

# Cryptography

Part 0

By: Killua4564

# Index

- Python tools & support
- Encoding / Operator
- Classical cryptography
  - Substitution cipher
  - Transposition cipher
  - The other ciphers
- Symmetric-key algorithm
  - Keystream
  - Block mode
- Public-key cryptography
  - Elliptic-curve cryptography
- Lattice-based cryptography
  - Linear algebra
  - RSA

# Python tools

- pwntools
- pycrypto / pycryptodome
- boltons
- gmpy / gmpy2
- hashpumpy
- mersenne-twister-predictor
- owiener
- pycipher
- tqdm
- xortool
- [sagemath](#) ([docker](#)) ([online](#))

# 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

bytes	<pre>bytes(4)           # b"\x00\x00\x00\x00" bytes([4])         # b"\x04" bytes([4]) * 4      # b"\x04\x04\x04\x04" bytes([1, 2, 3, 4]) # b"\x01\x02\x03\x04"</pre>
xor	<pre>def xor(x: bytes, y: bytes) -&gt; bytes:     return bytes(i ^ j for i, j in zip(x, y))</pre>
chunk	<pre>def chunk(data: bytes, k: int = 16) -&gt; list[bytes]:     return [data[i:i+k] for i in range(0, len(data), k)]</pre>
	<pre>def chunk(data: str, k: int = 32) -&gt; list[str]:     return [data[i:i+k] for i in range(0, len(data), k)]</pre>
	<pre>from boltons import iterutils iterutils.chunked(data, block_size)</pre>

# 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

pad	<pre>from Crypto.Util.Padding import pad text = pad(text, block_size)</pre>
	<pre>def pad(text: bytes, block_size: int = 16) -&gt; bytes:     padding = block_size - (len(text) % block_size)     return text + bytes([padding]) * padding</pre>
unpad	<pre>from Crypto.Util.Padding import unpad text = unpad(text, block_size)</pre>
	<pre>def unpad(text: bytes, block_size: int = 16) -&gt; bytes:     padding = text[-1]     assert 1 &lt;= padding &lt;= block_size     assert text.endswith(bytes([padding]) * padding)     return text[:-padding]</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°) + ((° -°) + (° Θ°)) + ((o^_o) + (o^_o)) + (° 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) + (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°) [° ε°] + (° 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

[illegible]

- ! + ( ) [ ] { }

```
alert(1)
```

编码  $\Psi$

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

☐ Eval Source

```
(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+
[])[+!+[]]+(!![]+[])[+[]]+(![]+[])(!![]+[])[+[]]+(![]+[])[+[]+!+[]]+
[+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]])[!+[]+!+[]+[]+[]]+[+!+[]]+(!![]+[])(!![]+[])[+[]]+(!
[]+[])[+[]+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[
!+[]+!+[]+!+[]]+(!![]+[])[!+[]+!+[]])[!+[]+!+[]+[]+[]]
```

# 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

# Operator

# XOR (Exclusive or)

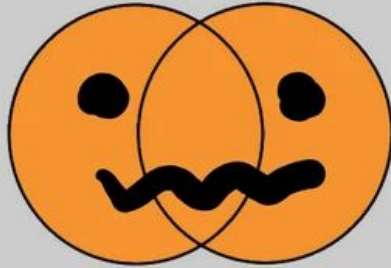


- 邏輯定義：
  - $A \text{ xor } B = (A \text{ or } B) \text{ and } (\neg A \text{ or } \neg B)$
  - $A \text{ xor } B = (A \text{ and } \neg B) \text{ or } (\neg A \text{ and } B)$
- 性質：
  - 交換律： $A \text{ xor } B = B \text{ xor } A$
  - 結合律： $(A \text{ xor } B) \text{ xor } C = A \text{ xor } (B \text{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

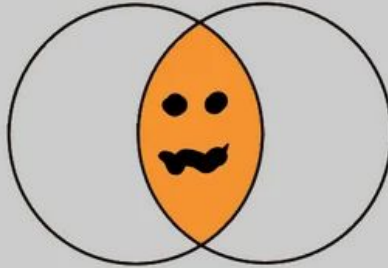
$A$	$B$	$A \oplus B$
False	False	False
False	True	True
True	False	True
True	True	False

# Operator

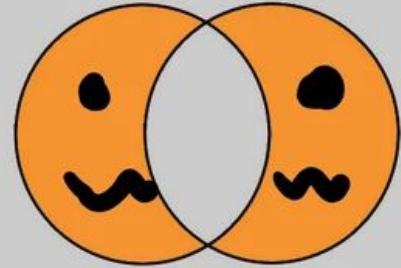
Trick OR Treat



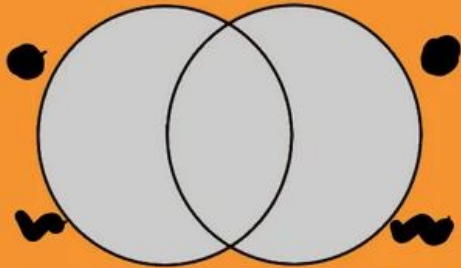
Trick AND Treat



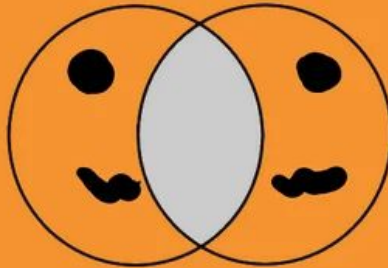
Trick XOR Treat



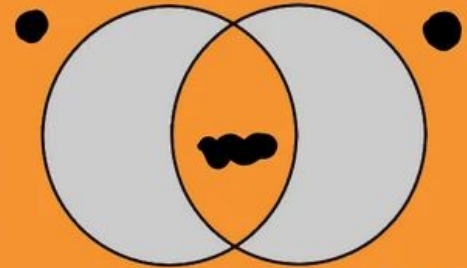
Trick NOR Treat



Trick NAND Treat



Trick XNOR Treat

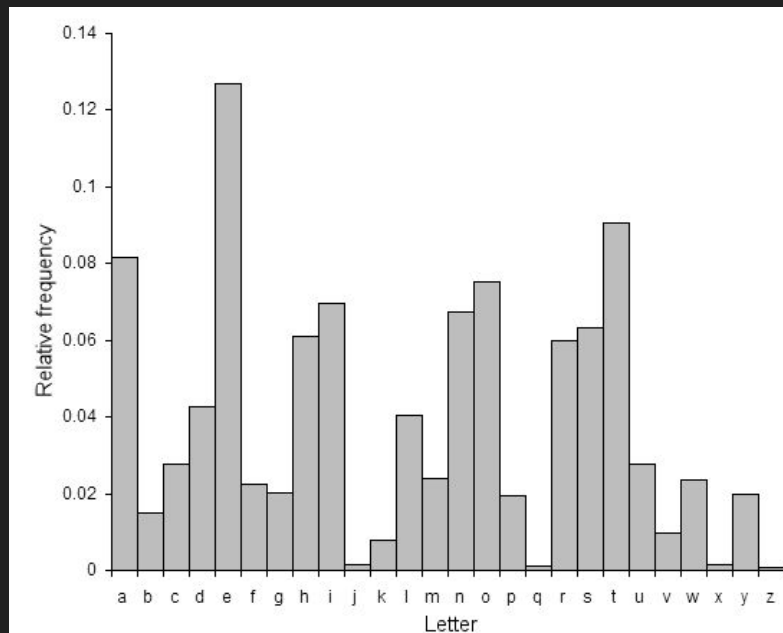


# Classical cryptography

Substitution cipher

# Substitution cipher

- 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 cipher



# Transposition cipher

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

# 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'.

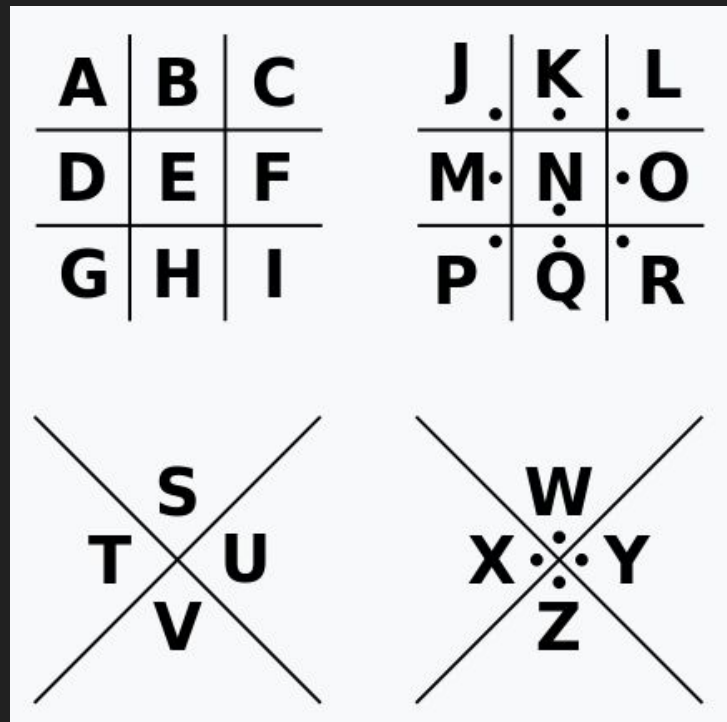
a	AAAAA	g	AABBA	n	ABBAA	t	BAABA
b	AAAAB	h	AABBB	o	ABBAB	u-v	BAABB
c	AAABA	i-j	ABAAA	p	ABBBA	w	BABAA
d	AAABB	k	ABAAB	q	ABBBB	x	BABAB
e	AABAA	l	ABABA	r	BAAAA	y	BABBA
f	AABAB	m	ABABB	s	BAAAB	z	BABBB



# Pigpen cipher

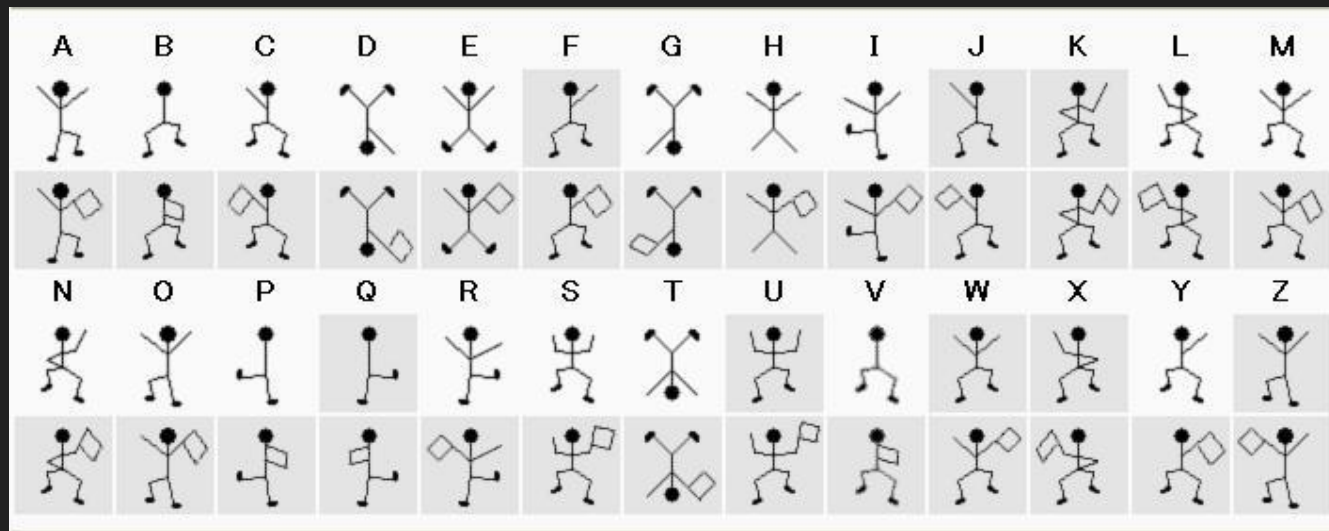
- 又稱共濟會密碼
- 以格子圖案為基礎的簡單替換式密碼

➤ ◡ ◢ ◣ ◤ ◥ ◦ ◧  
X MARKS THE SPOT



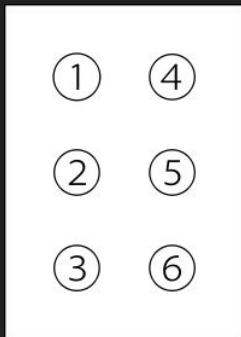
# Dancing men



























- 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



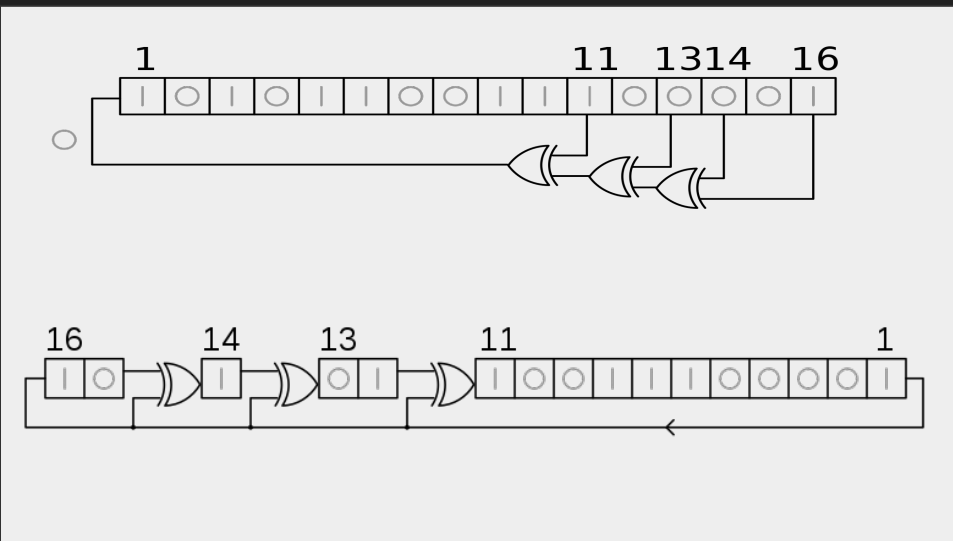
a	b	c	d	e	f	g	h	i	j
									
k	l	m	n	o	p	q	r	s	t
									
u	v	x	y	z					
									
									w
									

# Symmetric-key algorithm

Keystream

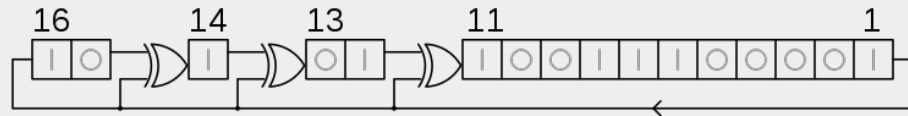
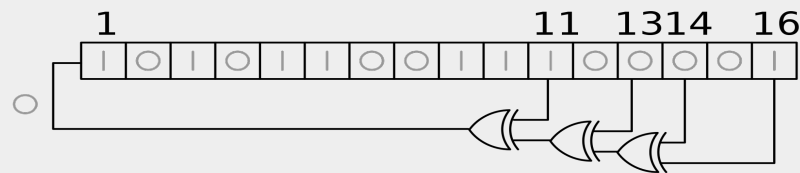
# LFSR (Linear Feedback Shift Register)

- 線性反饋移位暫存器
  - Fibonacci LFSR
  - Galois LFSR
  - Xorshift LFSR
- 一次 output 一個 bit
- 暫存器大小固定
  - 有很長的循環週期
- 在  $GF(2)$  底下矩陣乘法
- 找 taps 的方法
  - [berlekamp\\_massey](#)
- PRNG (Pseudo-Random Number Generator)



# LFSR (Linear Feedback Shift

Bits (n)	Feedback polynomial	Taps	Taps (hex)	Period ( $2^n - 1$ )
2	$x^2 + x + 1$	11	0x3	3
3	$x^3 + x^2 + 1$	110	0x6	7
4	$x^4 + x^3 + 1$	1100	0xC	15
5	$x^5 + x^3 + 1$	10100	0x14	31
6	$x^6 + x^5 + 1$	110000	0x30	63
7	$x^7 + x^6 + 1$	1100000	0x60	127
8	$x^8 + x^6 + x^5 + x^3 + 1$	10110100	0xB4	255
9	$x^9 + x^5 + 1$	100010000	0x110	511
10	$x^{10} + x^7 + 1$	1001000000	0x240	1,023
11	$x^{11} + x^9 + 1$	10100000000	0x500	2,047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	111000001000	0xE08	4,095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	1110010000000	0x1C80	8,191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	11100000000010	0x3802	16,383
15	$x^{15} + x^{14} + 1$	110000000000000	0x6000	32,767
16	$x^{16} + x^{15} + x^{13} + x^4 + 1$	1101000000001000	0xD008	65,535
17	$x^{17} + x^{14} + 1$	10010000000000000	0x12000	131,071
18	$x^{18} + x^{11} + 1$	100000010000000000	0x20400	262,143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	1110010000000000000	0x72000	524,287
20	$x^{20} + x^{17} + 1$	10010000000000000000	0x90000	1,048,575
21	$x^{21} + x^{19} + 1$	101000000000000000000	0x140000	2,097,151
22	$x^{22} + x^{21} + 1$	1100000000000000000000	0x300000	4,194,303
23	$x^{23} + x^{18} + 1$	10000100000000000000000	0x420000	8,388,607
24	$x^{24} + x^{23} + x^{22} + x^{17} + 1$	111000010000000000000000	0xE10000	16,777,215



```
def next(self) -> int:
    bit: int = functools.reduce(lambda x, y: x ^ y, (
        (self.register >> (self.SIZE - tap)) & 1
        for tap in self.taps
    ))
    ret: int = self.register & 1
    self.register = (bit << (self.SIZE - 1)) | (self.register >> 1)
    return ret
```

```
self.mask = 0
for tap in self.taps:
    self.mask |= (1 << (self.SIZE - tap))
```

```
def next(self) -> int:
    ret: int = self.register & 1
    self.register >>= 1
    self.register ^= ret * self.mask
    return ret
```

# Xorshift RNG (Shift-Register Generator)

- 又稱 Shift-Register Generators
- PRNG (Pseudo-Random Number Generator)
- 運算速度快 程式碼簡單
- 定義：
  - $\text{message} = \text{message} \text{ xor } (\text{message} \ll a)$
  - $\text{message} = \text{message} \text{ xor } (\text{message} \gg b)$
  - $\text{message} = \text{message} \text{ xor } (\text{message} \ll c)$
- 在 GF(2) 底下矩陣乘法

```
def xorshift(message: int) -> int:
    message ^= (message << 13) & MASK
    message ^= (message >> 7) & MASK
    message ^= (message << 17) & MASK
    return message
```

# Xorshift RNG (Shift-Register Generator)

```
def challenge():
    problem = ""
    nonce = bytes_to_long(os.urandom(8))
    for idx in range(200):
        nonce = xorshift(nonce)
        if bytes_to_long(os.urandom(1)) >= 128:
            problem += str(nonce & 1)
        else:
            problem += "."

    print(f"Problem: {problem}")
    if nonce != int(input("Answer: ")):
        print("Wrong answer.")
        return

    print("Correct answer.")
    print(f"Here is your flag: {FLAG}")
```

```
def xorshift(message: int) -> int:
    message ^= (message << 13) & MASK
    message ^= (message >> 7) & MASK
    message ^= (message << 17) & MASK
    return message
```



# Xorshift RNG (Shift-Register Generator)

```
F = GF(2)                                # compute addition operation on GF(2)
M = identity_matrix(F, 64)               # denote each bits situation in each row
SHL = companion_matrix([F(0)] * 64 + [F(1)], format="left")  # denote shift left for one bit
SHR = companion_matrix([F(0)] * 64 + [F(1)], format="right") # denote shift right for one bit

M += M * SHL ** 13
M += M * SHR ** 7
M += M * SHL ** 17

W, A, B = M, [], []
for binary in list(state):
    if binary != ".":
        A.append(W.T[0])                 # leak last bit situation
        B.append(Integer(binary))        # leak last bit result
    W *= M

A, B = Matrix(F, A).T, vector(F, B)
S = A.solve_left(B)                     # SA = B, type(S) = Vector_mod2_dense
print("origin:", ZZ(list(S), 2))
```

# Xorshift RNG (Shift-Register Generator)

- 全程在  $GF(2)$  底下進行加法運算
- 用 identity matrix 表示每個 bit 的初始值, 並進行運算
  - $M[i] \rightarrow$  表示初始值第  $i$  個 bit 為 1 的情況
  - $M[i][j] \rightarrow$  表示初始值第  $i$  個 bit 為 1 時第  $j$  個 bit 的結果
- 用 companion matrix 來表示運算的過程
  - $\text{format}=\text{left} \rightarrow$  表示向左位移一個 bit
  - $\text{format}=\text{right} \rightarrow$  表示向右位移一個 bit
- 把有線索的部分, 對應的條件假設和運算結果搜集起來
  - 包成 Matrix 和 vector 後用 solve\_left 求初始值的解

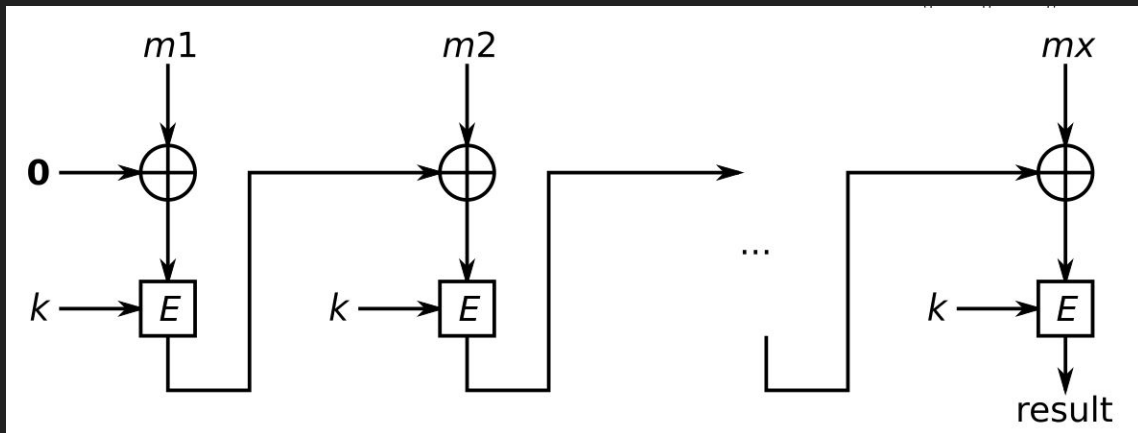
# Symmetric-key algorithm

Block mode

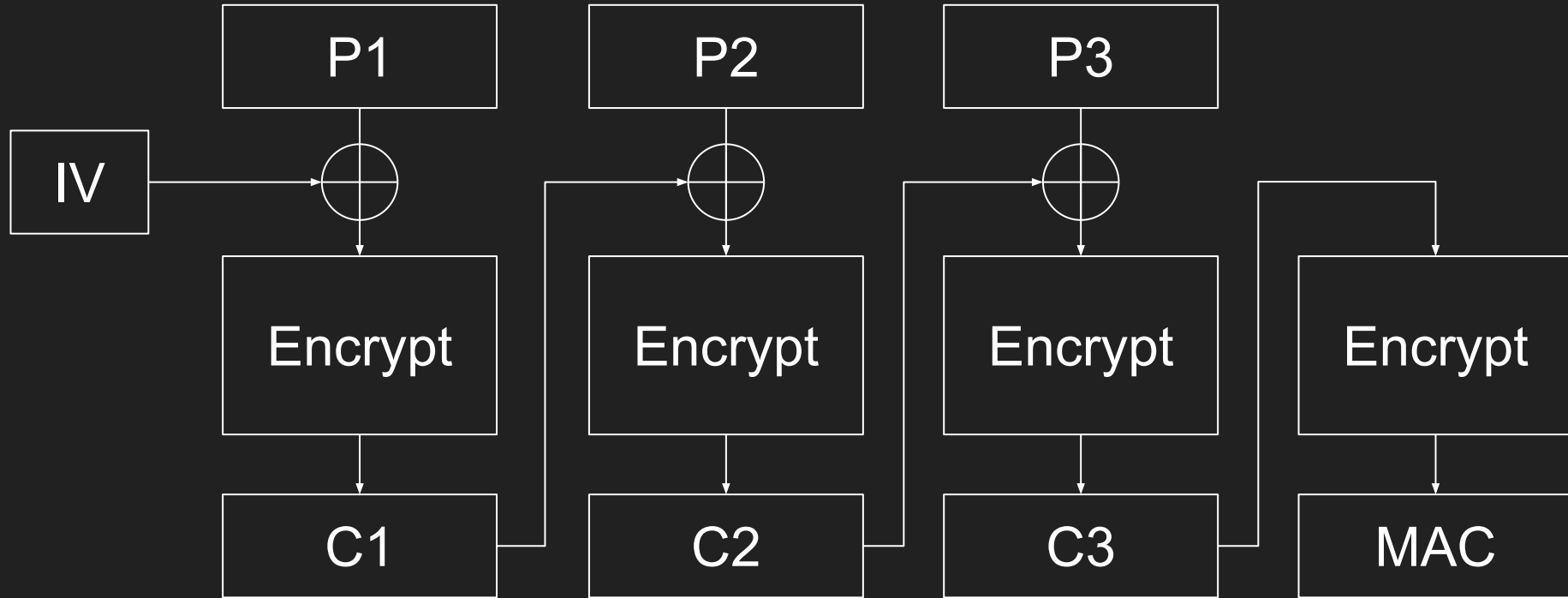
# CBC-MAC

- MAC (Message Authentication Code)
- 輸入 message 先用 CBC mode 加密
- 將最後一個 block 再用 ECB mode 加密一次
- 其中 CBC 和 ECB mode 的 key 是相同的

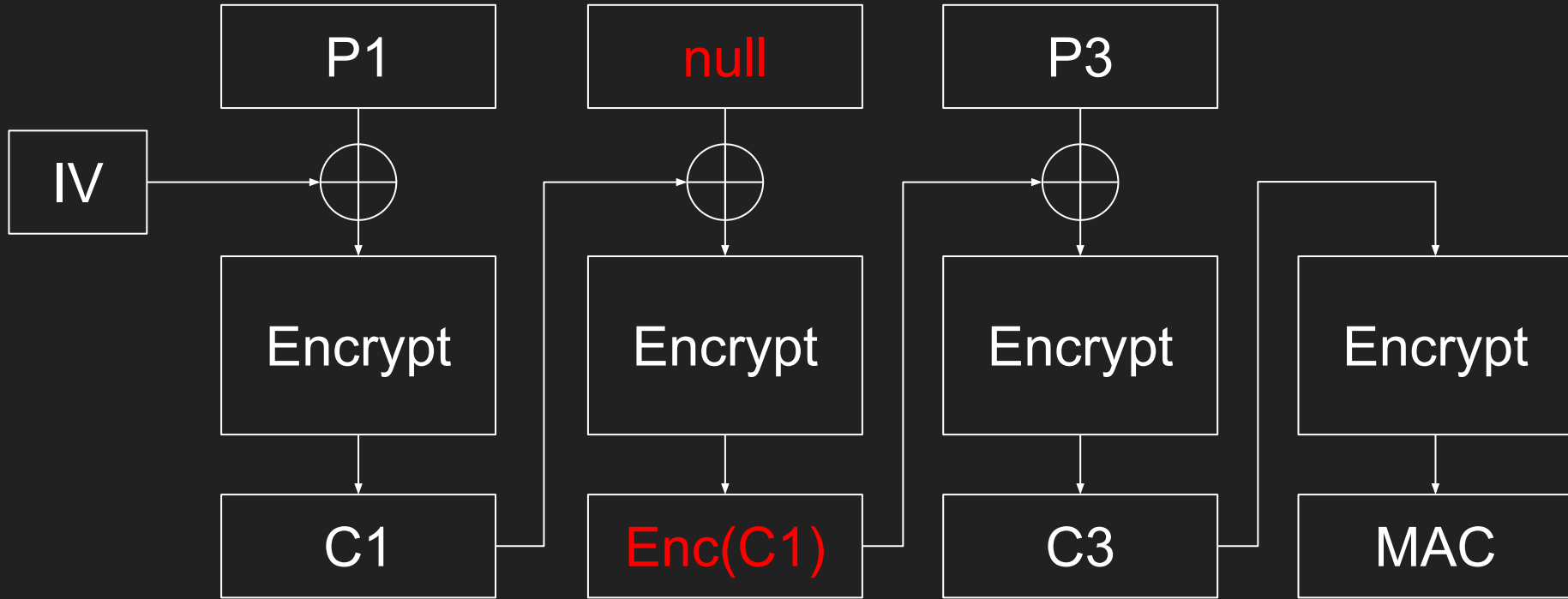
```
def generate_cbc_mac(text: bytes) -> bytes:  
    cipher = AES.new(key, AES.MODE_CBC, iv)  
    rawmac = cipher.encrypt(pad(text))  
    cipher = AES.new(key, AES.MODE_ECB)  
    cbcmac = cipher.encrypt(rawmac[-16:])  
    return cbcmac
```



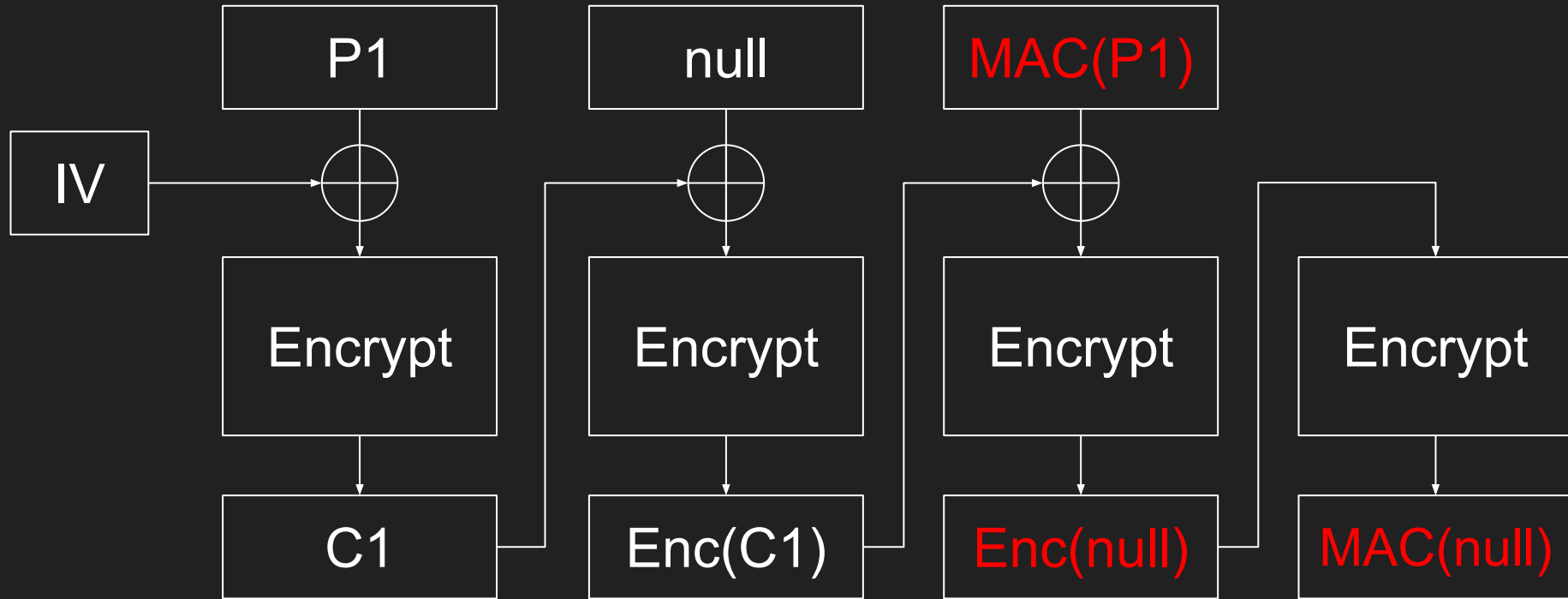
# CBC-MAC



# CBC-MAC

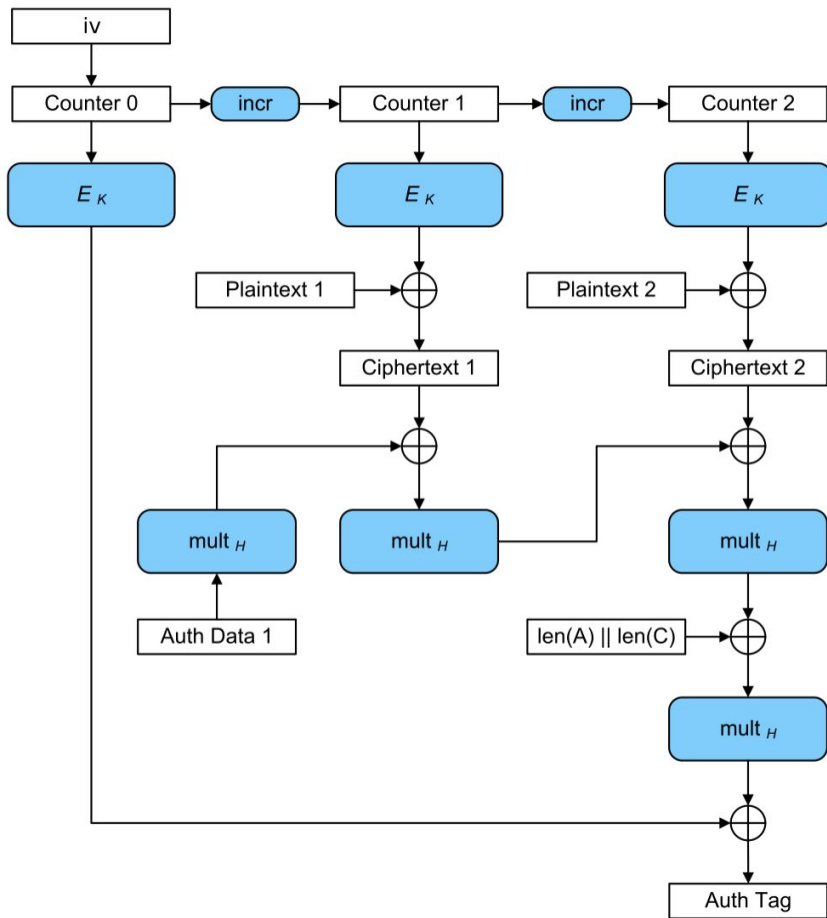


# CBC-MAC



# GCM mode (Galois/counter)

- 伽羅瓦/計數器模式
- 比 CTR mode 多了身份驗證
- 加解密都可平行, 也可以隨機讀取
  - 但 Auth Tag 需要等密文
- $\text{mult}_H$  是在  $\text{GF}(2^{128})$  底下做乘法
  - H 用 key 去做 GHASH 得到
- 重複使用 iv 會有 forbidden attack
  - 能拿到任意明文加密結果
  - 就可以偽造任何人的簽章





# GCM mode - forbidden attack

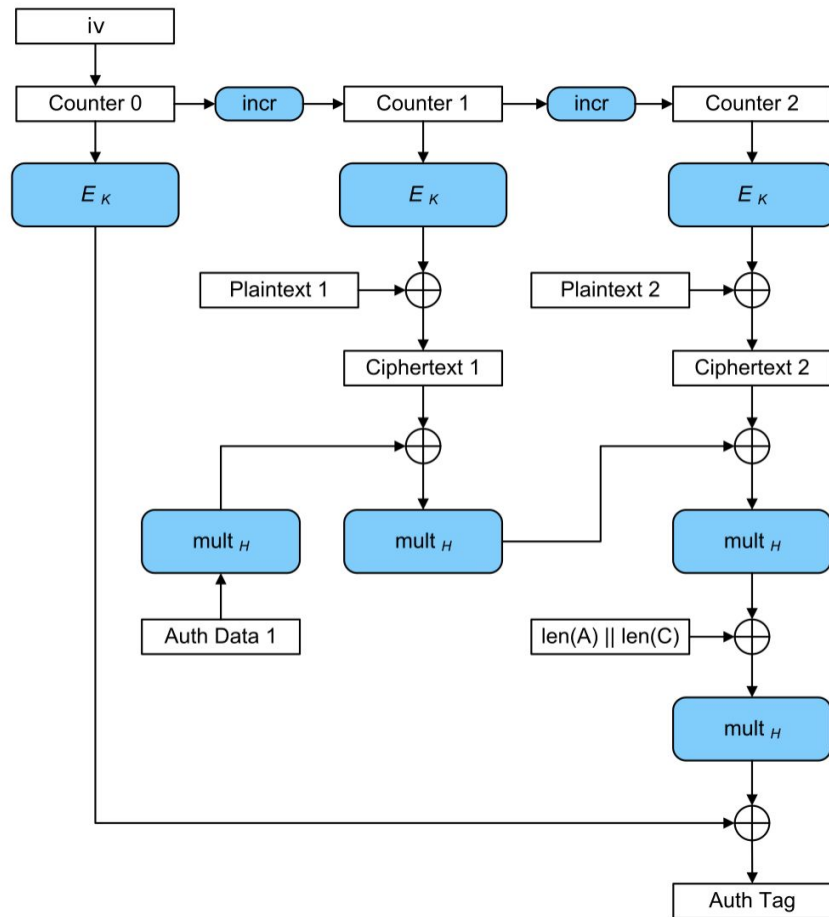
$$\text{Tag} = AH^4 + C1 \cdot H^3 + C2 \cdot H^2 + LH + E(J_0)$$

使用兩個已知明文加密+簽章會得到

$$T1 = A1 \cdot H^4 + C11 \cdot H^3 + C12 \cdot H^2 + LH + E(J_0)$$

$$T2 = A2 \cdot H^4 + C21 \cdot H^3 + C22 \cdot H^2 + LH + E(J_0)$$

相減消掉  $E(J_0)$  後用 sagemath 求出  $H$   
帶回原式算出  $E(J_0)$  後即可任意偽造



# Public-key cryptography

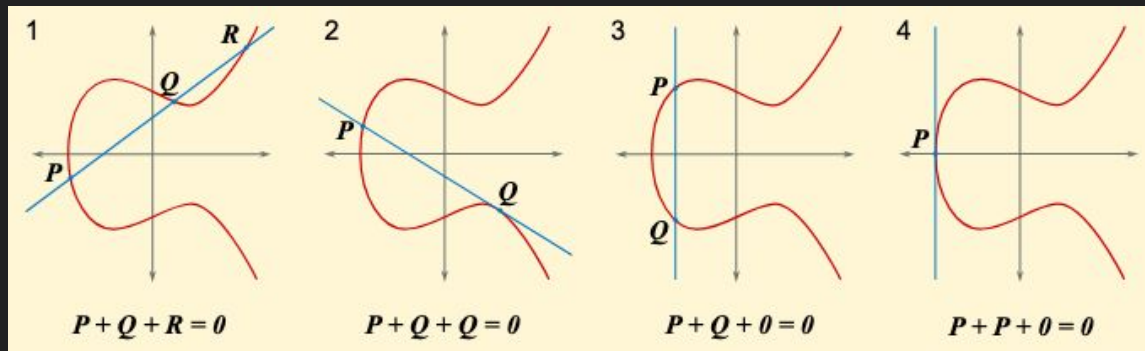
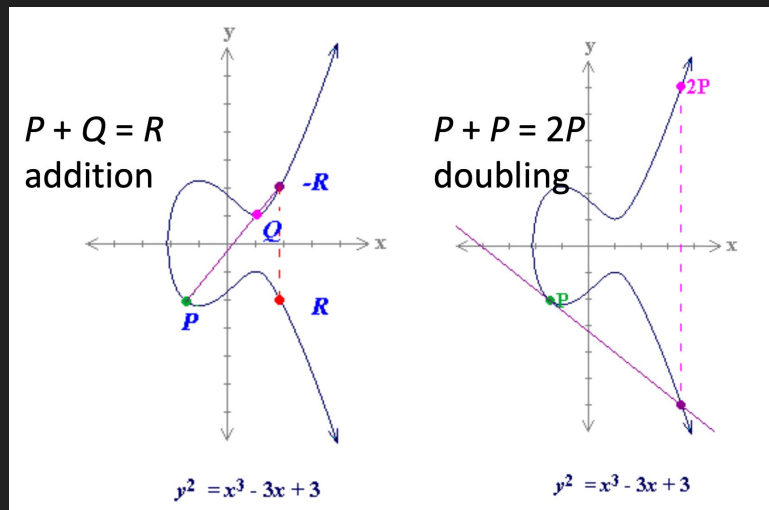
Elliptic-curve cryptography

# ECC (Elliptic-curve cryptography)

- 非對稱密碼學、公開金鑰加密系統
- 跟 RSA 對比
  - 較少 bits 數可達到相同安全等級
    - $\text{RSA}_{1024} = \text{ECC}_{160}$
    - $\text{RSA}_{2048} = \text{ECC}_{224}$
    - $\text{RSA}_{3072} = \text{ECC}_{256}$
  - 運算速度較快
  - 加密/簽章長度較小
- 安全性
  - 橢圓曲線離散對數
- 可以跟其他非對稱的算法結合
  - ECDH、ECIES、ECDSA...

# Elliptic-curve cryptography

- 方程式  $E(x, y): y^2 = x^3 + ax + b$ 
  - 判別式  $D = 4a^3 + 27b^2$
  - $D \neq 0$  防止曲線退化
- 曲線上加法運算
  - 將兩點連成一線，經過的第三點對稱於  $x$  軸，則為加法結果
  - 若沒經過第三點，則結果為原點（或稱無窮遠點）



# Elliptic-curve cryptography

- 方程式  $E(x, y): y^2 = x^3 + ax + b$
- 曲線上加法運算
  - 將兩點連成一線，經過的第三點對稱於  $x$  軸，則為加法結果
  - 若沒經過第三點，則結果為原點 (或稱無窮遠點)
  - 計算斜率  $\lambda$ 
    - 如果  $P \neq Q$  則  $\lambda = (y_1 - y_2) / (x_1 - x_2)$
    - 如果  $P = Q$  則  $\lambda = (3 * x_1^2 + a) / (2 * y_1)$
  - 找第三點  $(x_3, y_3) = (x, -y)$ 
    - $x_3 = \lambda^2 - x_1 - x_2$
    - $y_3 = \lambda * (x_1 - x_3) - y_1$
- 定義了加法運算，則可以用 [double-and-add](#) 方法做到乘法運算

# ECC (Elliptic-curve cryptography)

- 參數
  - 定義曲線 (a, b)
  - 定義場域 p
  - 曲線上的基點 G
  - 曲線的 order
- 金鑰生成
  - 選擇私鑰  $1 < d < p$
  - 計算公鑰  $Q = dG$
- 橢圓曲線離散對數 (ECDLP)
  - 給定 (P, Q) 且  $P = dQ$  求 d

E:  $y^2 = x^3 + 12x + 1$  over GF(23)

**generator**  $\rightarrow P = (0, 1)$

$2P = (13, 13)$     $3P = (5, 5)$

$4P = (3, 15)$     $5P = (6, 17)$

$6P = (19, 2)$     $7P = (17, 9)$

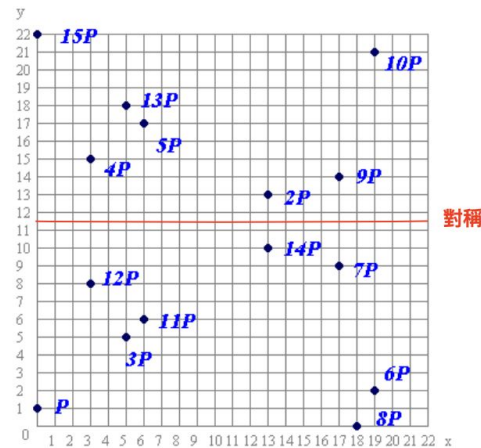
$8P = (18, 0)$     $9P = (17, 14)$

$10P = (19, 21)$     $11P = (6, 6)$

$12P = (3, 8)$     $13P = (5, 18)$

$14P = (13, 10)$     $15P = (0, 22)$

$16P = (0, 1) \rightarrow$  **order** is 16



# ECC (Elliptic-curve cryptography)

- 缺陷
  - ECC
    - smooth order
    - small private key
    - $\#E(F_p) = p$ 
      - [SmartAttack](#)
    - singular curve
      - [solve ECDLP](#)
  - ECDSA
    - reuse nonce
    - bounded nonce
      - [Lattice Attacks on ECDSA \(HNP\)](#)

# Lenstra elliptic-curve factorization

- 又稱 elliptic-curve factorization method (ECM)
- 是一種速度為 sub-exponential, 運用 ECC 做因數分解的方法
- 原理
  - 在  $Z_{\text{mod}}(n)$  底下隨機找 ECC 做  $P$  的運算
    - 分別計算  $(2!)P$ 、 $(3!)P$ 、 $(4!)P$ 、...、 $(B!)P$
  - 在計算斜率  $s$  時做的 inverse 順便檢查有沒有  $\gcd > 1$  產生
    - 可想成 Pollard's  $p-1$  的擴展找因數方式
    - 所以如果  $p-1$  的因數都很大的話會容易失敗
- 實作
  - [ECM online](#)
  - [cm\\_factor.sage](#)

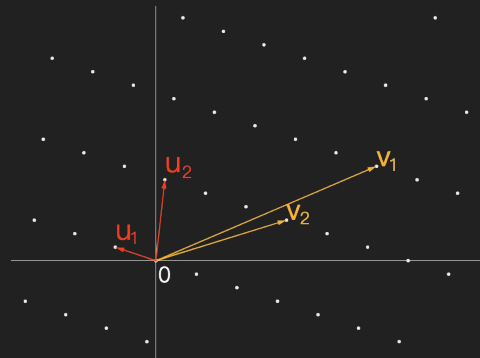


# Lattice-based cryptography

Linear algebra

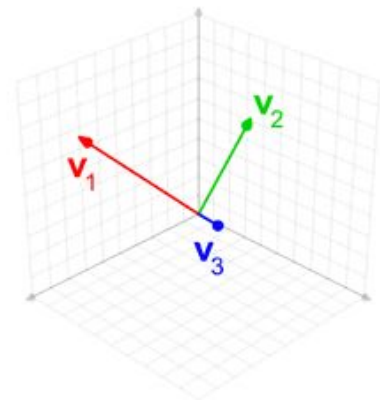
# Lattice-based cryptography

- 一個  $L$  是由許多 basis 向量 ( $b_1, b_2, \dots, b_n$ ) 所有線性組合的點的集合
  - $L \subset \mathbb{R}^n$ ,  $L = \{\sum (a_i * b_i), a_i \in \mathbb{Z}\}$
  - 舉例:  $(3, 1, 4), (1, 5, 9), (2, -1, 0)$  這三個 basis 會組成一個  $L \subset \mathbb{Z}^3$
- 一組 basis 只能組成一個  $L$ , 但同一個  $L$  不一定是同一組 basis
- 把 basis 組合成一個矩陣就能計算這個  $L$  的 volume
  - 也就是  $\text{Vol}(L) = |\det(B)| = |\det([b_1, b_2, \dots, b_n])|$
- 難題
  - SVP (最短向量問題)
    - 在給定的  $L$  中找長度非零且最短的向量
  - CVP (最近向量問題)
    - 給定一個同空間但不屬於  $L$  的向量  $w$
    - 在  $L$  中找到一個向量  $v$  使得  $\|v - w\|$  最短 (最靠近  $L$ )



# Gram-Schmidt

- 在給定 basis 的空間中，找到一組正交 basis
  - 即這組 basis 中所有向量都是垂直的
- 概念
  - 利用投影原理的基礎上找到正交 basis
  - 製造  $b^* = b - \text{proj}_a b$ , 則  $(a, b)$  為正交
- 算法
  - 以  $v_1$  為基礎讓其他向量做投影,  $u_1 = v_1$
  - 讓  $v_2$  對  $u_1$  正交成  $u_2 = v_2 - \text{proj}_{u_1} v_2$
  - 讓  $v_3$  對  $(u_1, u_2)$  正交成  $u_3 = v_3 - \text{proj}_{u_1} v_3 - \text{proj}_{u_2} v_3$
  - 讓  $v_n$  對  $(u_1, u_2, \dots, u_{n-1})$  正交成  $u_n = v_n - \sum (\text{proj}_{u_i} v_n \text{ for } 1 \leq i < n)$



# Gaussian reduction

- 將給定的 basis 減少成近似 SVP 的解
  - 高斯法只能用於二維
  - 此方法類似於 GCD 的算法
- 算法
  - 如果  $\|v_1\| > \|v_2\|$  則交換  $(v_1, v_2)$
  - 計算要減少的倍數  $m = \text{floor}(v_1 \cdot v_2 / v_1 \cdot v_1)$
  - 如果  $m = 0$  則回傳  $(v_1, v_2)$
  - 將  $v_2$  減少,  $v_2 = v_2 - m * v_1$

# LLL algorithm

- 給定更高維的 basis 減少成近似 SVP 的解
  - 在 300 維以內效率都不錯
- 算法
  - 給定 basis 集合  $B$  和參數  $\delta$  滿足  $1/4 < \delta < 1$  (預設 3/4)
  - 將  $B$  正交化成  $B^*$
  - 對於每個  $b_i$  做檢查  $1 < i \leq n$ 
    - 如果倍數  $m_{i,j} = b_i \cdot b_j^* / b_j^* \cdot b_j^* > 1/2$  則減少  $b_i$  的大小
      - $b_i = b_i - \text{floor}(m_{i,j}) \cdot b_j$  且對於新的  $B$  更新正交化的  $B^*$
    - 如果不滿足  $\delta \|b_{i-1}^*\|^2 \leq \|b_i^*\|^2 + m_{i,i-1} \|b_{i-1}^*\|^2$  (Lovász 條件)
      - 交換  $(b_i, b_{i-1})$  並更新  $B^*$

# LLL algorithm

- 給定更高維的 basis 減少成近似 SVP 的解
  - 在 300 維以內效率都不錯
- 減少後的 basis 大小上限
  - $\|b_1\| \leq (2/(4\delta-1)^{1/2})^{(n-1)/2} \det(L)^{1/n} = 2^{(n-1)/4} \det(L)^{1/n}$
- 實作
  - [sage.matrix.matrix\\_integer\\_dense.Matrix\\_integer\\_dense.LLL](#)

# Lattice-based cryptography

RSA

# Coppersmith method

- 由 Don Coppersmith 提出在單/雙變數多項式環下找小根 (small roots) 的方法
- 針對 RSA 攻擊的一個類別 Coppersmith's attack 就是基於此方法去實踐
  - 通常是針對低加密指數 (Low public exponent) 做攻擊
    - 低解密指數 (Low private exponent) 可以用 Wiener's Attack
- 問題
  - 在多項式  $f(x) = x^{\delta} + \dots + c \pmod{N}$  找小於上界  $X$  的所有小根  $x_0$
- 解法
  - 先引入一個參數  $\beta$ , 使得  $b \geq N^{\beta}$ , 這邊可以先預設  $\beta = 1$
  - 在模運算中求根很難, 就算  $b$  再小也都是一件難事
  - 但如果能找到一個在  $\mathbb{Z}\mathbb{Z}$  底下的  $f'(x)$  滿足  $f(x_0) = f'(x_0)$  問題就簡單許多
    - 可以將問題轉換成牛頓法 (Newton's method) 來解決



# Coppersmith method

- 解法 (續)
  - 定義  $f'(x)$ 
    - 定義兩個未知整數  $(m, t)$ , 生成一堆  $f(x)$  的變體當成  $g(x)$  集合
    - $g_{mi+j}(x) = x^j * N^{m-i} * f^i(x)$ , for  $0 \leq i < m, 0 \leq j < \delta$
    - $g_{m\delta+i}(x) = x^i * f^m(x)$ , for  $0 \leq i < t$
    - $f'(x) = \sum(a_i g_i(x))$ ,  $a_i \in \mathbb{Z}$
  - 因為所有  $g_i(x)$  都是由  $f(x)$  變體而來, 所以滿足  $f(x_0) = g_i(x_0) = 0 \pmod{b^m}$
  - 又因為  $f'(x)$  是  $g_i(x)$  的線性組合, 所以也滿足  $g_i(x_0) = f'(x_0) = 0 \pmod{b^m}$ 
    - 如果把  $g_i(x)$  看成 lattice  $L$  的 basis, 那  $f'(x)$  就是  $L$  上的一個向量
    - 又如果也滿足  $|f'(x_0)| < b^m$  的話, 則  $f'(x)$  是  $L$  上的 SVP
      - 可以將問題轉換成使用 LLL 解法
  - 解出  $f'(x)$  的根後留下滿足  $\gcd(N, f(x_0)) \geq N^\beta$  的根作為小根

# Coppersmith method

- 證明

- $|f'(x_0)| = \sum (c_i * x_0^i) \leq \sum (|c_i * x_0^i|) \leq \sum (|c_i * X^i|)$   
 $= \text{sqrt}(\sum (|c_i * X^i|)^2) \leq \text{sqrt}(n * \sum (|c_i * X^i|^2))$  (柯西不等式)  
 $= \text{sqrt}(n) * \|f'(xX)\|$ 
  - 問題轉換成  $\|f'(xX)\| < b^m / \text{sqrt}(n)$  則滿足  $|f'(x_0)| < b^m$
  - 而實作中真正帶入 LLL 算法的是  $f'(xX)$
  - 所以需要讓 LLL 算出來的向量  $v$  滿足  $\|v\| \leq \|f'(xX)\|$
- 已知  $\|v\| \leq 2^{(n-1)/4} \det(L)^{1/n}$ , 然而  $\det(L) = N^{\delta m(m+1)/2} X^{n(n-1)/2}$ 
  - 所以只要滿足  $2^{(n-1)/4} (N^{\delta m(m+1)/2} X^{n(n-1)/2})^{1/n} < b^m n^{-1/2} = N^{\beta m} n^{-1/2}$
- 在這之前先來決定  $(m, t)$ , 越小的話  $L$  會小一點, 而且最好能消掉變數
  - 順便再引入另一個參數  $\varepsilon$  在實務可以操作
  - $m = \text{ceil}(\beta^2 / \delta \varepsilon)$ ,  $t = \text{floor}(\delta m(1/\beta - 1))$

# Coppersmith method

- 證明 (續)
  - 這邊來處理  $2^{(n-1)/4} (N^{\delta m(m+1)/2} X^{n(n-1)/2})^{1/n} < N^{\beta m} n^{-1/2}$  讓  $X$  的範圍定義出來
  - 因為  $n = m\delta + t = m\delta/\beta$ , 所以  $2^{(n-1)/4} N^{\beta(m+1)/2} X^{(n-1)/2} < N^{\beta m} n^{-1/2}$
  - 把不是  $X$  的丟過去,  $X^{(n-1)/2} < N^{\beta m} n^{-1/2} 2^{-(n-1)/4} N^{-\beta(m+1)/2} < 2^{-(n-1)/4} n^{-1/2} N^{\beta m/2}$
  - 取  $(n-1)/2$  方根,  $X < 2^{-1/2} n^{-1/(n-1)} N^{\beta m/(n-1)} < 2^{-1/2} n^{-1/n} N^{\beta m/n}$
  - 再一次  $1/n = \beta/m\delta$ ,  $X < 2^{-1/2} n^{-1/n} N^{\beta * \beta/\delta} < (1/2) * N^{\beta * \beta/\delta}$
  - 引入剛剛定義的  $\varepsilon$ ,  $X < (1/2) * N^{\beta * \beta/\delta - \varepsilon}$
- 實作
  - [sage.rings.polynomial.small\\_roots](#) ([source code](#))
  - [multivariate small\\_roots](#)

# Håstad's broadcast attack

- 情境假設
  - 已知  $e$  對公鑰  $(e, n)$  的  $e$  相同
  - 加密同一明文  $m$  並獲得對應的密文  $(c_1, c_2, c_3)$ 
    - $c_1 = m^3 \bmod n_1$
    - $c_2 = m^3 \bmod n_2$
    - $c_3 = m^3 \bmod n_3$
- 解法
  - 用 CRT 拼湊成  $c = m^3 \bmod (n_1 * n_2 * n_3)$
  - 因為滿足  $m < (n_1, n_2, n_3)$ , 所以  $m^3 < n_1 * n_2 * n_3$
  - 則可視為在  $\mathbb{Z}$  (整數域) 開  $e$  方根

# Håstad's broadcast attack with linear padding

- 情境假設
  - 已知  $e$  對公鑰  $(e, n)$  的  $e$  相同
  - 加密同一明文  $m$  並獲得對應的密文  $(c_1, c_2, c_3)$  和線性參數  $(f_1, f_2, f_3)$ 
    - $c_1 = f_1(m)^3 \bmod n_1$
    - $c_2 = f_2(m)^3 \bmod n_2$
    - $c_3 = f_3(m)^3 \bmod n_3$
- 解法
  - 用 CRT 拼湊一組參數  $t$ , 滿足  $t_i \bmod n_i = 1$  和  $t_i \bmod n_j = 0$
  - 讓  $g(x) = \sum(t_i * (f_i(x)^e - c_i))$ , 則  $m$  為  $g(x)$  的小根
- 參數
  - 因為滿足  $m < (n_1, n_2, n_3)$ , 所以  $m^3 < n_1 * n_2 * n_3$
  - 也就是說  $m < N^{1/3}$ , 大概率會滿足  $X < (1/2) * N^{1/3-\epsilon}$

# Franklin–Reiter related-message attack

- 情境假設
  - 已知明文  $(m_1, m_2)$  有線性關係  $f$ ,  $m_2 = f(m_1)$
  - 已知公鑰  $(e, n)$  和對應的密文  $(c_1, c_2)$
- 解法
  - 考慮兩多項式
    - $g_1 = x^e - c_1$
    - $g_2 = f(x)^e - c_2$
  - $m_1$  同時是  $(g_1, g_2)$  的根, 也就是說  $(x - m_1)$  可以同時整除  $(g_1, g_2)$ 
    - 則  $\gcd(g_1, g_2) = x - m_1$
  - 注意算出來結果是  $\mathbb{Z}_{\text{mod}} \text{ PolynomialRing}$ 
    - 用 monic 化成首一多項式後加上負號取常數項
      - 也就是  $-\gcd(g_1, g_2).\text{monic()}[0]$

# Coppersmith's short-pad attack

- 情境假設
  - 已知明文  $(m_1, m_2)$  為明文  $m$  的填充  $(r_1, r_2)$ 
    - $m_1 = 2^k m + r_1$
    - $m_2 = 2^k m + r_2$
  - 已知公鑰  $(e, n)$  和對應的密文  $(c_1, c_2)$
- 解法
  - 考慮兩多項式
    - $g_1 = x^e - c_1$
    - $g_2 = (x + y)^e - c_2$
  - 其中假設  $y = r_2 - r_1$ , 則  $\gcd(g_1, g_2) = x - m_1$
  - 如果  $(g_1, g_2)$  有非「常數公因式」若且唯若 [結式](#)  $\text{res}(g_1, g_2) = 0$ 
    - sagemath [resultant](#)
    - 相關資料: [西爾維斯特矩陣](#)

# Coppersmith's short-pad attack

- 解法 (續)
  - 令  $h(y) = \text{res}(g1, g2)$ , 即  $h = g1.\text{resultant}(g2, x)$
  - 若  $(r1, r2) < n^{1/e^{**2}}$  則  $r2 - r1$  為  $h(y)$  的小根
  - 得到  $(r1, r2)$  關係後帶回原式變成 Related Message 問題
- 參數
  - $\beta = 1, \delta = e^2 (h.\text{degree})$
  - 能找的範圍是  $X < (1/2) * n^{1/e^{**2}-\epsilon}$
  - 把  $\epsilon$  調小 (預設  $\beta/8$ ) 可以增加  $X$  的空間
- 實作
  - 在 Zmod PolynomialRing 底下沒有實作結式, 所以要換環
  - 換環處理完後把新環的代數帶入
  - 或是在換環的同時也把新代數帶入整合成新的單變數多項式



# Known high bits message

- 情境假設
  - 已知明文  $m$  的大部分高位  $m_0$ , 即  $m = m_0 + x_0$  未知  $x_0$
  - 已知公鑰  $(e, n)$  和由  $m$  加密的密文  $c = m^e \bmod n$
- 解法
  - 如果滿足  $x_0 \leq n^{1/e}$
  - 則  $x_0$  是  $f(x) = (m_0 + x)^e - c \pmod{n}$  的小根
- 參數
  - $\beta = 1, \delta = e$  (f.degree)
  - 能找的範圍是  $X < (1/2) * n^{1/e-\epsilon}$
  - 把  $\epsilon$  調小 (預設  $\beta/8$ ) 可以增加  $X$  的空間

# Factoring with high bits known

- 情境假設
  - 已知  $n = pq$ , 不知  $(p, q)$
  - 已知  $p$  的高位  $p_0$ , 即  $p = p_0 + x_0$  未知  $x_0$
- 解法
  - 如果滿足  $x_0 \leq n^{1/4}$
  - 則  $x_0$  是  $f(x) = p_0 + x \pmod{n}$  的小根
- 參數
  - $\beta = 1/2$ 、 $\delta = 1$  (f.degree)
  - 但因為  $p_0$  不一定大於  $n^{1/2}$ , 所以  $\beta$  可以視情況調小
  - $\beta$  變小會讓  $X$  也變小, 解決方式有
    - 若已知  $x_0$  範圍則直接設定  $X$  為上界
    - 把  $\varepsilon$  調小 (預設  $\beta/8$ ) 可以增加  $X$  的空間 ( $X < (1/2) * N^{\beta\beta-\varepsilon}$ )

# Boneh-Durfee's attack

- 基於 Coppersmith method 實作對於低解密指數的攻擊
  - 相較於維納攻擊  $d < (1/3) * N^{1/4}$ , B-D 的攻擊範圍為  $d < N^{0.292}$
- 多變數求小根的方法
- 推導
  - $k\phi(n) + 1 = ed$
  - $k(n - p - q + 1) + 1 = 0 \pmod{e}$
  - $2k((n + 1) / 2 + (-p - q) / 2) + 1 = 0 \pmod{e}$
  - 解  $f(x, y) = 1 + x * (A + y)$  的小根  $(x, y) = (2k, (-p - q) / 2)$
- 參數
  - $(m, t)$  代表  $(x, y)$  在 Coppersmith method shift 的次數 ( $1 \leq t \leq m$ )
  - $(X, Y)$  代表  $(x, y)$  的上限, 其中  $x < 2N^{\delta}$ 、 $y < N^{1/2}$
- [boneh\\_durfee.sage](#)

# CTFs

- [CryptoCTF](#)
- [CryptoHack](#)
- [id0-rsa](#)