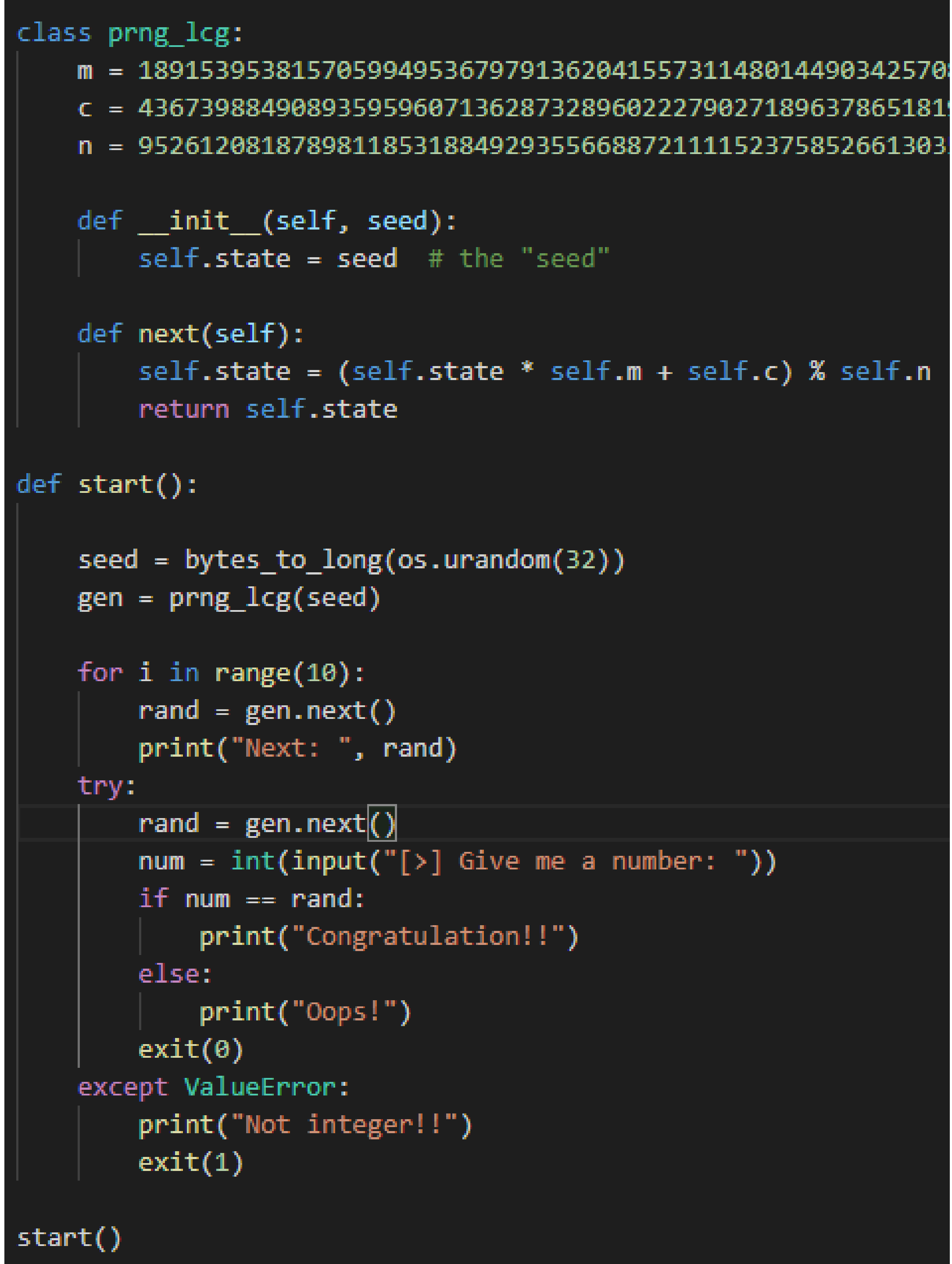
- **nc 140.114.77.172 60001**



```
t0 = s1 - s0
t1 = s2 - s1 = (s1*m + c) - (s0*m + c) = m*(s1 - s0) = m*t0 (mod n)
t2 = s3 - s2 = (s2*m + c) - (s1*m + c) = m*(s2 - s1) = m*t1 (mod n)
t3 = s4 - s3 = (s3*m + c) - (s2*m + c) = m*(s3 - s2) = m*t2 (mod n)
```



```
t2*t0 - t1*t1 = (m*m*t0 * t0) - (m*t0 * m*t0) = 0 (mod n)
```



```python
class prng_lcg:
    m = 18915395381570599495367979136204155731148014490342570
    c = 43673988490893595960713628732896022279027189637865181
    n = 95261208187898118531884929355668872111152375852661303

    def __init__(self, seed):
        self.state = seed  # the "seed"

    def next(self):
        self.state = (self.state * self.m + self.c) % self.n
        return self.state

def start():

    seed = bytes_to_long(os.urandom(32))
    gen = prng_lcg(seed)

    for i in range(10):
        rand = gen.next()
        print("Next: ", rand)
    try:
        rand = gen.next()
        num = int(input("[>] Give me a number: "))
        if num == rand:
            print("Congratulation!!")
        else:
            print("Oops!")
        exit(0)
    except ValueError:
        print("Not integer!!")
        exit(1)

start()
```

XORShift

運算速度快，程式碼簡單

定義

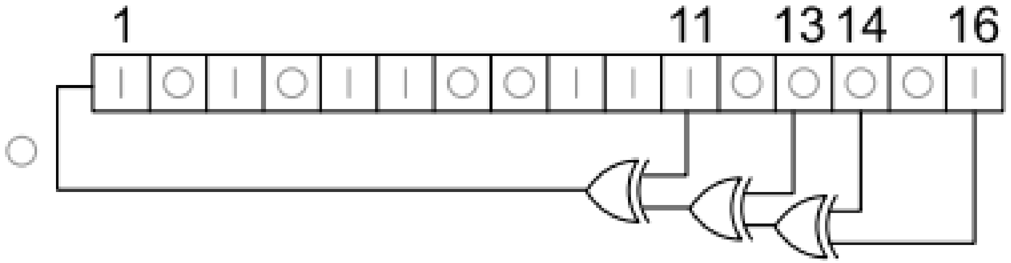
- $x \oplus= (x \ll a);\ \ x \oplus= (x \gg b);\ \ x \oplus= (x \ll c);$

分析技巧

- 把 x 看成 bit vector，XOR 和 Shift 可以寫成 GF(2) 下的矩陣乘法

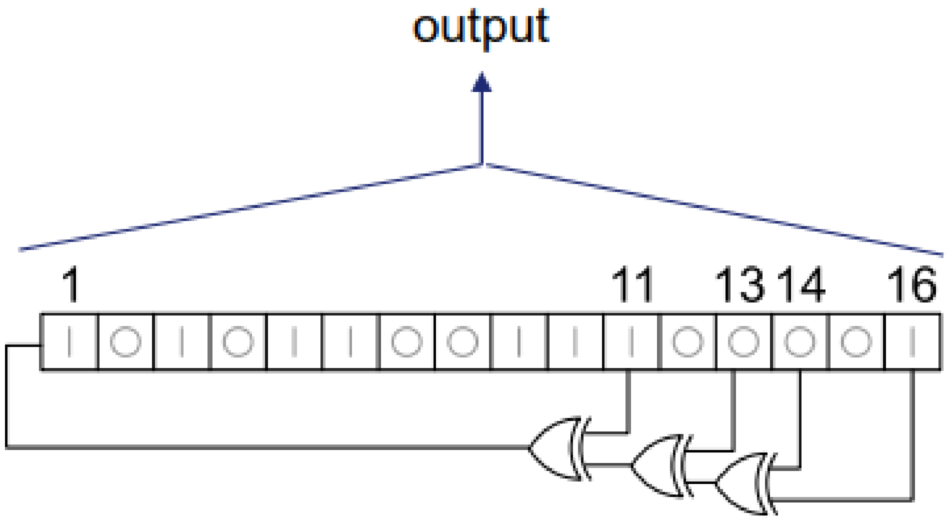
LFSR

- 暫存器的初始值是 seed
- 一次輸出一個 bit
    - 輸出的 bit 會變成第一個暫存器的值
- 分析技巧
    - 跟 XORShift 一樣用 bit vector 和 GF(2) 矩陣來處裡



Filtered LFSR

- 從輸出第一個暫存器改成對整個state做非線性運算
  - $s_{i+1} = \mathrm{LFSR}(s_i);\quad o = f(s_{i+1})$
- 分析方法
  - 找 f 的 線性 (或low degree) annihilator g
    - $\forall x : f(x) = 1 \text{,} \ g(x) = 0$
  - 對於所有 $f(s_i) = 1$ 的地方用 $g(s_i)$ 建聯立方程式並求解



Mersenne Twister (MT19937)

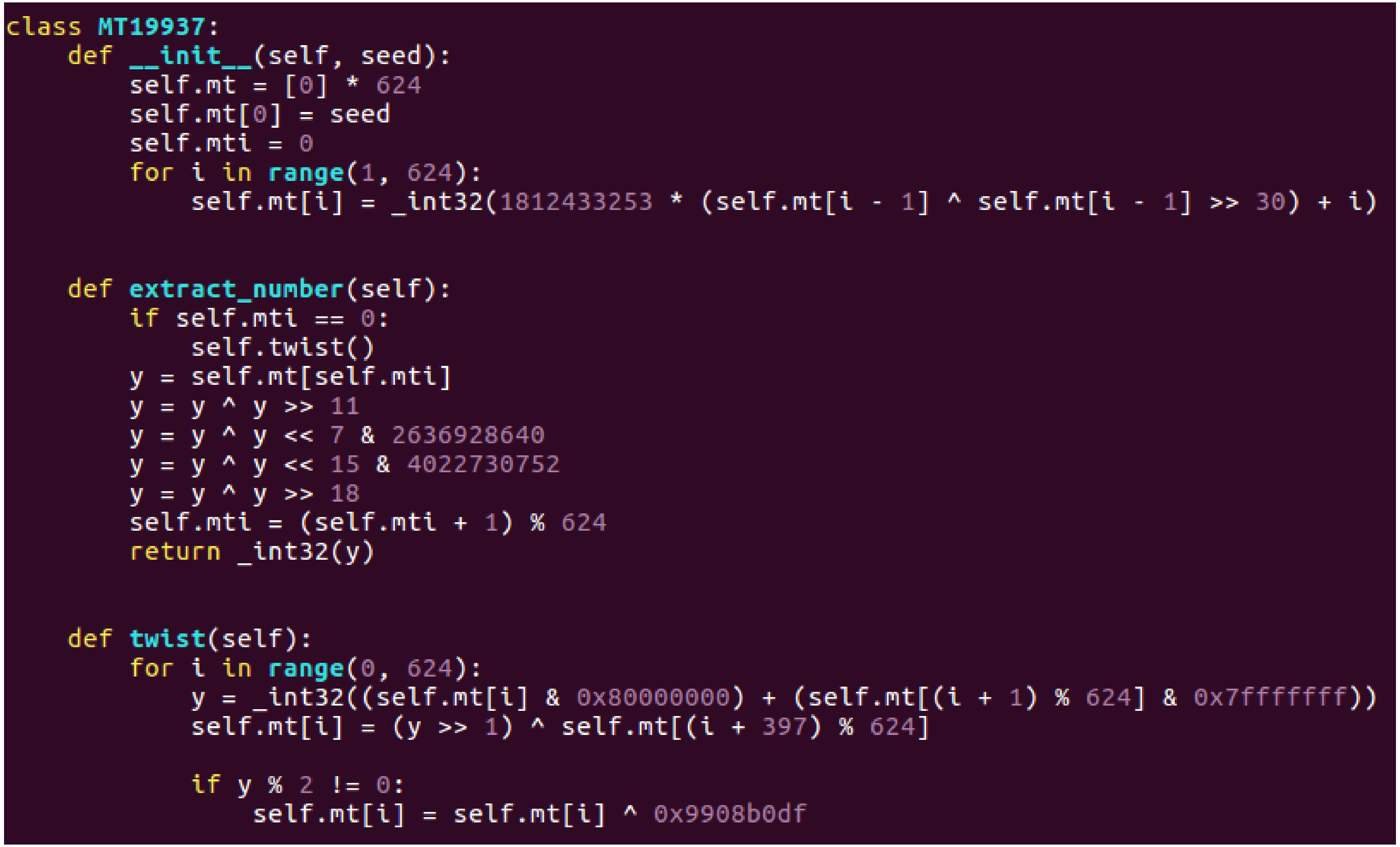
- 週期長、沒專利、通過很多測試…
- 許多語言的預設算法：MATLAB, PHP, Python, R, Ruby, Octave…

- 狀態為 624 個 32-bits 的數字
- x_i 由 x_{i-624}，x_{i-623}，x_{i-227} 三個數字運算而成

- 輸出前會先經過一個可逆的非線性函數
  - 可以由連續 624 個 32bits 的輸出還原出狀態
  - 可以由足夠多獨立的 bits 解方程式還原出狀態

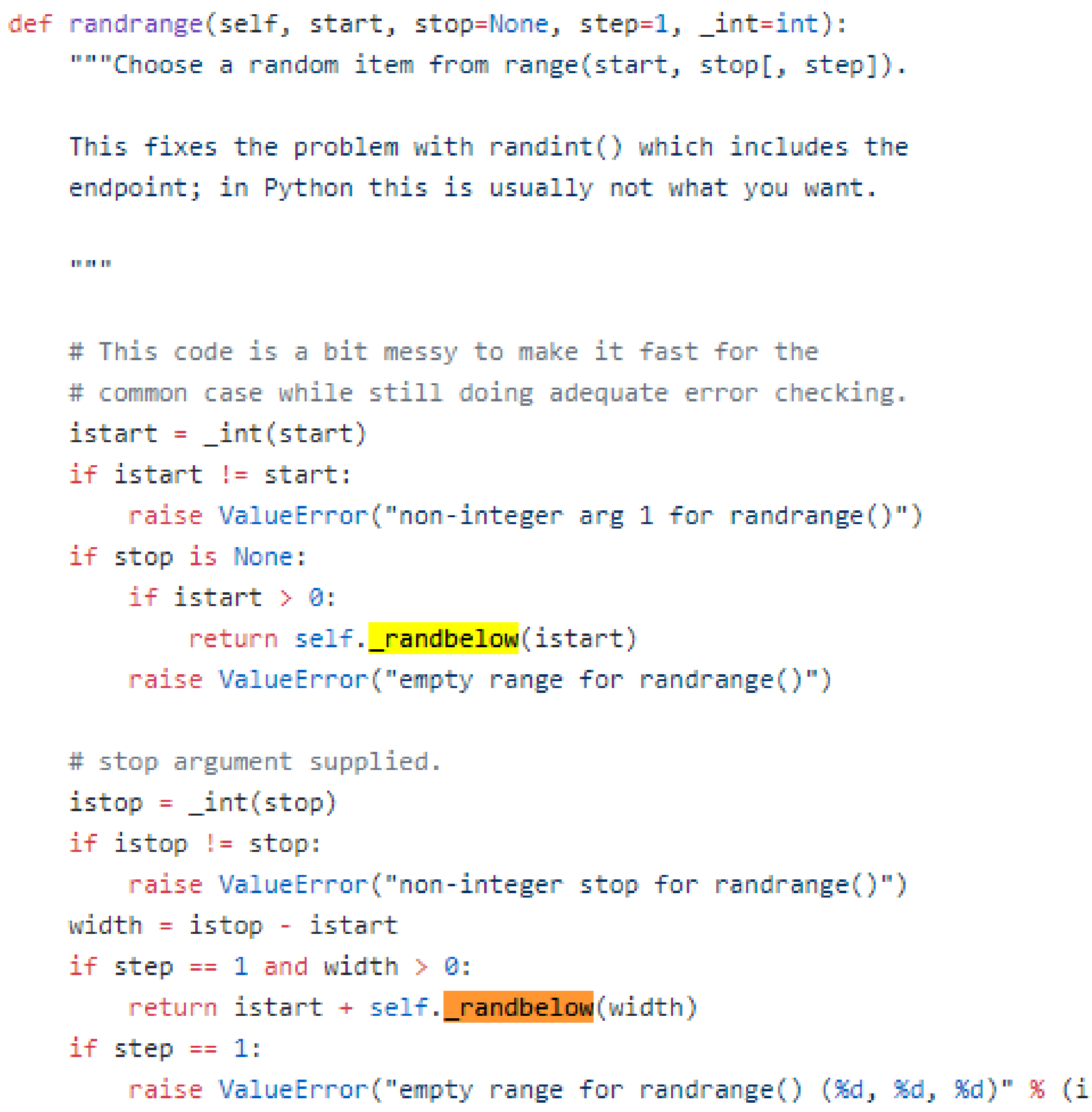
Lab2 – Crack MT19937

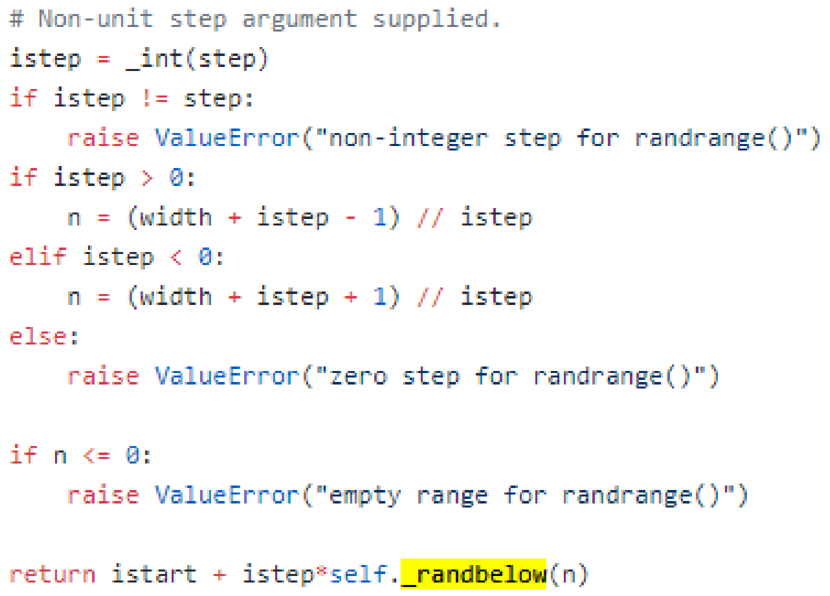
Crack MT19937



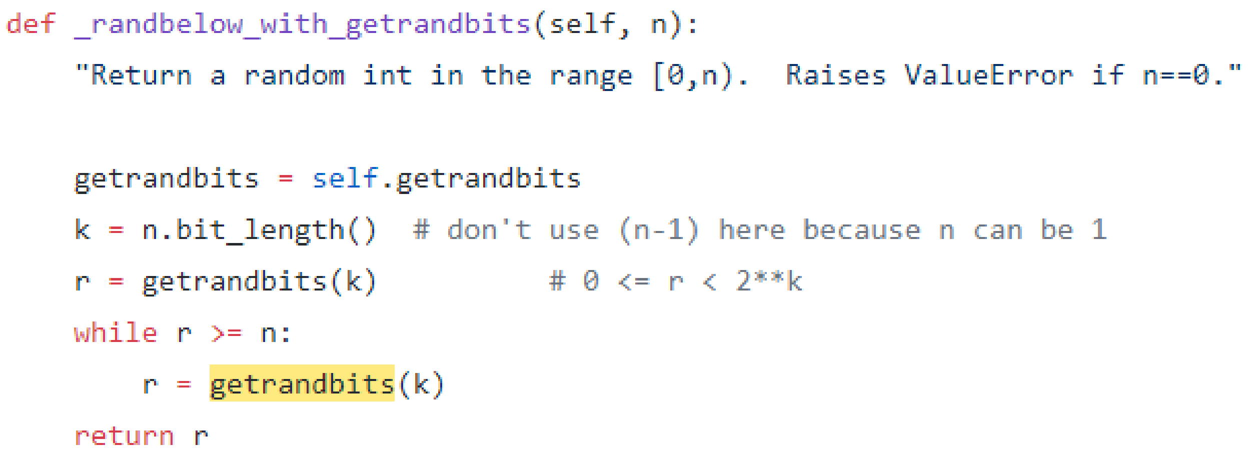
```
class MT19937:
    def __init__(self, seed):
        self.mt = [0] * 624
        self.mt[0] = seed
        self.mti = 0
        for i in range(1, 624):
            self.mt[i] = _int32(1812433253 * (self.mt[i - 1] ^ self.mt[i - 1] >> 30) + i)


    def extract_number(self):
        if self.mti == 0:
            self.twist()
        y = self.mt[self.mti]
        y = y ^ y >> 11
        y = y ^ y << 7 & 2636928640
        y = y ^ y << 15 & 4022730752
        y = y ^ y >> 18
        self.mti = (self.mti + 1) % 624
        return _int32(y)


    def twist(self):
        for i in range(0, 624):
            y = _int32((self.mt[i] & 0x80000000) + (self.mt[(i + 1) % 624] & 0x7fffffff))
            self.mt[i] = (y >> 1) ^ self.mt[(i + 397) % 624]

            if y % 2 != 0:
                self.mt[i] = self.mt[i] ^ 0x9908b0df
```

https://github.com/python/cpython/blob/master/Lib/random.py





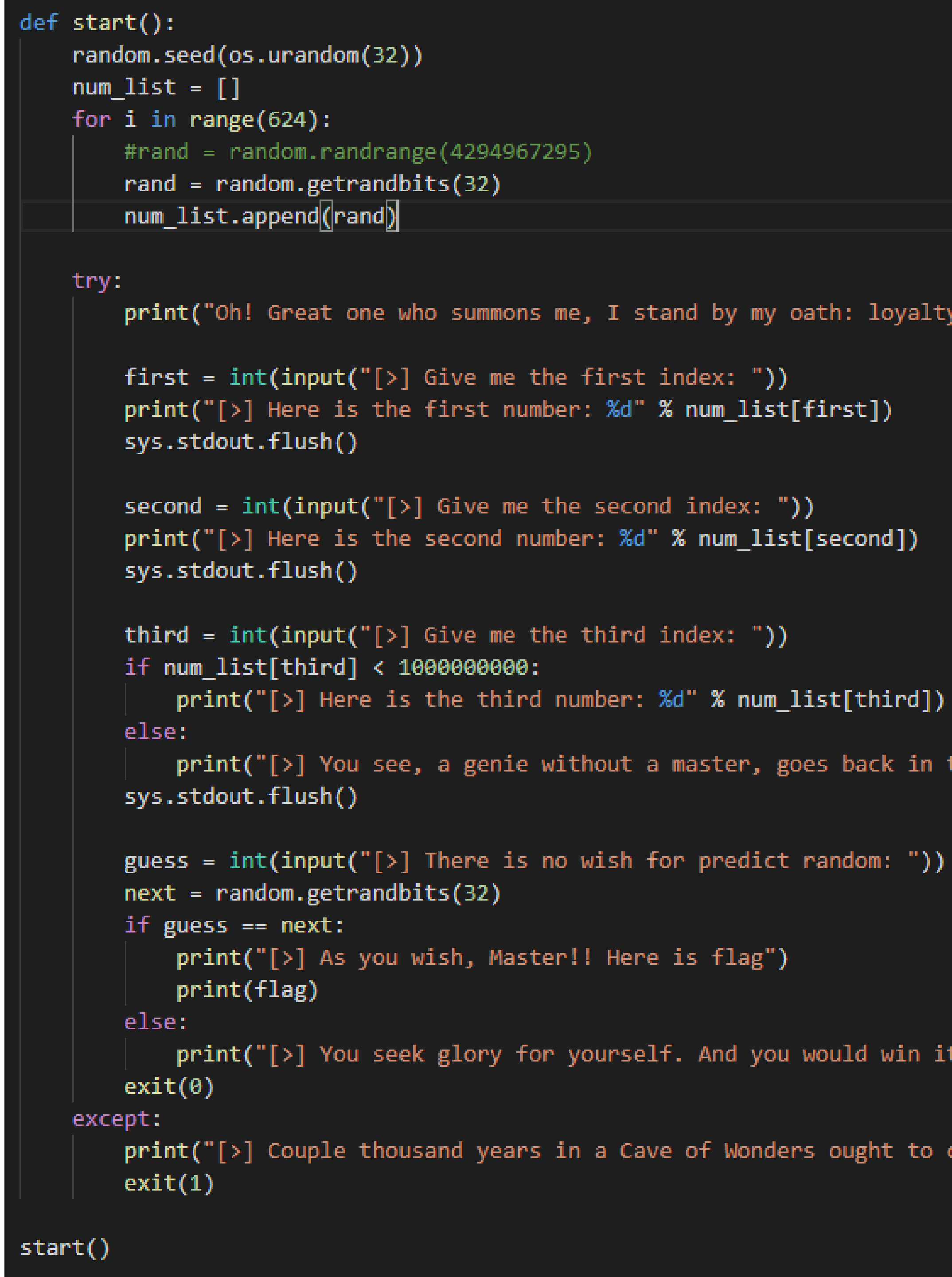
```python
def randrange(self, start, stop=None, step=1, _int=int):
    """Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

    """

    # This code is a bit messy to make it fast for the
    # common case while still doing adequate error checking.
    istart = _int(start)
    if istart != start:
        raise ValueError("non-integer arg 1 for randrange()")
    if stop is None:
        if istart > 0:
            return self._randbelow(istart)
        raise ValueError("empty range for randrange()")

    # stop argument supplied.
    istop = _int(stop)
    if istop != stop:
        raise ValueError("non-integer stop for randrange()")
    width = istop - istart
    if step == 1 and width > 0:
        return istart + self._randbelow(width)
    if step == 1:
        raise ValueError("empty range for randrange() (%d, %d, %d)" % (istart, istop, width))

    # Non-unit step argument supplied.
    istep = _int(step)
    if istep != step:
        raise ValueError("non-integer step for randrange()")
    if istep > 0:
        n = (width + istep - 1) // istep
    elif istep < 0:
        n = (width + istep + 1) // istep
    else:
        raise ValueError("zero step for randrange()")

    if n <= 0:
        raise ValueError("empty range for randrange()")

    return istart + istep*self._randbelow(n)
```



```python
def _randbelow_with_getrandbits(self, n):
    "Return a random int in the range [0,n).  Raises ValueError if n==0."

    getrandbits = self.getrandbits
    k = n.bit_length()  # don't use (n-1) here because n can be 1
    r = getrandbits(k)          # 0 <= r < 2**k
    while r >= n:
        r = getrandbits(k)
    return r
```

Crack MT19937 – Level 1

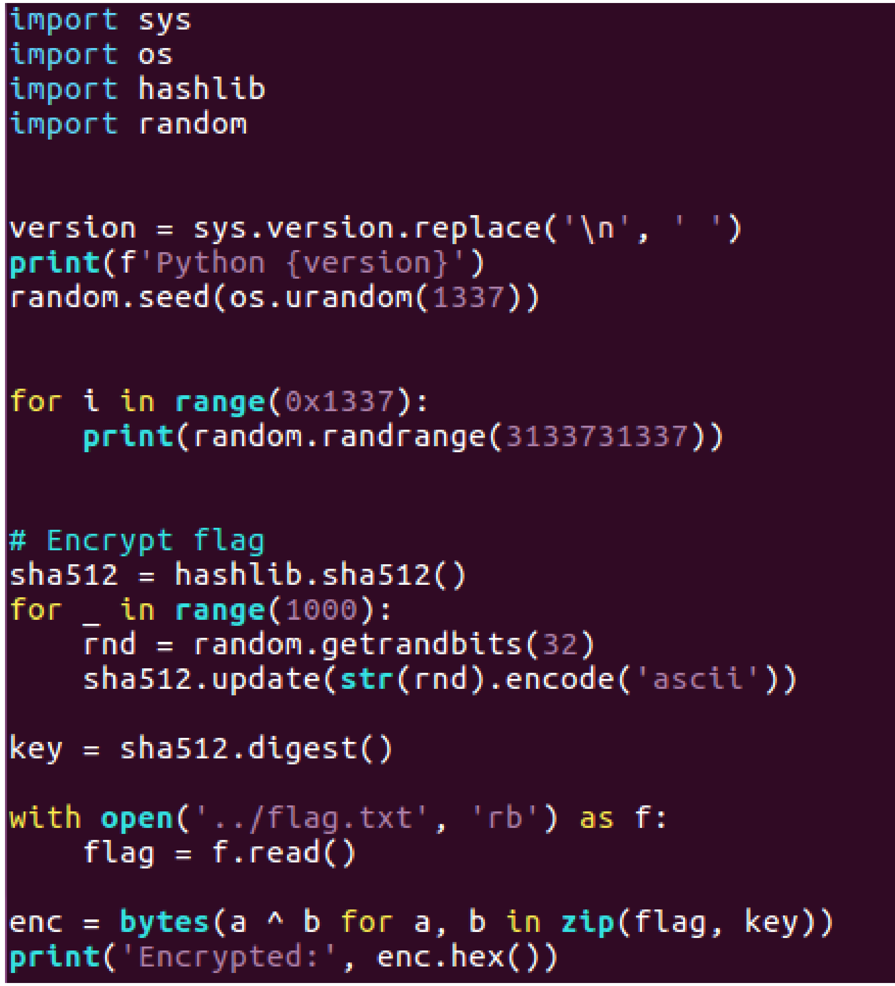
- **nc 140.114.77.172 60002**
- Choose 2 or 3 numbers from 624 “continuous” random number
- Each number in range(0, 4294967295)



```
def start():
    random.seed(os.urandom(32))
    num_list = []
    for i in range(624):
        #rand = random.randrange(4294967295)
        rand = random.getrandbits(32)
        num_list.append(rand)

    try:
        print("Oh! Great one who summons me, I stand by my oath: loyalt

        first = int(input("[>] Give me the first index: "))
        print("[>] Here is the first number: %d" % num_list[first])
        sys.stdout.flush()

        second = int(input("[>] Give me the second index: "))
        print("[>] Here is the second number: %d" % num_list[second])
        sys.stdout.flush()

        third = int(input("[>] Give me the third index: "))
        if num_list[third] < 1000000000:
            print("[>] Here is the third number: %d" % num_list[third])
        else:
            print("[>] You see, a genie without a master, goes back in
        sys.stdout.flush()

        guess = int(input("[>] There is no wish for predict random: "))
        next = random.getrandbits(32)
        if guess == next:
            print("[>] As you wish, Master!! Here is flag")
            print(flag)
        else:
            print("[>] You seek glory for yourself. And you would win i
        exit(0)
    except:
        print("[>] Couple thousand years in a Cave of Wonders ought to
        exit(1)

start()
```

Crack MT19937 – Level 2

- 2019 BalsnCTF - unpredictable
  - Filter out 25% number
  - Use the relationship between $X_{i+1}, X_{i+397}, X_{i+624}$
  - Construct tree structure to verify the index
  - Reduce to the shortest path problem
- How much filter ratio is probably non-invertible?



```python
import sys
import os
import hashlib
import random


version = sys.version.replace('\n', ' ')
print(f'Python {version}')
random.seed(os.urandom(1337))


for i in range(0x1337):
    print(random.randrange(3133731337))


# Encrypt flag
sha512 = hashlib.sha512()
for _ in range(1000):
    rnd = random.getrandbits(32)
    sha512.update(str(rnd).encode('ascii'))

key = sha512.digest()

with open('../flag.txt', 'rb') as f:
    flag = f.read()

enc = bytes(a ^ b for a, b in zip(flag, key))
print('Encrypted:', enc.hex())
```

Cryptographic Backdoor

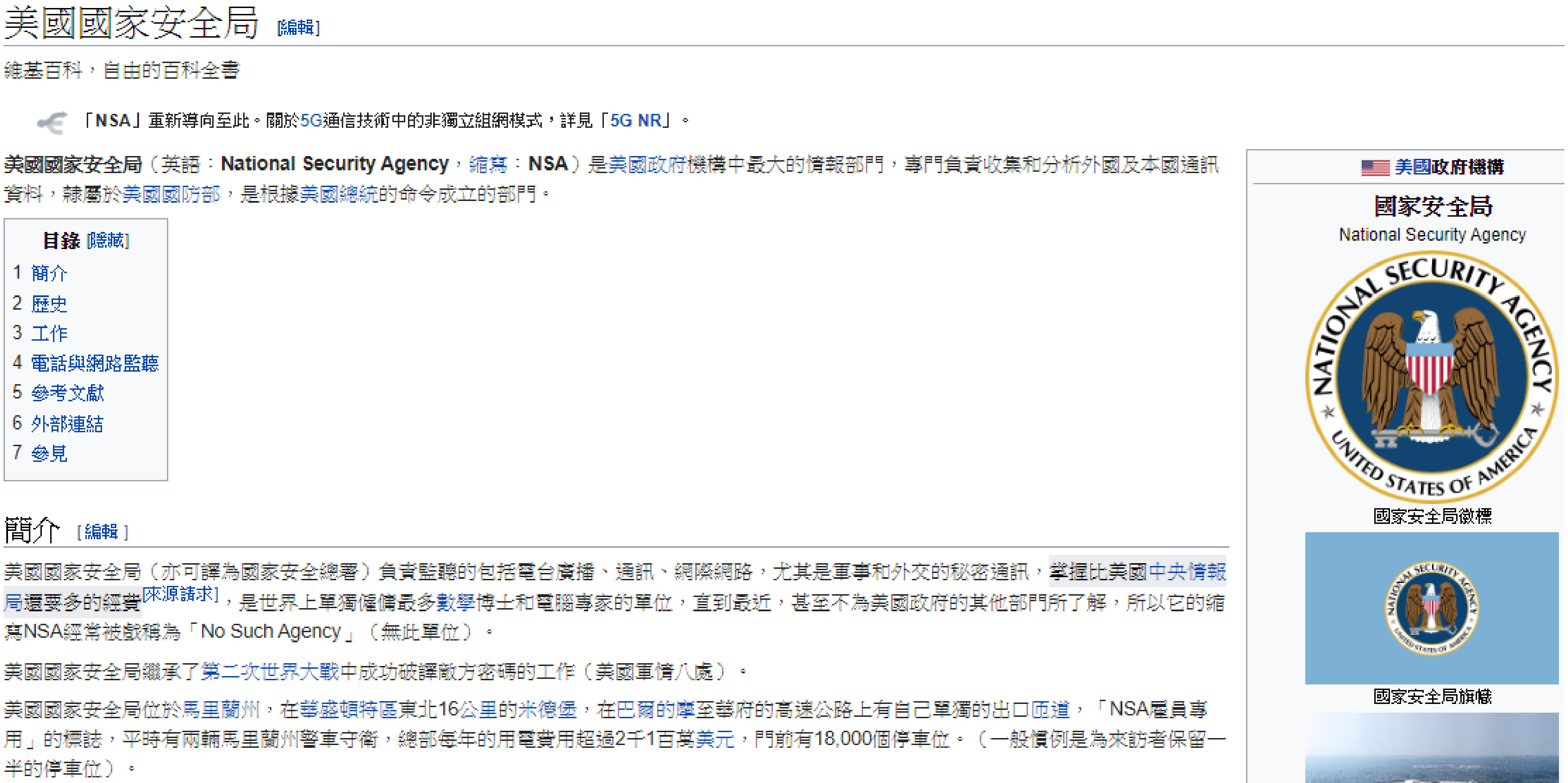
Trapdoor V.S. Backdoor

- Trapdoor
  - One way function
  - 不知道密鑰的人要算很久
- Backdoor
  - 定義:
    - A feature or defect that allows surreptitious access to data
  - 特色:
    - NOBUS (No One But Us)
    - Deniable
    - Reusable (optional)
    - Unmalleable (hard to replicate)
    - Forward-secure (previous exploits aren’t compromised)

Why random number is important?

- Most of Cryptosystems need random number
    - RSA: p,q
    - AES: iv, key
    - DSA: private key
    - ECC: points on curve
- If we enable to predict PRNG
    - We know everything!!!

NSA



美國國家安全局 [編輯]

維基百科，自由的百科全書

「NSA」重新導向至此。關於5G通信技術中的非獨立組網模式，詳見「5G NR」。

美國國家安全局（英語：**National Security Agency**，縮寫：**NSA**）是美國政府機構中最大的情報部門，專門負責收集和分析外國及本國通訊資料，隸屬於美國國防部，是根據美國總統的命令成立的部門。

目錄 [隱藏]

簡介 [編輯]

美國國家安全局（亦可譯為國家安全總署）負責監聽的包括電台廣播、通訊、網際網路，尤其是軍事和外交的秘密通訊，掌握比美國中央情報局還要多的經費[來源請求]，是世界上單獨僱傭最多數學博士和電腦專家的單位，直到最近，甚至不為美國政府的其他部門所了解，所以它的縮寫NSA經常被戲稱為「No Such Agency」（無此單位）。

美國國家安全局繼承了第二次世界大戰中成功破譯敵方密碼的工作（美國軍情八處）。

美國國家安全局位於馬里蘭州，在華盛頓特區東北16公里的米德堡，在巴爾的摩至華府的高速公路上有自己單獨的出口匝道，「NSA雇員專用」的標誌，平時有兩輛馬里蘭州警車守衛，總部每年的用電費用超過2千1百萬美元，門前有18,000個停車位。（一般慣例是為來訪者保留一半的停車位）。

美國政府機構

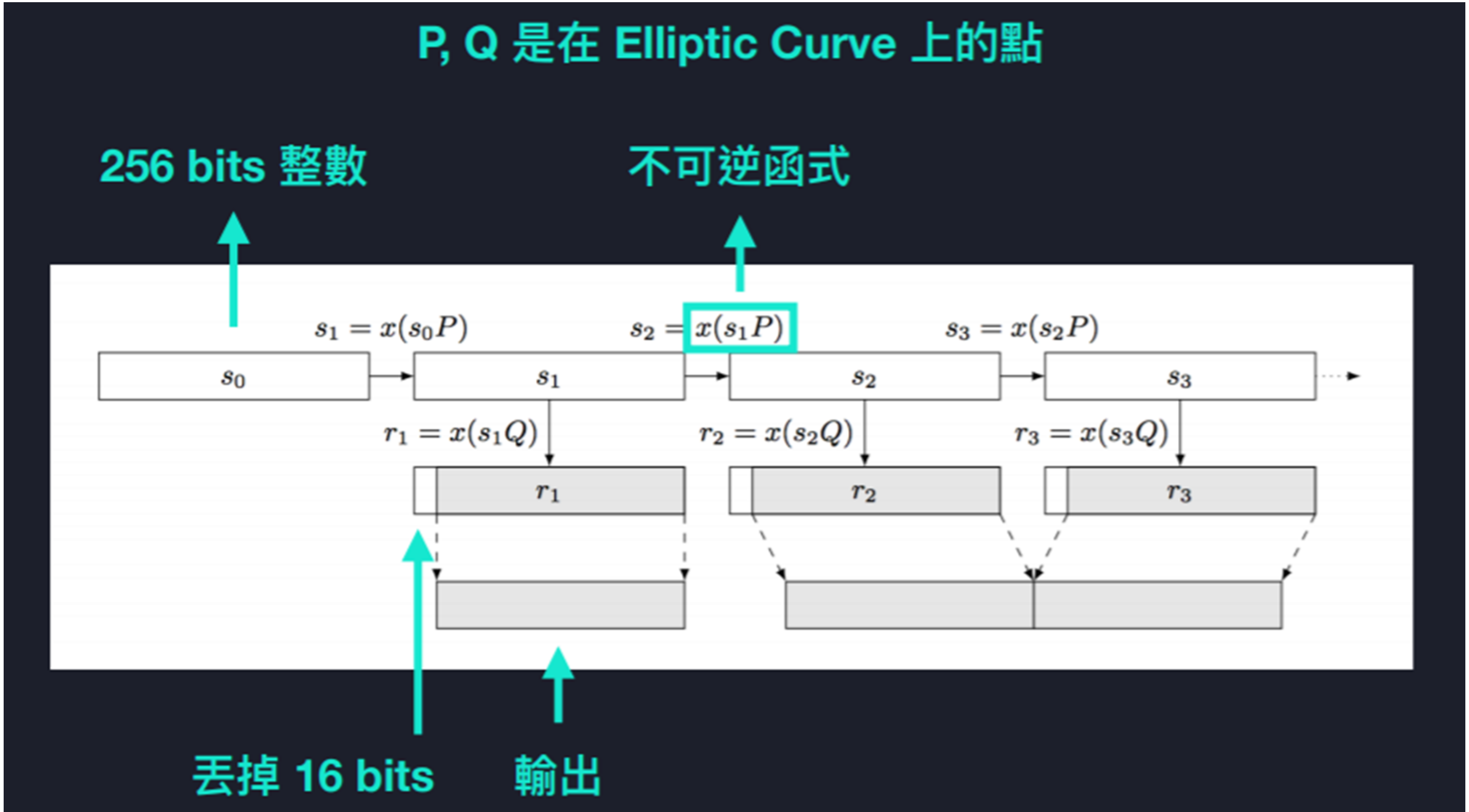
國家安全局

National Security Agency

國家安全局徽標

國家安全局旗幟

NSA – EC DRBG



P, Q 是在 Elliptic Curve 上的點

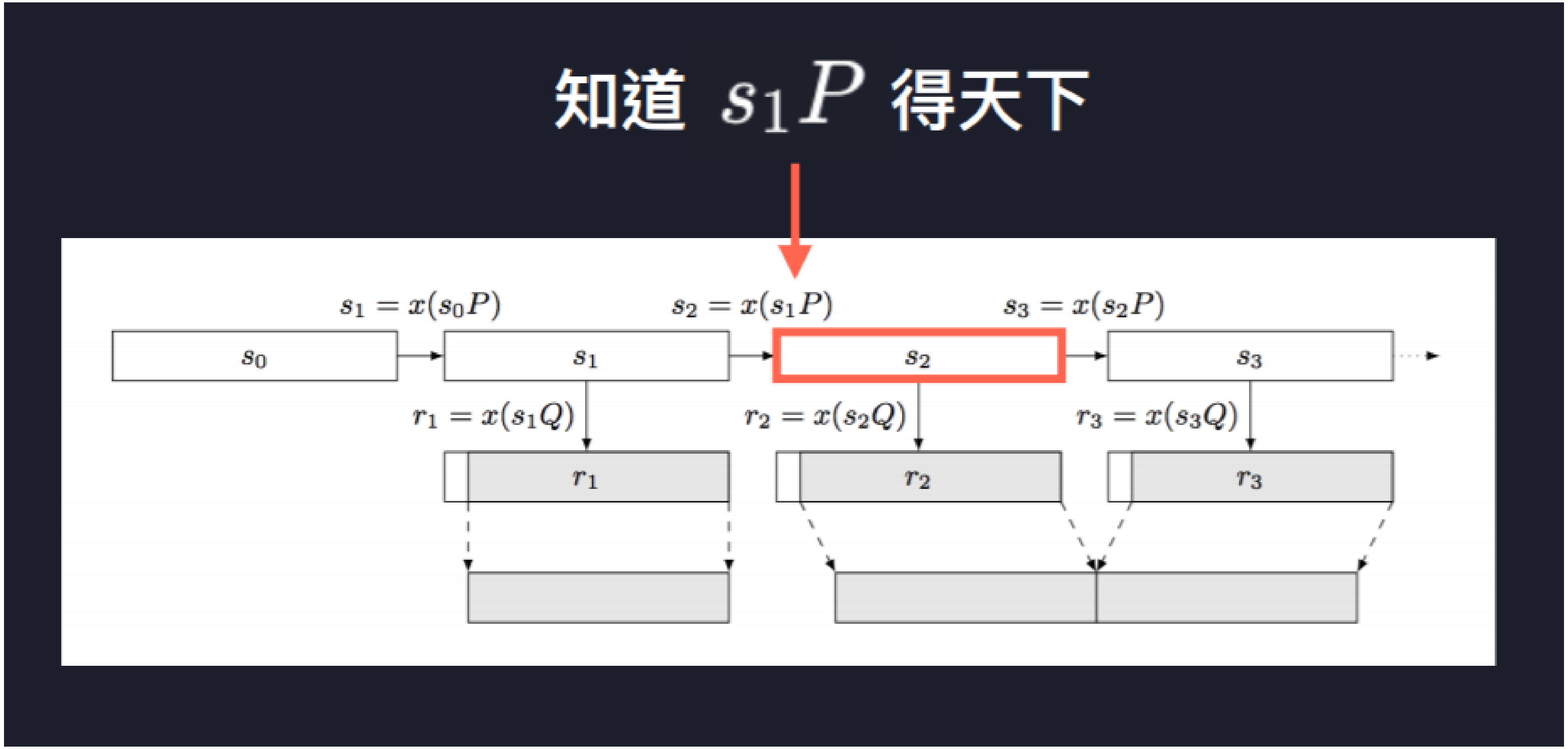
256 bits 整數

不可逆函式

丟掉 16 bits

輸出

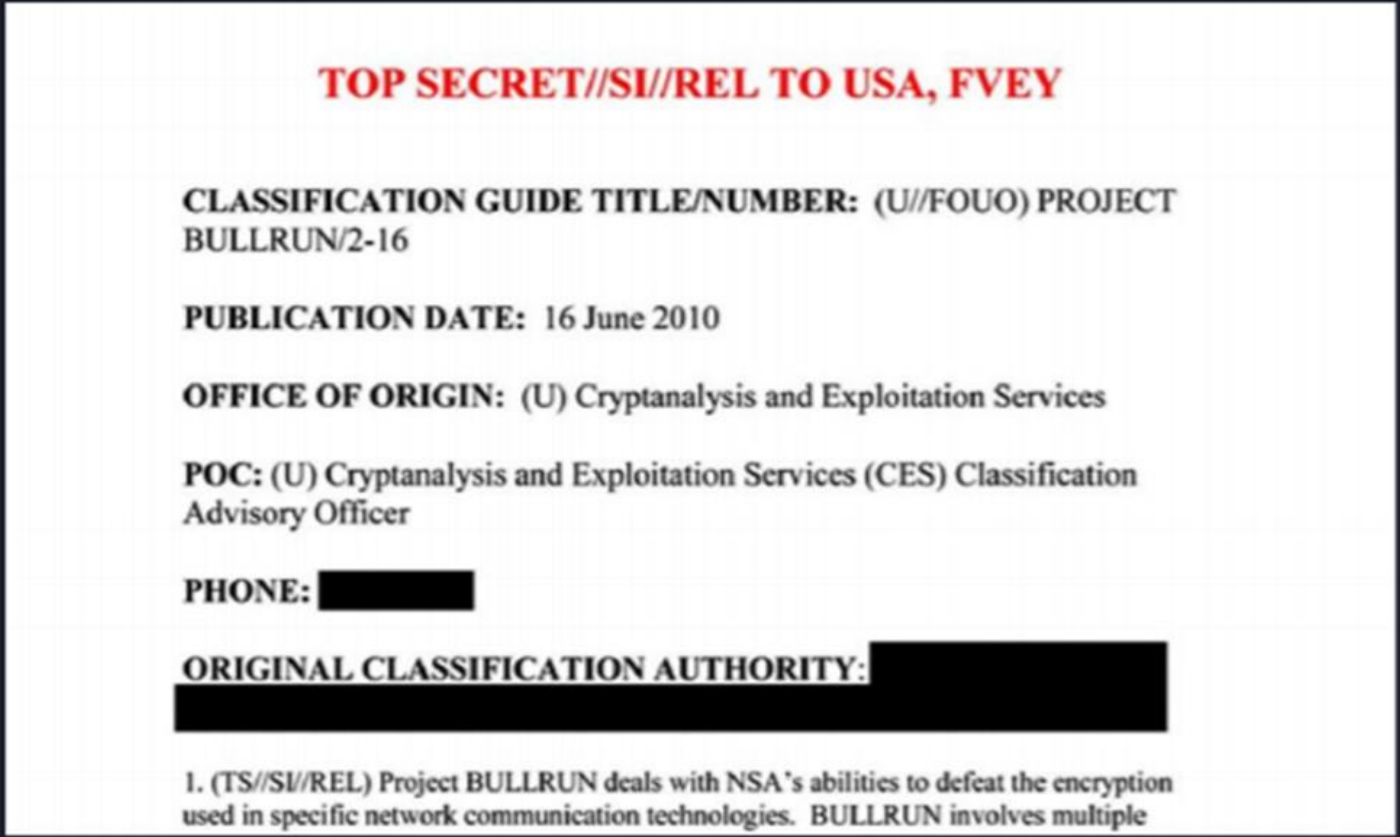
https://blog.0xbadc0de.be/archives/155



知道 s_1P 得天下

2013

Snowden leaks - Project BULLRUN
驗證後門真的存在



TOP SECRET//SI//REL TO USA, FVEY

CLASSIFICATION GUIDE TITLE/NUMBER: (U//FOUO) PROJECT BULLRUN/2-16

PUBLICATION DATE: 16 June 2010

OFFICE OF ORIGIN: (U) Cryptanalysis and Exploitation Services

POC: (U) Cryptanalysis and Exploitation Services (CES) Classification Advisory Officer

PHONE: [redacted]

ORIGINAL CLASSIFICATION AUTHORITY: [redacted]

1. (TS//SI//REL) Project BULLRUN deals with NSA's abilities to defeat the encryption used in specific network communication technologies. BULLRUN involves multiple

How 2 Random?

The Correct Way

- Understand algorithms of PRNG
- Change your seed periodically
- Make sure your seed is from hardware
- Also make sure you have enough entropy

Reference

- https://oalieno.github.io/security/crypto/classic/dual-ec/
- Sasdf slides from NCTU CTF

Thanks for Your Listening!