

Cryptography

Part 1 of 3

By: Killua4564

Index

- Encoding
- Classical cryptography
 - Substitution ciphers
 - Transposition ciphers
 - The other ciphers
- Symmetric-key algorithm
 - Stream cipher
 - Keystream
- Public-key cryptography
 - Number theory
 - RSA

Python tools

- pwntools
- pycrypto / pycryptodome
- gmpy / gmpy2
- mersenne-twister-predictor
- pycipher
- tqdm
- xortool

Python support

bytes to string	<code>var1.decode()</code>
string to bytes	<code>var2.encode()</code>
bytes to hex-string	<code>var3.hex()</code>
hex-string to bytes	<code>bytes.fromhex(var4)</code>
bytes to integer	<code>from Crypto.Util.number import bytes_to_long</code> <code>bytes_to_long(var5)</code>
	<code>int.from_bytes(var5, byteorder="big")</code>
integer to bytes	<code>from Crypto.Util.number import long_to_bytes</code> <code>long_to_bytes(var6)</code>
	<code>var6.to_bytes(byteorder="big")</code>

Python support

base64	<pre>import base64 base64.b32encode(var1.encode()).decode() base64.b32decode(var2.encode()).decode() base64.b64encode(var3.encode()).decode() base64.b64decode(var4.encode()).decode()</pre>
functools	<pre>import functools functools.reduce(lambda x, y: x + y, [1, 2, 3, 4, 5]) # 15</pre>
itertools	<pre>import itertools itertools.count(1) # 1, 2, 3, 4, 5, ... itertools.product("ABCD", repeat=2) # AA, AB, AC, AD, BA, BB, ... itertools.combinations("ABCD", 2) # AB, AC, AD, BC, BD, CD</pre>
string	<pre>import string string.digits # 0123456789 string.ascii_lowercase # abcdefghijklmnopqrstuvwxyz string.ascii_uppercase # ABCDEFGHIJKLMNOPQRSTUVWXYZ string.printable</pre>

Python support

pwntools	<pre>from pwn import remote conn = remote(addr, port) conn.sendline(data) conn.recvuntil(delims) conn.sendlineafter(delims, data)</pre>
pycryptodome	<pre>from Crypto.Util.number import GCD, inverse, isPrime g = GCD(x, y) d = inverse(e, phi) ok = isPrime(p)</pre>
gmpy2	<pre>import gmpy2 r = gmpy2.isqrt(x) r, ok = gmpy2.iroot(x, k) ok = gmpy2.is_prime(x) p = gmpy2.next_prime(x)</pre>

Encoding

Base58

- 用於 Bitcoin 中
- 避免掉容易讓人混淆的字元
- 跟 base64 比起來少了 0, O, l, I, +, /

編碼	字符	編碼	字符	編碼	字符	編碼	字符
0	1	16	H	32	Z	48	q
1	2	17	J	33	a	49	r
2	3	18	K	34	b	50	s
3	4	19	L	35	c	51	t
4	5	20	M	36	d	52	u
5	6	21	N	37	e	53	v
6	7	22	P	38	f	54	w
7	8	23	Q	39	g	55	x
8	9	24	R	40	h	56	y
9	A	25	S	41	i	57	z
10	B	26	T	42	j		
11	C	27	U	43	k		
12	D	28	V	44	m		
13	E	29	W	45	n		
14	F	30	X	46	o		
15	G	31	Y	47	p		

Base85

- 又稱 ascii85
- 希望在 human-readable 的情況下盡量縮短 encode 長度
- 使用 0-9 A-Z a-z 和 23 個符號 !#\$%&()*+,-;=>?@^_`{|}~
- 範例：

Man is distinguished, not only by his reason, but by this singular passion from other animals, which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable generation of knowledge, exceeds the short vehemence of any carnal pleasure.

```
9jqo^BlbD-BleB1DJ+*+F(f,q/0JhKF<GL>Cj@.4Gp$d7F!,L7@<6@)/0JDEF<G%<+EV:2F!,O<
DJ+*.@<*K0@<6L(Df-\0Ec5e;DffZ(EZee.B1.9pF"AGXBPCsi+DGm>@3BB/F*&OCAfu2/AKYi(
DIb:@FD,*)+C]U=@3BN#EcYf8ATD3s@q?d$AftVqCh[NqF<G:8+EV:..+Cf>-FD5W8ARloldIal(
DIId<j@<?3r@:F%a+D58'ATD4$B1@l3De:,-DJs`8ARoFb/0JMK@qB4^F!,R<AKZ&-DfTqBG%G>u
D.RTpAKYo'+CT/5+CeI#DII?(E,9)oF*2M7/c
```

AAEncode / JJEncode

Enter JavaScript source:

```
alert("Hello, JavaScript")
```

aaencode

```
°ω°/= /`m´) /~┐┐ // *∇`*/ ['_']; o=(° -°) =_3; c=(° Θ°) =(° -°)-(° -°); (° Д°) =(° Θ°) =(o^_o)/(o^_o);(° Д°)=(° Θ°: '_', ° ω° /:( (° ω° /==3) +'_') [° Θ° ], ° -° /:(° ω° /+ '_')[o^_o -(° Θ°) ], ° Д° /:( (° -° ==3) +'_') [° -° ]]; (° Д°) [° Θ°] =( (° ω° /==3) +'_') [c^_o];(° Д°) ['c'] =( (° Д°) +'_') [ (° -°) + (° -°) -(° Θ°) ];(° Д°) ['o'] =( (° Д°) +'_') [° Θ° ];(° o°) =(° Д°) ['c'] + (° Д°) ['o'] + (° ω° /+ '_') [° Θ° ] + ( (° ω° /==3) +'_') [° -° ] + ( (° Д°) +'_') [ (° -°) + (° -°) ] + ( (° -° ==3) +'_') [° Θ° ] + ( (° -° ==3) +'_') [ (° -°) - (° Θ°) ] + (° Д°) ['c'] + ( (° Д°) +'_') [ (° -°) + (° -°) ] + (° Д°) ['o'] + ( (° -° ==3) +'_') [° Θ° ];(° Д°) ['_'] =(o^_o) [° o° ] [° o° ];(° ε°) =( (° -° ==3) +'_') [° Θ° ] + (° Д°) . Д° /+ ( (° Д°) +'_') [ (° -°) + (° -°) ] + ( (° -° ==3) +'_') [o^_o -° Θ° ] + ( (° -° ==3) +'_') [° Θ° ] + (° ω° /+ '_') [° Θ° ]; (° -°) +=(° Θ° ); (° Д°) [° ε°] ="\\"; (° Д°) . Θ° /=(° Д° +° -°) [o^_o -(° Θ°) ]; (o^_o) =(° ω° /+ '_') [c^_o];(° Д°) [° o°] ="\\"; (° Д°) ['_'] ( (° Д°) ['_'] (° ε° + (° Д°) [° o° ] + (° Д°) [° ε°] + (° Θ°) + (° -°) + (° Θ°) + (° Д°) [° ε°] + (° Θ°) + ( (° -°) + (° Θ°) ) + (° -°) + (° Д°) [° ε°] + (° Θ°) + (° -°) + ( (° -°) + (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + ((o^_o) - (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + (° -°) + (° Д°) [° ε°] + ( (° -°) + (° Θ°) ) + (c^_o) + (° Д°) [° ε°] + (° -°) + ((o^_o) - (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + (° Θ°) + (c^_o) + (° Д°) [° ε°] + (° Θ°) + (° -°) + ( (° -°) + (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + ( (° -°) + (° Θ°) ) + (° -°) + (° Д°) [° ε°] + (° Θ°) + ( (° -°) + (° Θ°) ) + (° -°) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) - (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + (° -°) + (° Θ°) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + ((o^_o) + (o^_o)) + (° Д°) [° ε°] + (° Θ°) + (° -°) + (° Θ°) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) - (° Θ°) ) + (o^_o) + (° Д°) [° ε°] + (° Θ°) + (° -°) + (o^_o) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + ((o^_o) - (° Θ°) ) + (° Д°) [° ε°] + (° Θ°) + ( (° -°) + (° Θ°) ) + (° Θ°) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + (c^_o) + (° Д°) [° ε°] + (° Θ°) + ((o^_o) + (o^_o)) + (° -°) + (° Д°) [° ε°] + (° -°) + ((o^_o) - (° Θ°) ) + (° Д°) [° ε°] + ( (° -°) + (° Θ°) ) + (° Θ°) + (° Д°) [° o° ] (° Θ°) ('_');
```

UUEncode

- 延伸意思為 Unix-to-Unix encoding
- 以前常於 email 和 BBS 中看到
- 現在幾乎被 MIME 取代了

c						a						t					
67						97						116					
0	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1
16						54						5					
48						86						37					
0						v						%					

Column Row ↘	2	3	4	5
0	SP	0	@	P
1	!	1	A	Q
2	"	2	B	R
3	#	3	C	S
4	\$	4	D	T
5	%	5	E	U
6	&	6	F	V
7	'	7	G	W
8	(8	H	X
9)	9	I	Y
10	*	:	J	Z
11	+	;	K	[
12	,	<	L	\
13	-	=	M]
14	.	>	N	^
15	/	?	O	_

XXEncode

- 改進的 UUEncode
- 開頭會加上字串長度一起編碼

12345678901234567																											
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ! + () [] { }

```
alert(1)
```

编码 3

演示 10

幫助 ♥

下载 |

©2010-2012 天马行空工作室
推动开源 推动网络安全
Random_Coder

[illegible][illegible]

JSFuck

- ! + [] ()

```
false      =>  ![]
true       =>  !![]
undefined  =>  [] [[]]
NaN        =>  +[![]]
0          =>  +[]
1          =>  +!+[]
2          =>  !+[]+!+[]
10         =>  [+!+[]]+[+[]]
Array      =>  []
Number     =>  +[]
String     =>  []+[]
Boolean    =>  ![]
Function   =>  []["filter"]
eval       =>  []["filter"]["constructor"]( CODE )()
window     =>  []["filter"]["constructor"]("return this")()
```

```
alert(1)
```

Encode

☐ Eval Source
$$\begin{aligned} & ((! [] + []) + (! + []) + ((! [] + []) [! + [] + ! + []] + (! [] + []) [! + [] + ! + [] + []] + (! [] + \\ & []) [! + []] + (! [] + []) [+ []] + (! [] + []) (! [] + []) [+ []] + (! [] + []) [[]]) [! + [] + \\ & + [[]]] + (! [] + []) [! + [] + ! + []] + (! [] + []) [+ []] + (! [] + []) [! + [] + ! + [] + []] + \\ & (! [] + []) [! + []]) [! + [] + ! + [] + [+ []]] + [+ ! + []] + (! [] + []) (! [] + []) [+ []] + (! \\ & [] + []) [[]]) [! + [] + [+ []]] + (! [] + []) [! + [] + ! + []] + (! [] + []) [+ []] + (! [] + []) \\ & [! + [] + ! + [] + [+ []]] + (! [] + []) [+ ! + []]) [! + [] + ! + [] + [+ []]] \end{aligned}$$

BrainFuck

- 簡稱 BF, 是一種極小化的程式語言
- 目的是用最簡單、最小化編譯器創建有圖靈完備性的程式語言
- 舉例: print("Hello World!")

```
+++++++[>+++++>+++++++>+++>+<<<<-]  
>++.>+.+++++.++.>+.<<+++++.  
>+.+++.-----.->+.>.
```

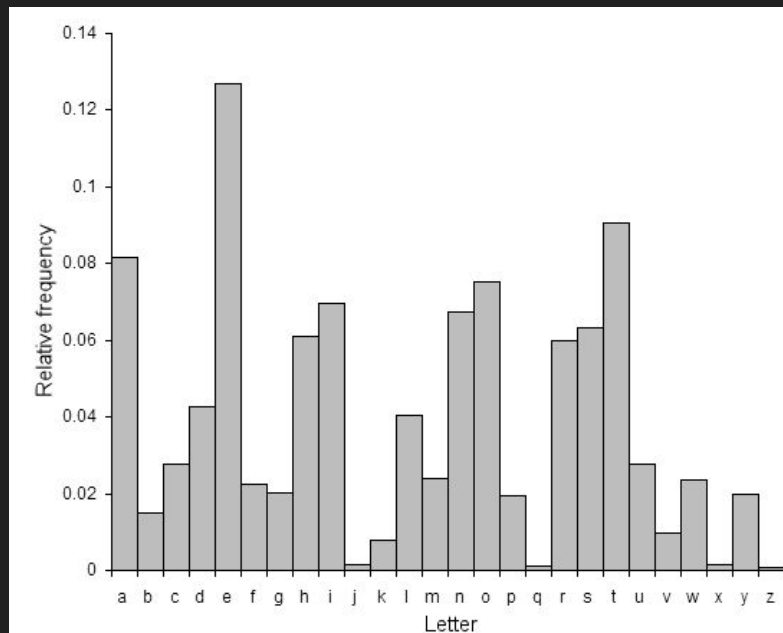
字元	含義
>	指標加一
<	指標減一
+	指標所指位元組的值加一
-	指標所指位元組的值減一
.	輸出指標所指位元組內容 (ASCII碼)
,	向指標所指的位元組輸入內容 (ASCII碼)
[若指標所指位元組的值為零，則向後跳轉，跳轉到其對應的] 的下一個指令處
]	若指標所指位元組的值不為零，則向前跳轉，跳轉到其對應的 [的下一個指令處

Classical cryptography

Substitution ciphers

Substitution ciphers

- 將字母做有系統的代換，替換成難以理解的文字
- 單表代換加密
 - 單一表的規律去替換字母
 - 明密文為一對一的對應關係
 - 密鑰空間較小則暴力破解
 - 密文長度夠長則詞頻分析
 - [Bigram](#)、[Trigram](#)
- 多表代換加密
 - 以多維表去多向對應替換字母
 - 明文的詞頻特性已經不在
 - 一般只能對各自算法找弱點破解



Caesar cipher

- 將明文每個字母在表中，向前或向後移動固定數目
- 舉例 key=3
 - 明文字母表: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 - 密鑰字母表: DEFGHIJKLMNOPQRSTUVWXYZABC
- 加解密
 - 明文: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
 - 密文: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
- 當 key=13 時叫 ROT13
- 破解 ([online](#))

Simple substitution cipher

- 將明文每個字母對應到完全混亂的字母表進行替換
- 舉例
 - 明文字母表: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 - 密鑰字母表: PHQGIUMEAYLNOFDXJKRCVSTZWB
- 加解密
 - 明文: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
 - 密文: CEI JVAQL HKDTF Udz YVOXR DSIK CEI NPBW GDM
- 密鑰字母表為明文字母表反轉時叫做埃特巴什碼 (Atbash cipher)
- 破解 ([online](#))

Affine cipher

- 將明文中的字母對應成數字後進行運算加密
- $E(x) = ax + b \bmod m$
- 最後再對應回來成為密文
- 舉例 $(a, b) = (5, 8)$ 對明文 = AFFINE CIPHER 進行加密

明文	A	F	F	I	N	E	C	I	P	H	E	R
x	0	5	5	8	13	4	2	8	15	7	4	17
$y = 5x + 8$	8	33	33	48	73	28	18	48	83	43	28	93
$y \bmod 26$	8	7	7	22	21	2	18	22	5	17	2	15
密文	I	H	H	W	V	C	S	W	F	R	C	P

Playfair cipher

- 將密鑰不重複依序填入 5*5 的矩陣內, 剩下的空間由未出現的字母依序填滿
 - 通常會去除 Q 或將 I/J 視為同一個字
- 將明文字母兩兩一組當成對角線, 在矩陣中畫出每組的矩形
 - 如果相同字母分到一組, 則中間加入 Q 或 X
- 再將每組矩形的另一條對角線對應成密文
 - 如果在同個 row 則取右邊的字母
 - 如果在同個 column 則取下面的字母
- 舉例 key=playfair example
 - 明文: HIDE THE GOLD IN THE TREE STUMP
 - 密文: BMODZBXDNABEKUDMUIXMMOUVIF
- 破解
 - [Simulated annealing](#)、[Genetic algorithm](#)

P	L	A	Y	F
I	R	E	X	M
B	C	D	G	H
K	N	O	Q	S
T	U	V	W	Z

Polybius square

- 在 5*5 的矩陣裡填入字母形成表
- 並用 5 個字元排序作為座標軸對應
- 舉例 1

- 明文:HELLO
- 密文:2315313134

- 舉例 2 (ADFGX 密碼)

- 明文:HELLO
- 密文:DDXFAGAGDF

- ADFGX 由來

- 1918 年, 第一次世界大戰將要結束時, 法軍截獲了一份德軍電報, 電文中的所有單詞都由 ADFGX 五個字母拼成, 因此被稱為 ADFGX 密碼。

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z
	A	D	F	G	X
A	b	t	a	l	p
D	d	h	o	z	k
F	q	f	v	s	n
G	g	j	c	u	x
X	m	r	e	w	y

Vigenère square

- 由一系列的凱薩組合的簡單多表加密
- 舉例 key=crypto
 - 明文: come greatwall
 - 密鑰擴充後對應密文: efkt zferrltzn

c	o	m	e	g	r	e	a	t	w	a	l	l
c	r	y	p	t	o	c	r	y	p	t	o	c

- 破解方式 ([online](#) [online](#))
 - 密鑰是循環的, 若知道密鑰長度 L
 - 則變成 L 個凱薩加密的問題
 - 卡西斯基試驗 (Kasiski examination)
 - 弗里德曼試驗 (Friedman test)
 - 詞頻分析

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère square

- 卡斯基試驗

- 常出現的單詞 (e.g. the) 有可能被同樣的金鑰部分加密
- 如果能找到多組重複的密文, 就有機會猜出金鑰長度

密鑰: ABCDAB CD ABCDA BCD ABCDABCDABCD
明文: **CRYPTO** IS SHORT FOR **CRYPTOGRAPHY**
密文: **CSASTP** KV SIQUT GQU **CSASTP**IUAQJB

- **DYDUXRMHTVDVNQDQNWDYDUXRMHARTJGWNQD**
- **DYDUXRMH** 間隔 18 個字母, 金鑰長度可能是 2, 3, 6, 9, 18
- **NQD** 間隔 20 個字母, 金鑰長度可能是 2, 5, 10, 20
- 此時就可以猜金鑰長度是 2 去嘗試破密

- 弗里德曼試驗

- 使用重合指數 (index of coincidence) 以密文字母頻率不均性來推測金鑰長度
- K_p 表示目標語言中任意兩個字母相同的機率 (0.067)
- K_r 表示字母表情況出現的概率 (1/26)
- $K_o = \frac{\sum_{i=1}^c n_i(n_i - 1)}{N(N - 1)}$, n_i 為字母 i 在密文中的頻率, N 是文本長度

$$\frac{K_p - K_r}{K_o - K_r}$$

Hill cipher

- 將明文每個字母對應成數字並化為向量，長度為 N
- 密鑰生成 $N * N$ 的矩陣 (必須可逆, $\gcd(\det(\text{矩陣}), \text{len}(\text{字母表})) = 1$)
- 相乘後結果對應回密文
- 舉例

- 明文: ACT
- 金鑰: GNUYQRBKP
- 密文: POH

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \equiv \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

Classical cryptography

Transposition ciphers

Transposition ciphers

- 字母本身不變
- 依照有系統的規則改變訊息的順序
- 通常是基於幾何設計

Plaintext	Transposition matrix 1a	Transposition matrix 1b
I ₁ P ₂ U ₃ L ₄ L ₅ E ₆ D ₇ T ₈ H ₉ E ₁₀ L ₁₁ E ₁₂	P ₁ E ₂ R ₃ M ₄ U ₅ T ₆ A ₇ T ₈ I ₉ O ₁₀ N ₁₁ S ₁₂	A ₁ E ₂ I ₃ M ₄ N ₅ O ₆ P ₇ R ₈ S ₉ T ₁₀ T ₁₁ U ₁₂
V ₁₃ E ₁₄ R ₁₅ A ₁₆ N ₁₇ D ₁₈ A ₁₉ C ₂₀ T ₂₁ I ₂₂ V ₂₃ A ₂₄	I ₁ P ₂ U ₃ L ₄ L ₅ E ₆ D ₇ T ₈ H ₉ E ₁₀ L ₁₁ E ₁₂	D ₁ P ₂ H ₃ L ₄ L ₅ E ₆ I ₇ U ₈ E ₉ E ₁₀ T ₁₁ L ₁₂
T ₁₃ E ₁₄ D ₁₅ A ₁₆ S ₁₇ E ₁₈ C ₁₉ R ₂₀ E ₂₁ T ₂₂ M ₂₃ E ₂₄	V ₁₃ E ₁₄ R ₁₅ A ₁₆ N ₁₇ D ₁₈ A ₁₉ C ₂₀ T ₂₁ I ₂₂ V ₂₃ A ₂₄	A ₁ E ₂ T ₃ A ₄ V ₅ I ₆ V ₇ R ₈ A ₉ D ₁₀ C ₁₁ N ₁₂
C ₁₃ H ₁₄ A ₁₅ N ₁₆ I ₁₇ S ₁₈ M ₁₉ U ₂₀ N ₂₁ V ₂₂ E ₂₃ I ₂₄	T ₁₃ E ₁₄ D ₁₅ A ₁₆ S ₁₇ E ₁₈ C ₁₉ R ₂₀ E ₂₁ T ₂₂ M ₂₃ E ₂₄	C ₁ E ₂ E ₃ A ₄ M ₅ T ₆ T ₇ D ₈ E ₉ E ₁₀ R ₁₁ S ₁₂
L ₁₃ I ₁₄ N ₁₅ G ₁₆ T ₁₇ H ₁₈ E ₁₉ C ₂₀ O ₂₁ N ₂₂ C ₂₃ E ₂₄	C ₁₃ H ₁₄ A ₁₅ N ₁₆ I ₁₇ S ₁₈ M ₁₉ U ₂₀ N ₂₁ V ₂₂ E ₂₃ I ₂₄	M ₁ H ₂ N ₃ N ₄ E ₅ V ₆ C ₇ A ₈ I ₉ S ₁₀ U ₁₁ I ₁₂
A ₁₃ L ₁₄ E ₁₅ D ₁₆ P ₁₇ A ₁₈ S ₁₉ S ₂₀ A ₂₁ G ₂₂ E ₂₃ B ₂₄	L ₁₃ I ₁₄ N ₁₅ G ₁₆ T ₁₇ H ₁₈ E ₁₉ C ₂₀ O ₂₁ N ₂₂ C ₂₃ E ₂₄	E ₁ I ₂ O ₃ G ₄ C ₅ N ₆ L ₇ N ₈ E ₉ H ₁₀ C ₁₁ T ₁₂
E ₁₃ H ₁₄ I ₁₅ N ₁₆ D ₁₇ A ₁₈ N ₁₉ O ₂₀ L ₂₁ D ₂₂ B ₂₃ O ₂₄	A ₁₃ L ₁₄ E ₁₅ D ₁₆ P ₁₇ A ₁₈ S ₁₉ S ₂₀ A ₂₁ G ₂₂ E ₂₃ B ₂₄	S ₁ L ₂ A ₃ D ₄ E ₅ G ₆ A ₇ E ₈ B ₉ A ₁₀ S ₁₁ P ₁₂
O ₁₃ K ₁₄ C ₁₅ A ₁₆ S ₁₇ E ₁₈	E ₁₃ H ₁₄ I ₁₅ N ₁₆ D ₁₇ A ₁₈ N ₁₉ O ₂₀ L ₂₁ D ₂₂ B ₂₃ O ₂₄	N ₁ H ₂ L ₃ N ₄ B ₅ D ₆ E ₇ I ₈ O ₉ A ₁₀ O ₁₁ D ₁₂
	O ₁₃ K ₁₄ C ₁₅ A ₁₆ S ₁₇ E ₁₈	K ₁ A ₂ O ₃ C ₄ E ₅ S ₆
Transposition matrix 2a	Transposition matrix 2b	Ciphertext
J ₁ I ₂ G ₃ S ₄ A ₅ W ₆ P ₇ U ₈ Z ₉ L ₁₀ E ₁₁	A ₁ E ₂ G ₃ I ₄ J ₅ L ₆ P ₇ S ₈ U ₉ W ₁₀ Z ₁₁ Z ₁₂	E ₁ T ₂ N ₃ V ₄ U ₅ E ₆ C ₇ D ₈ I ₉ A ₁₀ E ₁₁ C ₁₂
D ₁₃ A ₁₄ C ₁₅ M ₁₆ E ₁₇ S ₁₈ N ₁₉ P ₂₀ E ₂₁ E ₂₂ H ₂₃ I ₂₄	E ₁ I ₂ C ₃ A ₄ D ₅ H ₆ N ₇ M ₈ P ₉ S ₁₀ E ₁₁ E ₁₂	C ₁ H ₂ N ₃ C ₄ K ₅ G ₆ I ₇ E ₈ E ₉ E ₁₀ T ₁₁ A ₁₂
L ₁₃ H ₁₄ K ₁₅ H ₁₆ T ₁₇ E ₁₈ N ₁₉ O ₂₀ A ₂₁ L ₂₂ L ₂₃ A ₂₄	T ₁₃ A ₁₄ K ₁₅ H ₁₆ L ₁₇ L ₁₈ N ₁₉ H ₂₀ O ₂₁ E ₂₂ A ₂₃ L ₂₄	H ₁ N ₂ E ₃ A ₄ A ₅ A ₆ I ₇ D ₈ L ₉ A ₁₀ B ₁₁ L ₁₂
A ₁₃ N ₁₄ G ₁₅ D ₁₆ N ₁₇ A ₁₈ L ₁₉ V ₂₀ M ₂₁ E ₂₂ C ₂₃ E ₂₄	N ₁₃ E ₁₄ G ₁₅ N ₁₆ A ₁₇ C ₁₈ L ₁₉ D ₂₀ V ₂₁ A ₂₂ M ₂₃ E ₂₄	E ₁ A ₂ S ₃ H ₄ L ₅ C ₆ T ₇ I ₈ S ₉ L ₁₀ N ₁₁ N ₁₂
B ₁₃ E ₁₄ I ₁₅ T ₁₆ V ₁₇ N ₁₈ G ₁₉ D ₂₀ I ₂₁ V ₂₂ T ₂₃ C ₂₄	V ₁₃ C ₁₄ I ₁₅ E ₁₆ B ₁₇ T ₁₈ G ₁₉ T ₂₀ D ₂₁ N ₂₂ I ₂₃ V ₂₄	L ₁ G ₂ D ₃ O ₄ U ₅ M ₆ H ₇ D ₈ T ₉ O ₁₀ I ₁₁ T ₁₂
L ₁₃ A ₁₄ E ₁₅ O ₁₆ U ₁₇ R ₁₈ D ₁₉ A ₂₀ N ₂₁ E ₂₂ I ₂₃ C ₂₄	U ₁₃ C ₁₄ E ₁₅ A ₁₆ L ₁₇ I ₁₈ D ₁₉ O ₂₀ A ₂₁ R ₂₂ N ₂₃ E ₂₄	P ₁ P ₂ O ₃ V ₄ D ₅ A ₆ E ₇ C ₈ S ₉ E ₁₀ A ₁₁ N ₁₂
E ₁₃ A ₁₄ E ₁₅ I ₁₆ E ₁₇ B ₁₈ O ₁₉ E ₂₀ D ₂₁ E ₂₂ S ₂₃ H ₂₄	E ₁₃ H ₁₄ E ₁₅ A ₁₆ E ₁₇ S ₁₈ O ₁₉ I ₂₀ E ₂₁ B ₂₂ D ₂₃ E ₂₄	R ₁ B ₂ R ₃ S ₄ E ₅ A ₆ M ₇ I ₈ N ₉ D ₁₀ S ₁₁ E ₁₂
A ₁₃ A ₁₄ E ₁₅ T ₁₆ C ₁₇ R ₁₈ U ₁₉ C ₂₀ S ₂₁ O ₂₂ L ₂₃ N ₂₄	C ₁₃ N ₁₄ E ₁₅ A ₁₆ A ₁₇ L ₁₈ U ₁₉ T ₂₀ C ₂₁ R ₂₂ S ₂₃ O ₂₄	L ₁ E ₂ V ₃ E ₄ E ₅ O ₆
S ₁₃ I ₁₄ T ₁₅ P ₁₆ D ₁₇ S ₁₈	D ₁₃ T ₁₄ I ₁₅ S ₁₆ P ₁₇ S ₁₈	

Rail fence cipher

- 也叫 zigzag cipher
- 在矩陣中 V 型的填入明文
- 最後水平閱讀則成密文
- 舉例 key=3
 - 明文:WEAREDISCOVEREDRUNATONCE
 - 密文:WECRUOERDSOEERNTNEAIVDAC

```
W . . . E . . . C . . . R . . . U . . . O . . .  
. E . R . D . S . O . E . E . R . N . T . N . E  
. . A . . . I . . . V . . . D . . . A . . . C .
```

Scytale

- 由左到右依序填入明文
- 由上往下閱讀成密文
- 可想成柵欄密碼的一種變體
- 舉例 key=4
 - 明文: IAMHURTVERYBADLYHELP
 - 密文: IRYYATBHMVAEHEDLURLP



	I	a	m	h	u	
—	r	t	v	e	r	—
	y	b	a	d	l	
	y	h	e	l	p	

Curve cipher

- 由左到右依序填入明文
- 曲線迂迴的方式閱讀成密文
- 可想成柵欄密碼的一種變體
- 舉例 key=7
 - 明文: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
 - 密文: GESFCINPHODTMWUQOURYZEJREHBXVALOOKT

T	h		e	q	u	i		c
k	b		r	o	w	n		f
o	x		j	u	m	p		s
o	v		e	r	t	h		e
l	a		z	y	d	o		g

Columnar transposition cipher

- 由左到右依序填入明文
- 根據密鑰的字母順序依序以列讀取成密文
- 舉例 key=HOWAREU
 - 明文:THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
 - 密文:QOURYINPHOTKOOLHBXVAUWMTDCFSEGERJEZ

h 3	o 4	w 7	a 1	r 5	e 2	u 6
T	h	e	q	u	i	c
k	b	r	o	w	n	f
o	x	j	u	m	p	s
o	v	e	r	t	h	e
l	a	z	y	d	o	g

Classical cryptography

The other ciphers

Morse code

- 國際間通用的電報編碼
- 發明者：塞繆爾·摩斯 (Samuel Morse)
- 常用統一符號/縮寫：
 - CQ: 呼叫任意站台
 - CUL: 再見
 - DE: 來自
 - K: 發送結束
 - SK: 通訊結束
 - CQD/SOS: 求救訊號

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A ● —
B — ● ● ●
C — ● — ●
D — ● ●
E ●
F ● ● — ●
G — — ●
H ● ● ● ●
I ● ●
J ● — — —
K — ● —
L ● — ● ●
M — —
N — ●
O — — —
P ● — — ●
Q — — ● —
R ● — ●
S ● ● ●
T —

U ● ● —
V ● ● ● —
W ● — —
X — ● ● —
Y — ● — —
Z — — ● ●

1 ● — — — —
2 ● ● — — —
3 ● ● ● — —
4 ● ● ● ● —
5 ● ● ● ● ●
6 — ● ● ● ●
7 — — ● ● ●
8 — — — ● ●
9 — — — — ●
0 — — — — —

Bacon's cipher

- 隱寫術的一種
- 發明者：法蘭西斯·培根 (Francis Bacon)
- 在純文本中使用兩種不同的字體對應 A 和 B
- 將真正的訊息隱藏在假訊息中
- 舉例 A=normal, B=bold：
 - 秘密：steganography
 - 文本：To encode a message each letter of the plaintext is replaced by a group of five of the letters 'A' or 'B'.

a	AAAAA	g	AABBA	n	ABBAA	t	BAABA
b	AAAAB	h	AABBB	o	ABBAB	u-v	BAABB
c	AAABA	i-j	ABAAA	p	ABBBA	w	BABAA
d	AAABB	k	ABAAB	q	ABBBB	x	BABAB
e	AABAA	l	ABABA	r	BAAAA	y	BABBA
f	AABAB	m	ABABB	s	BAAAB	z	BABBB

Pigpen cipher

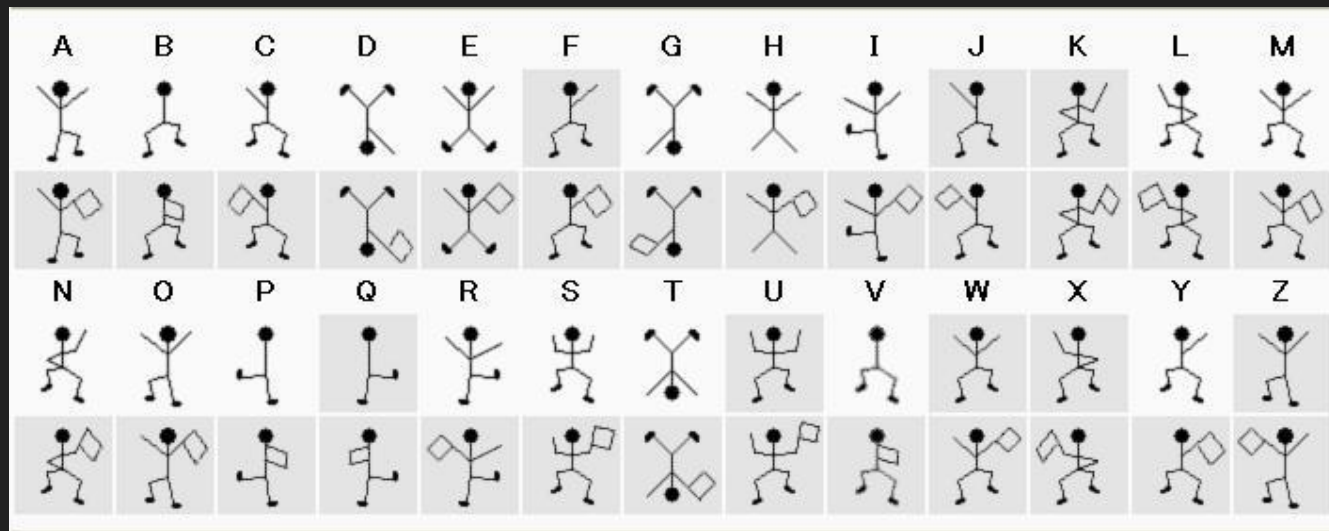
- 又稱共濟會密碼
- 以格子圖案為基礎的簡單替換式密碼

➤ ◡ ◢ ◣ ◤ ◥ ◦ ◧
X MARKS THE SPOT

A	B	C	J	K	L
D	E	F	M	N	O
G	H	I	P	Q	R
S T U V			W X Y Z		

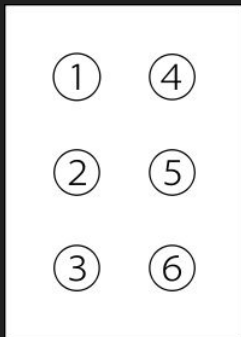
Dancing men



























- 出自於「福爾摩斯探案」的「小舞人探案」
- 以小人圖案為基礎的簡單替換式密碼



Braille

- 又稱盲文
- 台灣版



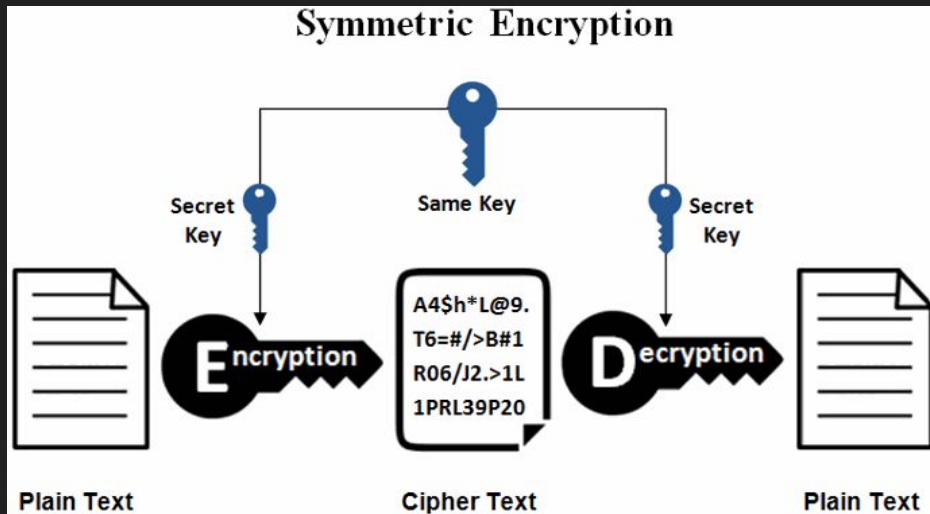
a	b	c	d	e	f	g	h	i	j
									
k	l	m	n	o	p	q	r	s	t
									
u	v	x	y	z					
									
									w
									

Symmetric-key algorithm

Stream cipher

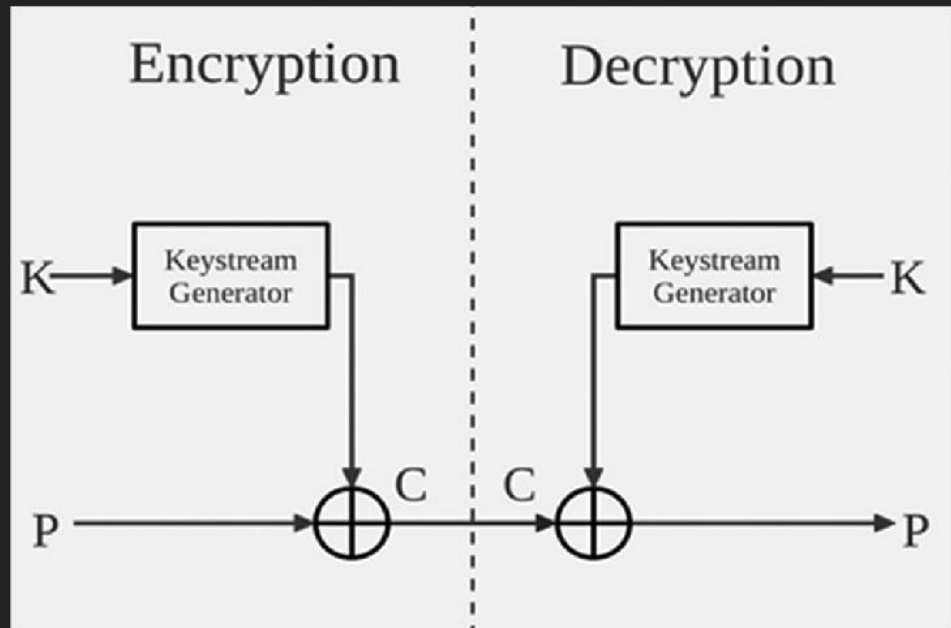
Symmetric-key algorithm

- 對稱密鑰加密
- 加解密用同一把金鑰
 - 如何安全的傳輸金鑰
- 加解密的速度要快
 - 資料可能非常龐大



Stream cipher

- 又稱資料流加密、流加密
- 通常是以 bit 為單位做 xor 運算
- 解決對稱加密的完善保密性
 - 由克勞德·夏農提出的觀點
 - 密文不應該透露任何明文的資訊
 - $P(E(m, k_i) = c) = P(E(m, k_j) = c)$



XOR (Exclusive or)

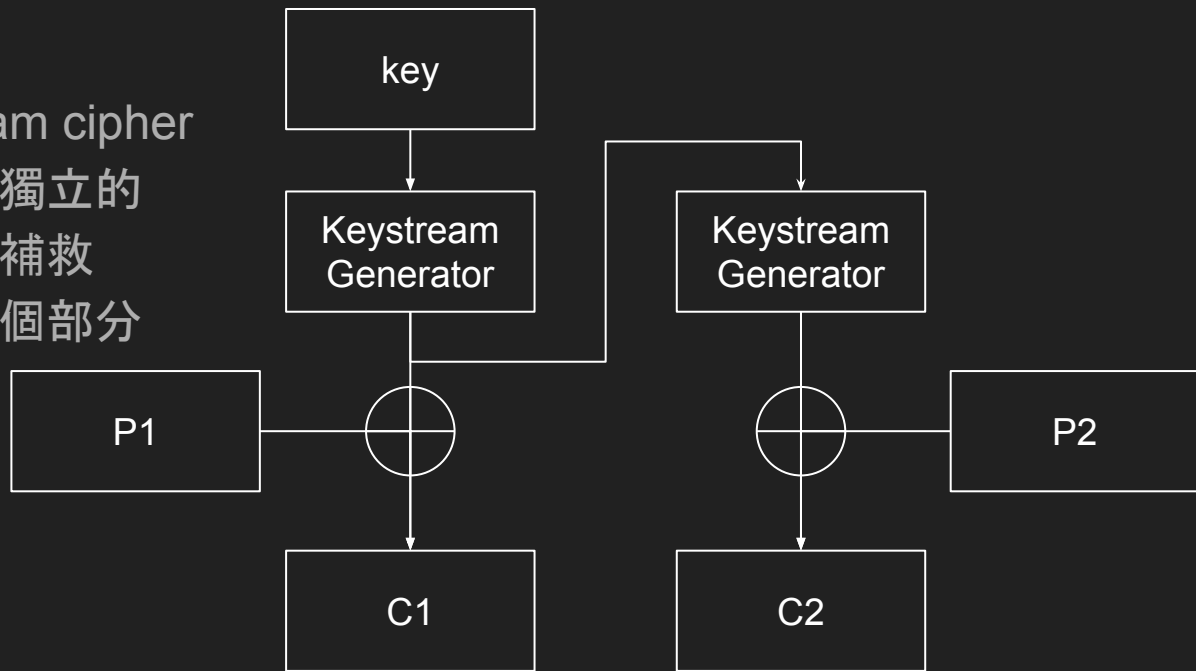


- 邏輯定義：
 - $A \text{ xor } B = (A \text{ or } B) \text{ and } (\neg A \text{ or } \neg B)$
 - $A \text{ xor } B = (A \text{ and } \neg B) \text{ or } (\neg A \text{ and } B)$
- 性質：
 - 交換律： $A \text{ xor } B = B \text{ xor } A$
 - 結合律： $(A \text{ xor } B) \text{ xor } C = A \text{ xor } (B \text{ xor } C)$
 - 恆等律： $A \text{ xor } 0 = A$
 - 歸零律： $A \text{ xor } A = 0$

A	B	$A \oplus B$
False	False	False
False	True	True
True	False	True
True	True	False

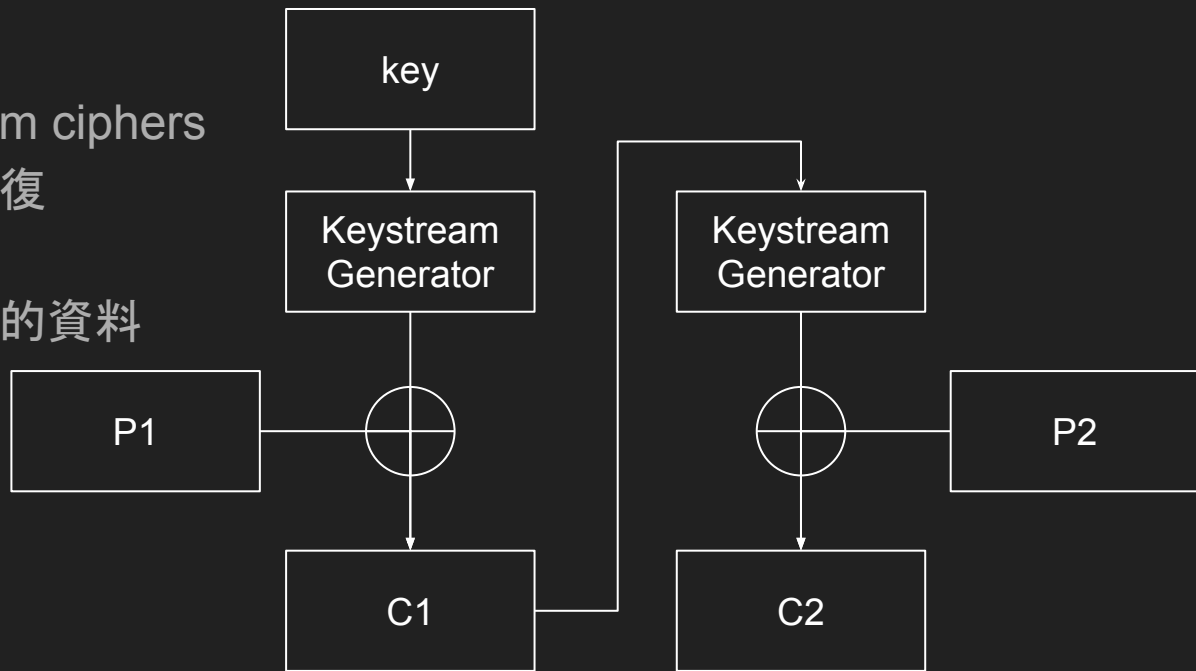
Synchronous stream ciphers

- 同步串流加密
- 又稱 binary additive stream cipher
- 對明文和密文來說金鑰是獨立的
- 資料缺漏可系統性的嘗試補救
- 損壞的部分不會影響下一個部分
- 不可平行加解密
- 造成 Bit-flipping attack



Self-synchronizing stream ciphers

- 自同步串流加密
- 又稱 asynchronous stream ciphers
- 資料有錯更容易發現和修復
- 可平行/部分加解密
- 部分損壞可能會影響往後的資料

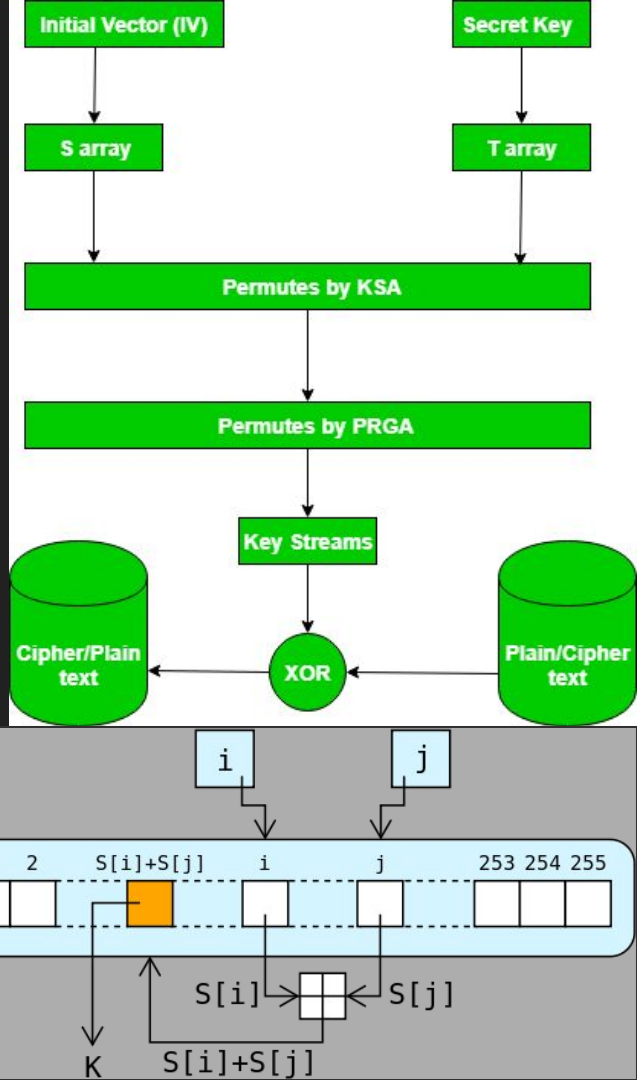


RC4 (Rivest Cipher 4)

- Synchronous stream cipher
- key size: 40 ~ 2048 bits
- 曾經在 TLS 裡支援過
 - SNAP FMS attack

```
@staticmethod
def KSA(key: list[int]) -> list[int]:
    s_box, j = list(range(256)), 0
    for i in range(256):
        j = (j + s_box[i] + key[i % len(key)]) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]
    return s_box

@staticmethod
def PRGA(s_box: list[int]) -> typing.Generator[int, None, None]:
    i, j = 0, 0
    s_box = s_box.copy()
    while True:
        i = (i + 1) % 256
        j = (j + s_box[i]) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]
        yield s_box[(s_box[i] + s_box[j]) % 256]
```



Symmetric-key algorithm

Keystream

OTP (One-Time Pad)

- RNG (Random Number Generator)
- 一次性密碼本, 必須是 Truly Random 產生
- 理論上被證明具有完善保密性
- 密碼至少要和資料一樣長
- 只能被用一次, 用完就要丟
- 傳輸密碼本的通道必須安全

CIHJT	UUHML	FRUGC	ZIBGD	BQPNI	PDNJG	LPLLP	YJYXM
DCXAC	JSJUK	BIOYT	MWQPX	DLIRC	BEXYK	VKIMB	TYIPE
UOLYQ	OKOXH	PIJKY	DRDBC	GEFZG	UACKD	RARCD	HBYRI
DZJYO	YKAIE	LIUYW	DFOHU	IOHZV	SRNDD	KPSSO	JMPQT
MHQHL	OHQQD	SMHNP	HHOHQ	GXRPI	XBVIP	LLZAA	VCMOG
AWSSZ	YMFNI	ATMON	IXPBY	FOZLE	CVYSJ	XZGPU	CTFQY
HOVHU	OCJGU	QMWQV	OIGOR	BFHIZ	TYFDB	VBRMN	XNLZC

LCG (Linear Congruential Generator)

$$X_{n+1} = (aX_n + c) \bmod m$$

- 線性同餘方法
- 運算速度快、易寫、佔用記憶體少
- 有固定的週期大小, 常用能產出較長週期
 - $m \in \text{Prime}$, $c = 0$
 - m 是 2 的指數, $c = 0$
 - $c \neq 0$ 則需要滿足以下條件
 - $\text{gcd}(c, m) = 1$
 - $\forall x \in m$ 的質因數分解, $x|(a-1)$
 - $4|(a-1)$ if $4|m$
- 有足夠的輸出可以把輸入都推算回來
 - 因此衍伸出很多變體
- PRNG (Pseudo-Random Number Generator)

LCG (Linear Congruential Generator)

$$X_{n+1} = (aX_n + c) \bmod m$$

- 不知道 c 的情況
 - $x1 = a * x0 + c \bmod m$
 - $\rightarrow c = x1 - a * x0 \bmod m$
- 不知道 a, c 的情況
 - $x1 = a * x0 + c \bmod m$
 - $x2 = a * x1 + c \bmod m$
 - $\rightarrow x1 - x2 = a * (x0 - x1) \bmod m$
 - $\rightarrow a = (x1 - x2) / (x0 - x1) \bmod m$
 - 在 \bmod 下的除法要用反模數的乘法來進行
 - `py3.8+` 支援 `pow(a, -1, p)`
 - $\rightarrow a = (x1 - x2) * \text{inverse}(x0 - x1, m) \bmod m$

LCG (Linear Congruential Generator)

$$X_{n+1} = (aX_n + c) \bmod m$$

- 不知道 a, c, m 的情況
 - $x_1 = a * x_0 + c \bmod m$
 - $x_2 = a * x_1 + c \bmod m$
 - $\rightarrow x_1 - x_2 = a * (x_0 - x_1) \bmod m = a * (x_0 - x_1) + k_1 * m$
 - $\rightarrow x_2 - x_3 = a * (x_1 - x_2) \bmod m = a * (x_1 - x_2) + k_2 * m$
 - $\rightarrow x_3 - x_4 = a * (x_2 - x_3) \bmod m = a * (x_2 - x_3) + k_3 * m$
- 思路: 需要先求出 m
 - 如果能拿到足夠多 m 的倍數 $m_1, m_2, m_3 \dots$
 - $\gcd(m_1, m_2, m_3, \dots)$ 大概率會算出 m

LCG (Linear Congruential Generator)

$$X_{n+1} = (aX_n + c) \bmod m$$

- 思路: 需要先求出 m
 - 如果能拿到足夠多 m 的倍數 $m_1, m_2, m_3 \dots$
 - $\gcd(m_1, m_2, m_3, \dots)$ 大概率會算出 m
- 先找到 m 的倍數 (需要消掉 a)
 - $t_0 = x_0 - x_1 \pmod{m}$
 - $t_1 = x_1 - x_2 = a * (x_0 - x_1) = a * t_0 \pmod{m}$
 - $t_2 = x_2 - x_3 = a * (x_1 - x_2) = a * t_1 \pmod{m}$
 - $\rightarrow t_0 * t_2 - t_1 * t_1 = t_0 * (a^2 * t_0) - (a * t_0) * (a * t_0) = 0 \pmod{m}$
- 生出一堆 m 的倍數
 - $m_1 = (x_0 - x_1) * (x_2 - x_3) - (x_1 - x_2) * (x_1 - x_2)$
 - $m_2 = (x_1 - x_2) * (x_3 - x_4) - (x_2 - x_3) * (x_2 - x_3)$
 - $m_3 = (x_2 - x_3) * (x_4 - x_5) - (x_3 - x_4) * (x_3 - x_4)$

Mersenne Twister

- 梅森旋轉算法
- 旋轉週期長度為梅森質數，因此而命名
- 程式語言常用的 PRNG (MT19937)
 - R、Python、Ruby、Free Pascal、PHP、Matlab、C++11
 - 但 lib 中實作上多少有些差異
- 生成亂數方法
 - 輸入 seed 擴展成 624 個 state
 - 以舊的 state 生成下一輪 state
 - 取 state 值加料後輸出隨機值
 - state 用完了從第二步驟重複

```
def __init__(self, seed: int):
    self.MT = [0] * 624
    self.MT[0] = seed
    self.index = 0
    for idx in range(1, 624):
        self.MT[idx] = (0x6C078965 * (self.MT[idx-1] ^ self.MT[idx-1] >> 30) + idx) & self.MASK

def generate_numbers(self):
    for idx in range(624):
        y = ((self.MT[idx] & 0x80000000) + (self.MT[(idx + 1) % 624] & 0x7FFFFFFF)) & self.MASK
        self.MT[idx] = (y >> 1) ^ self.MT[(idx + 397) % 624] ^ (0x9908B0DF * (y % 2))

def extract_number(self):
    if self.index == 0:
        self.generate_numbers()
    y = self.MT[self.index]
    y ^= y >> 11
    y ^= y << 7 & 0x9D2C5680
    y ^= y << 15 & 0xEFC60000
    y ^= y >> 18
    self.index = (self.index + 1) % 624
    return y & self.MASK
```

Mersenne Twister

- 破解方法
 - 蒐集 624 個連續的 random 值
 - 輸入成 624 個 MT state
 - 算出之後每輪的 MT state
 - 預測接下來的所有 random 值

```
import random
from mt19937predictor import MT19937Predictor
# pip install mersenne-twister-predictor
```

```
predictor = MT19937Predictor()
for _ in range(624):
    rand = random.getrandbits(32)
    predictor.setrandbits(rand, 32)
```

```
assert random.getrandbits(32) == predictor.getrandbits(32)
```

```
def __init__(self, seed: int):
    self.MT = [0] * 624
    self.MT[0] = seed
    self.index = 0
    for idx in range(1, 624):
        self.MT[idx] = (0x6C078965 * (self.MT[idx-1] ^ self.MT[idx-1] >> 30) + idx) & self.MASK

def generate_numbers(self):
    for idx in range(624):
        y = ((self.MT[idx] & 0x80000000) + (self.MT[(idx + 1) % 624] & 0x7FFFFFFF)) & self.MASK
        self.MT[idx] = (y >> 1) ^ self.MT[(idx + 397) % 624] ^ (0x9908B0DF * (y % 2))

def extract_number(self):
    if self.index == 0:
        self.generate_numbers()
    y = self.MT[self.index]
    y ^= y >> 11
    y ^= y << 7 & 0x9D2C5680
    y ^= y << 15 & 0xEFC60000
    y ^= y >> 18
    self.index = (self.index + 1) % 624
    return y & self.MASK
```

/dev/random

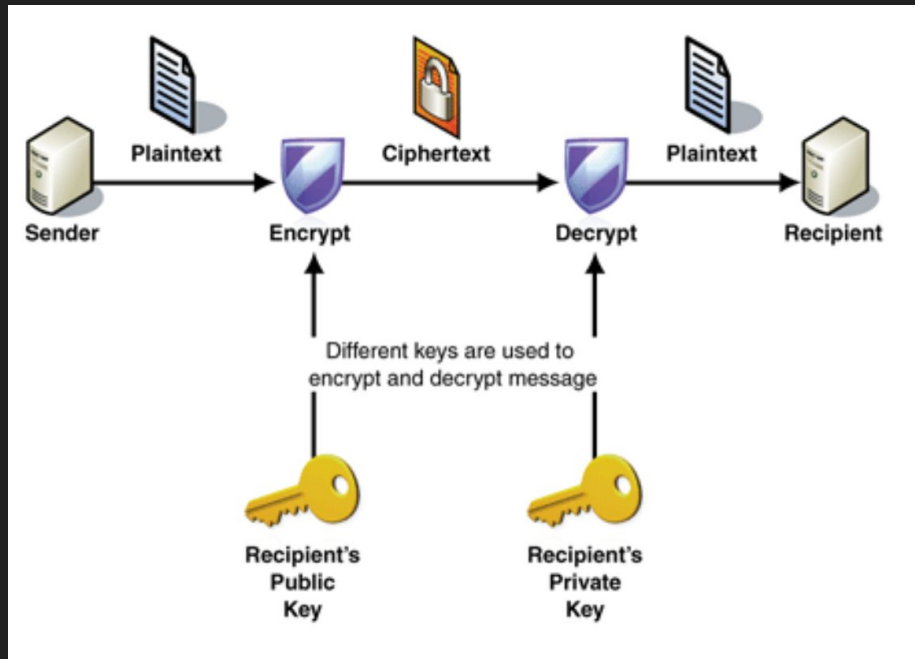
- 類 UNIX 系統中的特殊檔案, 可以做為 RNG 或 PRNG
- Linux
 - 以背景噪聲產生真正的亂數
 - 建議在生成高強度的金鑰時使用
 - 如果搜集的噪聲用完時, 調用函數就會被阻擋
 - 衍伸出 /dev/urandom (unblocked)
 - 會重複利用噪聲來當作 PRNG
 - 不建議使用在高強度的金鑰
- MacOS
- Windows

Public-key cryptography

Number theory

Public-key cryptography

- 公開金鑰密碼學
 - 又稱非對稱式密碼學
- 有公鑰和私鑰
 - 公鑰是公開的
 - 私鑰是不能公開的
 - 公鑰加密只能私鑰解密
 - 私鑰簽章只能公鑰驗證
- 應用途徑
 - TLS/SSL 連線加密
 - 金鑰交換 (key exchange)
 - 數位簽章 (Digital Signature)



Euler's totient function

$\varphi(n)$ = 小於 n 的正整數中跟 n 互質的數的個數

- $\varphi(1) = 1$
 - $\varphi(2) = 1$
 - $\varphi(3) = 2$
 - $\varphi(4) = 2$
 - $\varphi(5) = 4$
 - $\varphi(6) = 2$
 - $\varphi(7) = 6$
 - $\varphi(8) = 4$
 - $\varphi(9) = 6$
 - $\varphi(10) = 4$
 - $\varphi(11) = 10$
- $p \in \text{Prime}$
 - $\varphi(p) = p - 1$
 - $p \in \text{Prime}, k \in \mathbb{N}$
 - $\varphi(p^k) = p^k - p^{k-1}$
 - $p, q \in \text{Prime}$
 - $\varphi(pq) = \varphi(p) * \varphi(q) = (p-1) * (q-1)$



Fermat's little theorem

- 定理
 - $a \in \mathbb{N}, p \in \text{Prime}, \gcd(a, p) = 1$
 - $a^{p-1} \equiv 1 \pmod{p}$
- 證明
 - 設 $S = \{1, 2, 3, 4, \dots, p-1\}$
 - $S1 = \{\forall x \in S, ax \pmod{p}\}$
 - $S = \text{Permutation}(S1) = S1$
 - $S * a^{p-1} \equiv S \pmod{p}$
- 歐拉擴充
 - $a, n \in \mathbb{N}, \gcd(a, n) = 1$
 - $a^{\varphi(n)} \equiv 1 \pmod{n}$

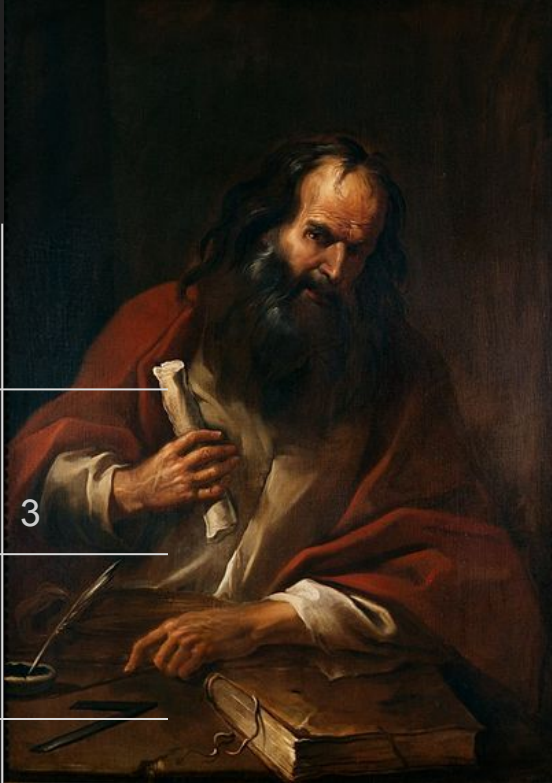


Euclidean algorithm

- $147 = 1071 - 462 * 2$
- $21 = 462 - 147 * 3$
- $\rightarrow 21 = 462 - (1071 - 462 * 2) * 3$
- $\rightarrow 21 = 462 * 7 - 1071 * 3$

- 如果 a, b 互質 ($\gcd(a, b) = 1$)
- $ax + by = \gcd(a, b) = 1$
- $\rightarrow ax \equiv 1 \pmod{b}$
- $\rightarrow x = \text{inverse}(a, b)$
- $\rightarrow x = \text{pow}(a, -1, b)$

	1071	462	
2	924		
	147	462	
		441	3
	147	21	
7	147		
	0	21	



extended Euclidean algorithm

```
def egcd(a: int, b: int) -> tuple[int, int, int]:  
    if a == 0:  
        return (b, 0, 1)  
    g, y, x = egcd(b % a, a)  
    return (g, x - (b // a) * y, y)
```

```
def inverse(a: int, b: int) -> int:  
    g, x, y = egcd(a, b) #  $ax + by = g$   
    if g == 1:  
        return x % b  
    raise ValueError("base is not invertible for the given modulus.")
```



Public-key cryptography

RSA

RSA

- 由三個人名字縮寫組成
 - Ron Rivest (羅納德 李維斯特)
 - Adi Shamir (阿迪 薩莫爾)
 - Leonard Adleman (倫納德 阿德曼)
- 安全性
 - 因數分解問題 (keysize = 2048 ~ 4096 bits)
- 安全認證
 - PKCS#1
 - ANSI X9.31
 - IEEE 1363

RSA

- 金鑰生成
 - 選擇質數因子
 - $p, q \in \text{Prime}$
 - 計算模數 n 和 φ
 - $n = pq, \varphi(n) = (p - 1) * (q - 1)$
 - 選擇公鑰指數 e
 - $e \in \mathbb{N}, 1 < e < \varphi(n), \gcd(e, \varphi(n)) = 1$
 - 計算私鑰指數 d
 - $d = \text{inverse}(e, \varphi(n))$
- 金鑰組成
 - 公鑰 = (e, n)
 - 私鑰 = (d, n)

RSA

- 金鑰組成
 - 公鑰 = (e, n)
 - 私鑰 = (d, n)
- 加解密流程
 - $c = m^e \bmod n$
 - $m = c^d \bmod n$
- 簽章驗證流程
 - $\text{sig} = H(m)^d \bmod n$
 - $H(m) = \text{sig}^e \bmod n$
- 正確性
 - $c^d \bmod n \rightarrow m^{\text{ed} \bmod \varphi(n)} \bmod n \rightarrow m \bmod n \rightarrow m$
 - $\text{sig}^e \bmod n \rightarrow H(m)^{\text{ed} \bmod \varphi(n)} \bmod n \rightarrow H(m) \bmod n \rightarrow H(m)$

Factor modulus

- when $p == q$
 - $n = pq = p^2$
 - $\varphi(n) = p^2 - p$
- twin prime
 - $n1 = pq$
 - $n2 = (p + 2) * (q + 2) = pq + 2(p + q) + 4 = n1 + 2(p + q) + 4$
 - $\rightarrow p + q = (n2 - n1 - 4) / 2$
 - $\varphi(n1) = (p - 1) * (q - 1) = pq - (p + q) + 1 = n1 - (p + q) + 1$
 - $\varphi(n2) = (p + 1) * (q + 1) = pq + (p + q) + 1 = n1 + (p + q) + 1$
- Common Factor Attack
 - p, q reuse
 - $\gcd(n_i, n_j) = p_i = p_j$
- <http://factordb.com/index.php>
- <https://github.com/bbuhrow/yafu>
- <https://www.alpertron.com.ar/ECM.HTM>

Fermat's factorization method

- 當 $|p - q|$ 很小的時候使用
- 推導
 - $a = (p + q) / 2$
 - $b = (p - q) / 2$
 - $n = (a + b) * (a - b) = a^2 - b^2 \approx a^2$
 - $a \approx \text{sqrt}(n)$
- 實作
 - 把 a 用 $\text{sqrt}(n)$ 代入並迭代
 - 測試到 $a^2 - n$ 為完全平方數
 - $b = \text{sqrt}(a^2 - n)$
 - $(p, q) = (a + b, a - b)$

```
import gmpy2

def fermat(n: int) -> tuple[int, int]:
    a = gmpy2.isqrt(n) + 1
    b = a ** 2 - n
    while not gmpy2.iroot(b, 2)[1]:
        a += 1
        b = a ** 2 - n
    b = gmpy2.iroot(b, 2)[0]
    return (a + b, a - b)
```

Pollard's p-1 algorithm

- 當 $p - 1$ 光滑 (smooth) 時使用
 - $p - 1$ 的質因數分解有很多元素
- 費馬小定理
 - $a^{p-1} \equiv 1 \pmod{p}$ ($p|n$)
 - $\rightarrow a^{k(p-1)} \equiv 1 \pmod{p}$ ($k \in \mathbb{N}$)
 - $p | \gcd(a^{k(p-1)} - 1, n)$
- 實作
 - 找一個 $b \in \mathbb{N}$ 且 $k(p - 1) | b$
 - 則 $\gcd(a^b - 1, n)$ 有可能算出來 p
 - 讓 b 用階層去迭代可以完全覆蓋到 $k(p - 1)$
 - 通常讓 $a=2$ 運算起來比較快

```
961636823328237596348218824232909922601
376289738418409379676317994599355901125
00000000000000000000000000000000000000
00000000000000000000000000000000000000
00000000000000000000000000000000000001
```

```
from Crypto.Util.number import GCD
```

```
def pollard(n: int) -> int:
    a, b = 2, 2
    while True:
        a = pow(a, b, n)
        p = GCD(a - 1, n)
        if 1 < p < n:
            return p
        b += 1
```

Williams's $p+1$ algorithm

- 當 $p + 1$ 光滑 (smooth) 時使用
 - $p - 1$ 光滑時也可以使用
 - 但計算 $p - 1$ 光滑會比 Pollard 慢
- 設 $B = p + 1$ 質因數分解的最大質數
 - 則設 $M = \{\forall x, x \in \text{Prime}, x \leq B\}$ 的積, 滿足 $p+1|M$
- 但 M 非常大且 $M \nmid n$, 所以借助盧卡斯數列 (Lucas sequence)
 - 由計算很大的 M 改成計算 $V_M(A, 1)$ 去分解 n
 - 在 $i = 1, 2, 3, \dots, M$ 的情況下做 $\gcd(V_i(A, 1) - 2, n)$ 去嘗試分解 n
- 實作細節
 - A 通常選擇不是 2 的質數, 選合數容易重複跑嘗試過的 A
 - 考慮到 $p + 1$ 可能會有重根多因子, 所以 M 會直接用 $B!$ 去跑

```
B = B or int(gmpy2.isqrt(n))
for A in tqdm.tqdm(gen_prime(), disable=True):
    v = A
    for i in tqdm.trange(1, B + 1, desc=f"A={A}"):
        v = mlucas(v, i)
        g = GCD(v - 2, n)
        if g > 1:
            return g
```

Williams's p+1 algorithm - Lucas sequence

- 盧卡斯數列有兩類

- $P, Q \in \mathbb{Z}$

- | | | |
|-----------------|-----------------|--|
| $U_0(P, Q) = 0$ | $U_1(P, Q) = 1$ | $U_n(P, Q) = P \cdot U_{n-1}(P, Q) - Q \cdot U_{n-2}(P, Q), n > 1$ |
|-----------------|-----------------|--|

- | | | |
|-----------------|-----------------|--|
| $V_0(P, Q) = 2$ | $V_1(P, Q) = P$ | $V_n(P, Q) = P \cdot V_{n-1}(P, Q) - Q \cdot V_{n-2}(P, Q), n > 1$ |
|-----------------|-----------------|--|

- 表達式

- 由遞歸式得知特徵方程為 $x^2 - Px + Q = 0$

- 得 $\lambda = (P \pm \sqrt{P^2 - 4Q}) / 2$

- 設兩根為 a, b 且判別式為 D , 則得

- $U_n(P, Q) = (a^n - b^n) / \sqrt{D}$

- $V_n(P, Q) = a^n + b^n$

Williams's p+1 algorithm - Lucas sequence

- 盧卡斯數列

- $U_n(P, Q) = (a^n - b^n) / \text{sqrt}(D)$

- $V_n(P, Q) = a^n + b^n$

$U_0(P, Q) = 0$	$U_1(P, Q) = 1$	$U_n(P, Q) = P \cdot U_{n-1}(P, Q) - Q \cdot U_{n-2}(P, Q), n > 1$
$V_0(P, Q) = 2$	$V_1(P, Q) = P$	$V_n(P, Q) = P \cdot V_{n-1}(P, Q) - Q \cdot V_{n-2}(P, Q), n > 1$

- V 的特性

- $V_{m+n} = V_m V_n - Q^n V_{m-n}$

- $V_{2n} = V_n^2 - 2Q^n$

- $V_{mn}(P, Q) = V_m(V_n(P, Q), Q^n)$

- 特殊名稱

- $U_n(1, -1)$: 斐波那契數

- $V_n(1, -1)$: 盧卡斯數

- $U_n(2, -1)$: 佩爾數

```
def mlucas(b: int, k: int) -> int:
    # It returns k-th V(b, 1)
    v1, v2 = b % n, (b ** 2 - 2) % n
    for bit in bin(k)[3:]:
        if int(bit):
            v1, v2 = (v1 * v2 - b) % n, (v2 ** 2 - 2) % n
        else:
            v1, v2 = (v1 ** 2 - 2) % n, (v1 * v2 - b) % n
    return v1
```

Common modulus attack

- 共模攻擊
- 相同 m , n 不同 e 產生不同 c
 - $m^{e_1} \bmod n = c_1$
 - $m^{e_2} \bmod n = c_2$
- 若滿足 $\gcd(e_1, e_2) = 1$, 則有線性方程滿足 $s_1 * e_1 + s_2 * e_2 = 1$
 - 注意 (s_1, s_2) 為方程上一組解 (egcd)
 - $s_1 = \text{inverse}(e_1, e_2)$
 - $s_2 = (1 - s_1 * e_1) / e_2$
- $c_1^{s_1} * c_2^{s_2} \bmod n$
 - $\rightarrow m^{e_1*s_1} * m^{e_2*s_2} \bmod n$
 - $\rightarrow m^{e_1*s_1+e_2*s_2} \bmod n$
 - $\rightarrow m$

To be continued