

Crypto in CTF

中国信息通信研究院
邵子扬



01

密码学的世界

走进密码学

常见工具

进制的世界

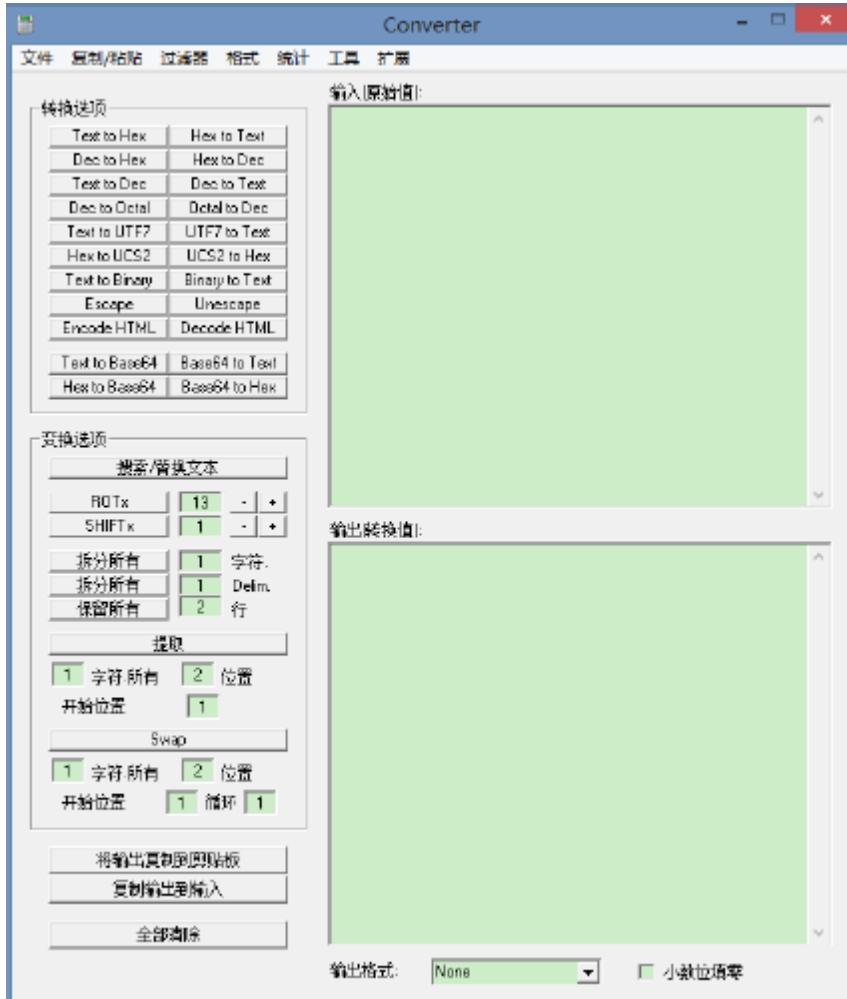
编码与解码

□ Crypto —— cryptography

- 简单讲，就是给你一串字符串，你要得到其中含义
- 内容：编码 / 古典加密 / 哈希 / 现代加密 / 自定义加密 / 特殊密码
- 难点
 - 需要扎实数学功底
 - 需要清晰的逻辑思维
 - 需要较为熟练的编程技巧



▶ 加解密工具



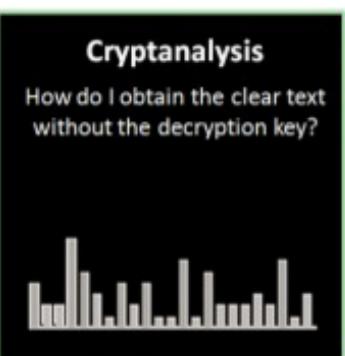
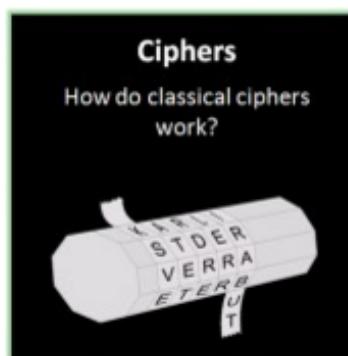
- Converter
- Text Decoder Toolkit
- JPocketKnife

► 在线加解密网站

<http://www.cryptool-online.org/>



What is CrypTool-Online?



Ciphers

How do classical ciphers work?

Cryptanalysis

How do I obtain the clear text without the decryption key?

Encrypt directly within your browser

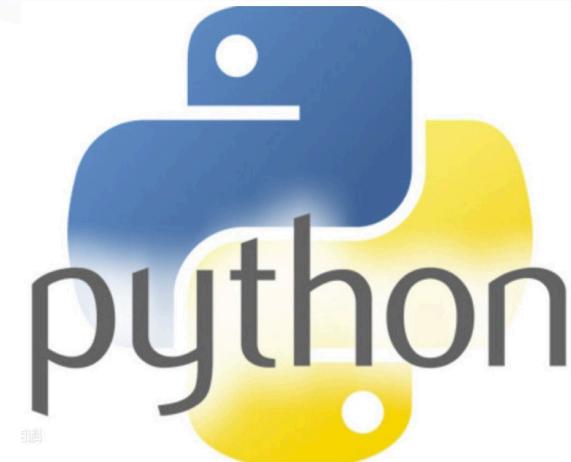
CrypTool-Online provides an exciting insight into the world of **cryptology**. A great variety of ciphers, encryption methods and analysis tools are introduced, often together with illustrated examples. Our emphasis is on making explanations easy to understand with the goal to further the general interest in cryptology and cryptanalysis. Therefore, this website also provides applets to experiment with the introduced methods and to learn the principles in an **interactive way**.

You can learn the fundamentals of historically relevant ciphers in a little while (e.g. the

► PYTHON大法好

脚本工具（基本用2.7.x）

优点：大量的第三方库、命令行式编程、简单的语法、跨平台



```
→ python-client python
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> "DqCkRbDm9sGeLhfoYYGbL75KJidVUZ4/Wtmmya8+8QK+Z86AnQx0qQ==".decode('base64')
"\x0e\x a0\x a4E\x b0\x e6\x f6\x c1\x 9e.\x 17\x e8U\x 81\x 9b/\x beJ&'UQ\x 9e?Z\x d9\x a6\x c9\x af>\x f1\x 02\x beg\x ce\x 80\x 9d\x 0cN\x a9"
>>> "sezR9jT/ht3tBgwI0cDijQ==".decode('base64')
'\xb1\xec\xd1\xf64\xff\x86\xdd\xed\x06\x0c"\xd1\xc0\xe2\x8d'
>>> "nqoe23wnz892as31".decode('base64')
"\x9e\xaa\x1e\xdb|\xcf\xcfvj\xcd\xf5"
```

► 信息单位

- 一个二进制数字叫做比特/位元 (bit)
 - 0/1
- 两个十六进制 (hex) 数字称作一个字节/位元组 (byte)
 - 0 ~ 255 / 0x00 ~ 0xFF
- 十进制 (dec) 就是我们平常的内容
 - 10, (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

可以看到基本上，数字代表这个进制有多少种数字

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	NUL	(null)	32	20 040	 	Space		64	40 100	@	Ø	96	60 140	`	`		
1	1 001	SOH	(start of heading)	33	21 041	!	!	!	65	41 101	A	A	97	61 141	a	a		
2	2 002	STX	(start of text)	34	22 042	"	"	"	66	42 102	B	B	98	62 142	b	b		
3	3 003	ETX	(end of text)	35	23 043	#	#	#	67	43 103	C	C	99	63 143	c	c		
4	4 004	EOT	(end of transmission)	36	24 044	$	\$	\$	68	44 104	D	D	100	64 144	d	d		
5	5 005	ENQ	(enquiry)	37	25 045	%	%	%	69	45 105	E	E	101	65 145	e	e		
6	6 006	ACK	(acknowledge)	38	26 046	&	&	&	70	46 106	F	F	102	66 146	f	f		
7	7 007	BEL	(bell)	39	27 047	'	'	'	71	47 107	G	G	103	67 147	g	g		
8	8 010	BS	(backspace)	40	28 050	(((72	48 110	H	H	104	68 150	h	h		
9	9 011	TAB	(horizontal tab)	41	29 051)))	73	49 111	I	I	105	69 151	i	i		
10	A 012	LF	(NL line feed, new line)	42	2A 052	*	*	*	74	4A 112	J	J	106	6A 152	j	j		
11	B 013	VT	(vertical tab)	43	2B 053	+	+	+	75	4B 113	K	K	107	6B 153	k	k		
12	C 014	FF	(NP form feed, new page)	44	2C 054	,	,	,	76	4C 114	L	L	108	6C 154	l	l		
13	D 015	CR	(carriage return)	45	2D 055	-	-	-	77	4D 115	M	M	109	6D 155	m	m		
14	E 016	SO	(shift out)	46	2E 056	.	.	.	78	4E 116	N	N	110	6E 156	n	n		
15	F 017	SI	(shift in)	47	2F 057	/	/	/	79	4F 117	O	O	111	6F 157	o	o		
16	10 020	DLE	(data link escape)	48	30 060	0	Ø	Ø	80	50 120	P	P	112	70 160	p	p		
17	11 021	DC1	(device control 1)	49	31 061	1	1	1	81	51 121	Q	Q	113	71 161	q	q		
18	12 022	DC2	(device control 2)	50	32 062	2	2	2	82	52 122	R	R	114	72 162	r	r		
19	13 023	DC3	(device control 3)	51	33 063	3	3	3	83	53 123	S	S	115	73 163	s	s		
20	14 024	DC4	(device control 4)	52	34 064	4	4	4	84	54 124	T	T	116	74 164	t	t		
21	15 025	NAK	(negative acknowledge)	53	35 065	5	5	5	85	55 125	U	U	117	75 165	u	u		
22	16 026	SYN	(synchronous idle)	54	36 066	6	6	6	86	56 126	V	V	118	76 166	v	v		
23	17 027	ETB	(end of trans. block)	55	37 067	7	7	7	87	57 127	W	W	119	77 167	w	w		
24	18 030	CAN	(cancel)	56	38 070	8	8	8	88	58 130	X	X	120	78 170	x	x		
25	19 031	EM	(end of medium)	57	39 071	9	9	9	89	59 131	Y	Y	121	79 171	y	y		
26	1A 032	SUB	(substitute)	58	3A 072	:	:	:	90	5A 132	Z	Z	122	7A 172	z	z		
27	1B 033	ESC	(escape)	59	3B 073	;	;	;	91	5B 133	[[123	7B 173	{	{		
28	1C 034	FS	(file separator)	60	3C 074	<	<	<	92	5C 134	\	\	124	7C 174	|			
29	1D 035	GS	(group separator)	61	3D 075	=	=	=	93	5D 135]]	125	7D 175	}	}		
30	1E 036	RS	(record separator)	62	3E 076	>	>	>	94	5E 136	^	^	126	7E 176	~	~		
31	1F 037	US	(unit separator)	63	3F 077	?	?	?	95	5F 137	_	_	127	7F 177		DEL		

Source: www.LookupTables.com

最广泛的符号编码

- 用7个比特组成
- 用十进制中0~127表示
- 对应十六进制0x00~0x7F
- 每个值对应一个字符

► 扩展字符集

128	Ҫ	144	É	161	í	177	ܹ	193	݉	209	ݔ	225	݊	241	݌
129	ü	145	æ	162	ó	178	ܻ	194	ݏ	210	ݔ	226	ݍ	242	݇
130	é	146	Æ	163	ú	179	ܺ	195	ݏ	211	ݔ	227	݊	243	݄
131	â	147	ö	164	ñ	180	ܺ	196	ݏ	212	ݔ	228	ݔ	244	ݏ
132	ä	148	ö	165	Ñ	181	ܺ	197	ݏ	213	ݔ	229	݊	245	ݏ
133	à	149	ò	166	ܺ	182	ܺ	198	ݏ	214	ݔ	230	݊	246	ݏ
134	å	150	û	167	ܺ	183	ܺ	199	ݏ	215	ݏ	231	ݏ	247	ݏ
135	ç	151	ù	168	ܺ	184	ܺ	200	ݏ	216	ݏ	232	ݏ	248	ܺ
136	è	152	—	169	ܺ	185	ܺ	201	ݏ	217	ݏ	233	ݏ	249	ܺ
137	ë	153	Ö	170	ܺ	186	ܺ	202	ݏ	218	ݏ	234	ݏ	250	ܺ
138	è	154	Ü	171	ܺ	187	ܺ	203	ݏ	219	ݏ	235	ݏ	251	ݏ
139	ï	156	£	172	ܺ	188	ܺ	204	ݏ	220	ݏ	236	ݏ	252	ܺ
140	î	157	¥	173	ܺ	189	ܺ	205	ݏ	221	ݏ	237	ݏ	253	ܺ
141	ì	158	—	174	ܺ	190	ܺ	206	ݏ	222	ݏ	238	ݏ	254	ݏ
142	Ã	159	ƒ	175	ܺ	191	ܺ	207	ݏ	223	ݏ	239	ݏ	255	ݏ
143	Â	160	á	176	ܹ	192	ܺ	208	ݏ	224	ݏ	240	ݏ		

扩展到了128-255

美国人用来表示制表符等

► GB2312-GBK



大陆使用两个字节的扩展字符数值

组成了GB2312编码了汉字；

GBK在其基础有加了2w个汉字。

台湾用的是BIG5

第一个字符是0x80之下的是ASCII

▶ ANSI & MBCS

- MBCS是多种编码的统称
- 两者是一致的，通常微软以ANSI表示，但是python用sys.getdefaultencoding()获取得到的将会是MBCS
- 在简体Win中，ANSI指代的会是默认的GBK

► Unicode & UTF

- 一个字符（无论什么字符）两个字节，共65535不同字符
- unicode和gbk并不兼容，编码不一致，需查表转换
- UTF (UCS Transfer Format)
 - unicode的传输，UTF8每次传8bits，UTF16每次传16bits
 - FFFF —— 标志 (\u) 后文本是高位在位
 - FFFE —— 标志 (\u) 后文本是低位在位
- UTF就是Unicode实现方式，UTF8是其中一种实现方式，使用1-4字节表示字符

► UTF-8编码

Unicode符号范围 (十六进制)		UTF-8编码方式 (二进制)
0000 0000 - 0000 007F		0xxxxxxx (ASCII)
0000 0080 - 0000 07FF		110xxxxx 10xxxxxx
0000 0800 - 0000 FFFF		1110xxxx 10xxxxxx 10xxxxxx
0001 0000 - 0010 FFFF		11110xxx 10xxxxxx 10xxxxxx 10xxxxxx



运算

□ 与运算

□ 或运算

□ 非运算（可逆）

□ 异或运算（可逆）

表2.2.1 与逻辑真值表

输入		输出
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

表2.2.2 或逻辑真值表

输入		输出
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

表2.2.3 非逻辑真值表

输入		输出
A	Y	
0	1	
1	0	

表2.2.6 异或逻辑真值表

输入		输出
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

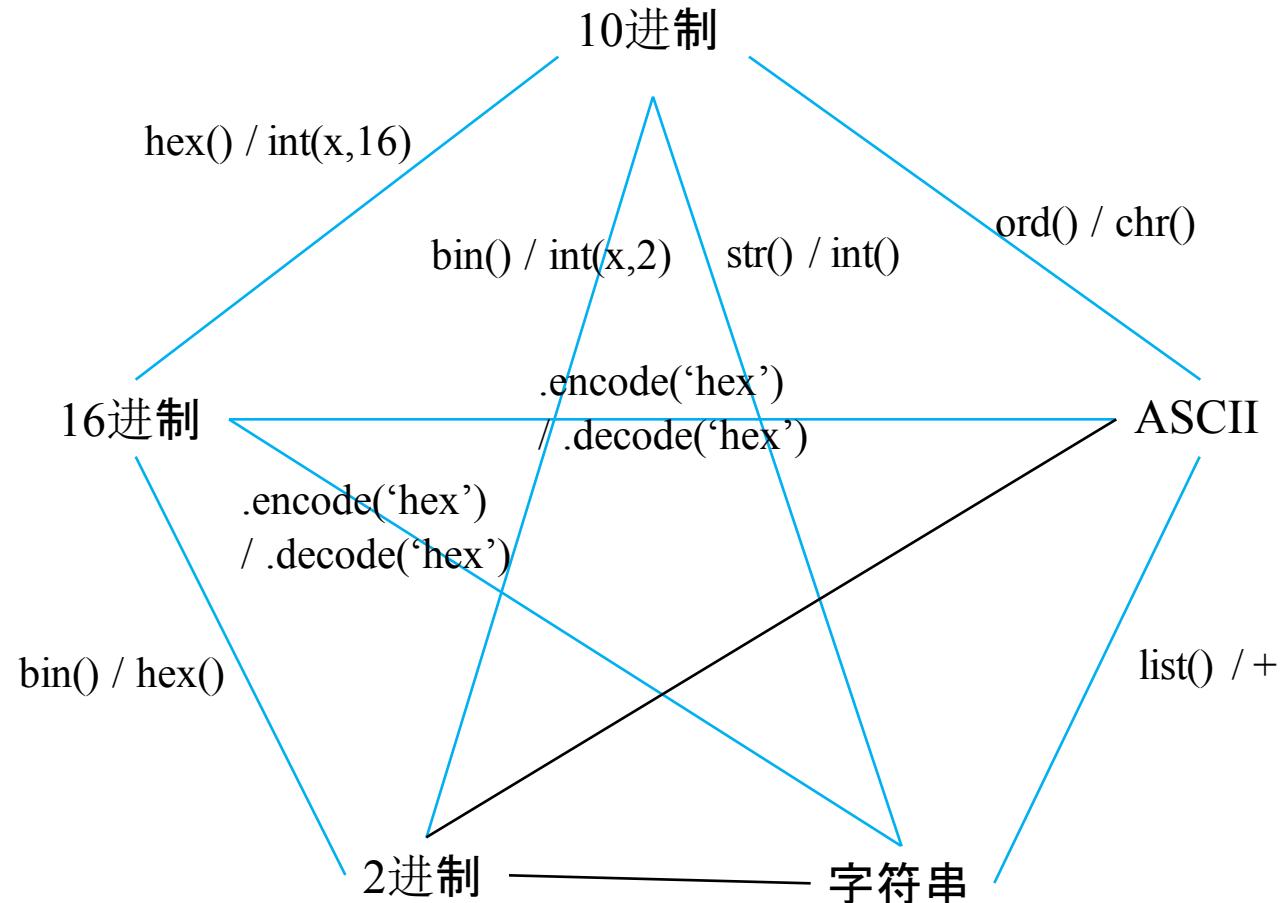
▶ 移位运算

- 逻辑移位：无符号数，空位均补0
- 算术移位：有符号数，符号位保持不变
 - 左移或右移n位相当于乘以或除以 2^n （乘法/除法）
- 算术移位对空出的位应该补0还是1？
 - 算术运算是对真值操作。

▶ python中的运算

运算	例子	运算	例子
四则	>>> 1+2*3-10/3%2 4	^	>>> 0xca ^ 0xfe 52
幂	>>> 5**1024 556268464626...12890625L	&	>>> 13 & 60 12
位移	>>> 1<<16 65535	~	>>> ~128 -129
//	>>> 21 // 10 2	 	>>> 1 2 3
in	>>> "c" in "abc" True	and	>>> 13 and 60 60
not in	>>> "c" not in "abC" True	or	>>> 13 or 60 13
is	>>> "20" is "20" True	not	>>> not 0 True
is not	>>> "ab" is not "AB" True	合并	>>> 'A'*4+"0x860X640xca" 'AAAA0x86d0xca'

▶ Python与进制转换



▶ 常见编码

- ASCII编码
- Base64/32/16编码
- shellcode编码
- Quoted-printable编码
- XXencode编码
- UUencode编码
- URL编码
- Unicode编码
- Escape/Unescape编码
- HTML实体编码

编码转换工具

常用编码转换

- » Quoted-printable
- » Base64编码
- » UUencode编码
- » XXencode编码
- » URL编码
- » Escape编码/解码
- » js转unicode
- » native转ascii
- » css转unicode
- » html转unicode
- » html转ubb
- » htmlencode

常用编码表

- » ASCII码表
- » gb2312编码表
- » gbk编码表
- » 区位码(分区)
- » 区位码(混排)

开发常用转换工具

- » html转js转html
- » html转asp转html
- » html转php转html
- » html转jsp转html

在线Escape编码/加密、Unescape解码/解密、

Escape/Unescape加密解码/编码解码,又叫%u编码,从以往经验看编码字符串来问题非常简单了。Escape编码/加密,就是字符对应UTF-16 16进制表示方式前符“中”, UTF-16BE是:“6d93”,因此Escape是“%u6d93”,反之也一样

Escape/unEscape文本: 复杂程度: 基本类型(常用字符不编码)

然而部分由于 Unicode 版本发展原因,很多浏览器

↑ 将你电脑文件直接拖入试试^-^

转换结果:

► BASE系列编码

```
>>> import base64
>>> s="stay hungry,stay foolish"
>>> e1=base64.b64encode(s)
>>> print e1
c3RheSBodW5ncnksc3RheSBmb29saXNo
>>> e2=base64.b32encode(s)                                编码
>>> print e2
ON2GC6JANB2W4Z3SPEWHG5DBPEQGM33PNRUXG2A=
>>> e3=base64.b16encode(s)
>>> print e3
737461792068756E6772792C7374617920666F6F6C697368
```

```
>>> d1=base64.b64decode(e1)
>>> print d1
stay hungry,stay foolish
>>> d2=base64.b32decode(e2)
>>> print d2
stay hungry,stay foolish
>>> d3=base64.b16decode(e3)
>>> print d3
stay hungry,stay foolish
```

解码

▶ Escape/Unescape

- Escape/Unescape加密解码/编码解码,又叫%u编码，采用UTF-16BE模式，Escape编码/加密,就是字符对应UTF-16 16进制表示方式前面加%u。
- Unescape解码/解密，就是去掉” %u” 后，将16进制字符还原后，由utf-16转码到自己目标字符。
 - 如：字符 “中” ，UTF-16BE是：“6d93” ，因此Escape是 “%u6d93” 。

源文本: The

编码后: %u0054%u0068%u0065

► URLencode

网页进行传输数据用

% + HEX_ASCII

test123

%74%65%73%74%31%32%33



Quoted-printable 编码

Quoted-Printable

```
=E5=9C=A8=E6=89=80=E6=9C=89=E9=82=AE=E4=BB=B6=E5=A4=84=E7=90=86=E7=9A=84=E5=90=84=E5=BC=8F=E5=90=84=E6=A0
=B7=E7=9A=84=E7=BC=96=E7=A0=81=E4=B8=AD=EF=BC=8C=E5=BE=88=E5=A4=9A=E7=BC=96=E7=A0=81=E7=9A=84=E7=9B=AE=E7
=9A=84=E9=83=BD=E6=98=AF=E9=80=9A=E8=BF=87=E7=BC=96=E7=A0=81=E6=89=8B=E6=AE=B5=E4=BD=BF=E5=BE=97=E4=B8=83
=E4=BD=8D=E5=AD=97=E7=AC=A6=E7=9A=84=E9=82=AE=E4=BB=B6=E5=8D=8F=E8=AE=AE=E4=BD=93=E7=B3=BB=E5=8F=AF=E4=BB
=A5=E4=BC=A0=E9=80=81=E5=85=AB=E4=BD=8D=E7=9A=84=E4=BA=8C=E8=BF=9B=E5=88=B6=E6=96=87=E4=BB=B6=E3=80=81=E5
=8F=8C=E5=AD=97=E8=8A=82=E8=AF=AD=E8=A8=80=E6=96=87=E5=AD=97=E7=AD=89=E7=AD=89=E3=80=82=20Quoted-
Printable=E4=B9=9F=E6=98=AF=E8=BF=99=E6=A0=B7=E4=B8=80=E4=BA=9B=E7=BC=96=E7=A0=81=E4=B8=AD=E7=9A=84=E4=B8
=80=E4=B8=AA=EF=BC=8C=20=E5=AE=83=E7=9A=84=E7=9B=AE=E7=9A=84=E5=90=8C=E6=A0=B7=E6=98=AF=E5=B8=AE=E5=8A=A9
=E9=9D=9EASCII=20=E7=BC=96=E7=A0=81=E7=9A=84=E4=BF=A1=E4=BB=B6=E4=BC=A0=E8=BE=93=E9=80=9A=E8=BF=87=20SMTP
=E3=80=82Quoted-
Printable=20=E7=BC=96=E7=A0=81=E6=98=AF=E5=AD=97=E7=AC=A6=E5=AF=B9=E5=BA=94=E7=9A=84=E7=BC=96=E7=A0=81=EF
=BC=8C=E6=AF=8F=E4=B8=AA=E6=9C=AA=E7=BC=96=E7=A0=81=E7=9A=84=E4=BA=8C=E8=BF=9B=E5=88=B6=E5=AD=97=E7=AC=A6
=EF=BC=96=E7=9A=84=E7=BC=96=E7=A0=81=E7=9A=84=E4=BB=B6=E4=BC=A0=E8=BE=93=E9=80=9A=E8=BF=87=20UTF-8
 UTF-8  简体中文(GB2312)  繁体中文(BIG5)  日语(EUC-JP)  朝鲜语(EUC-KR) 请选择
```

Quoted-Printable 编码 Quoted-Printable 解码 拷贝 剪切 粘贴 清除

UTF-8

在所有邮件处理的各式各样的编码中，很多编码的目的都是通过编码手段使得七位字符的邮件协议体系可以传送八位的二进制文件、双字节语言文字等等。Quoted-Printable也是这样一些编码中的一个，它的目的同样是帮助非ASCII编码的信件传输通过SMTP。Quoted-Printable编码是字符对应的编码，每个未编码的二进制字符被编码成三个字符，即一个等号和一个十六进制的数字，如“A8”。

- 可打印字符引用编码，多用途互联网邮件扩展（MIME）一种实现方式，一般在邮件头可见。
- 任何一个8位的字节可被编码为3个字符：一个等号后跟随两个十六进制（0-9或A-F）表示该字节的数值

▶ 一些编码特征总结

名称	例子	特征总结
Quoted-printable	=E5=9C	等号+hex
Base64	MTIxMg==	大小写一串，可能有=+
Base32	GEZDCMQ=	大写一串，可能有=
Base16	616263313233	很大一串数字，有几个字母
HTML实体	<script> <	lt和gt等常见，参考实体表
URL编码	%3D%231a	百分号，无百分号长度%2=0，只对符号中文用
UNICODE/Escape	\u52a0\u5bc6 / %u00a0%u0068	U是特点，而且4位
UTF7	+/v8APQAJ-1a	+/开头，大小写字母+数字
XXencode	4NjS0-eRKpkQm-jR	0-63之间的ASCII
UU编码	M5&AE(%U:6-K(&)R;W=N(&9O>	(%&*这类特殊字符，32-35之间的ASCII

02

代码混淆与加密

▶ 常见代码混淆与加密

- asp混淆加密
- php混淆加密
- css/js混淆加密
- VBScript.Encode混淆加密
- ppencode
- rrencode
- jjencode/aaencode
- JSfuck
- jother
- brainfuck编程语言

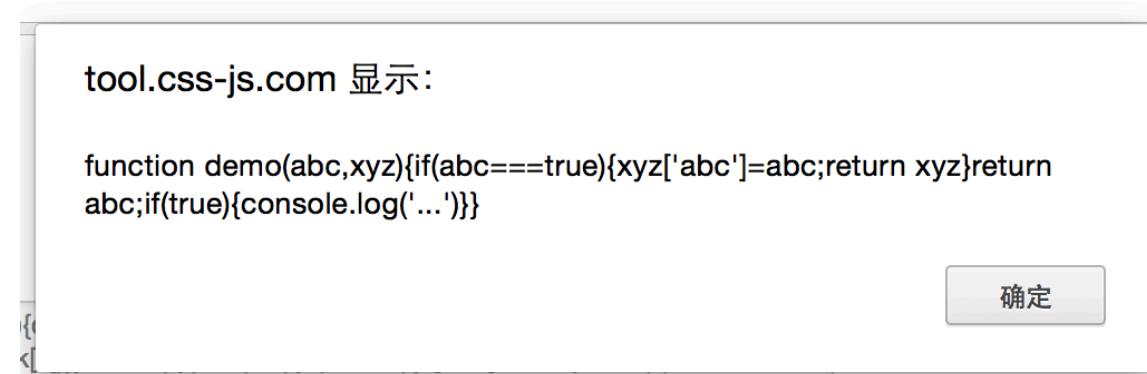
▶ JS/CSS加密

变成前面有个eval，逻辑比较复杂。

其实只要将eval改成alert再执行以下就可以得到源码。

CSS, JavaScript 压缩, 美化, 加密, 解密

```
eval(function(p,a,c,k,e,d){e=function(c){return c;if(!''.replace(/\/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d[e]]};e=function(){return '\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\b'+e(c)+'\b','g'),k[c])}}return p}('6 5(0,1){3(0==4){1['0']=0;2 1}2 0;3(4,{8.7(...)},9,9,'abc|xyz|return|if|true|demo|function|log|console'.split('|'),0,{}})
```



► PHP加密

有时候可以直接进行代码审计，但是有时候就要运行调试了。

```
$hh = "p"."r"."e"."g"."_". "r"."e"."p"."l"."a"."c"."e"; //字符串preg_replace  
$hh("/jc/e",$_POST['h'],"111jc222"); //hh变量为preg_replace, 然后就可以当函数用
```

```
<?php  
  
session_start();  
  
// 如果$_POST['code']为真，执行$_SESSION['theCode'] = trim($_POST['code'])操作  
$_POST['code'] && $_SESSION['theCode'] = trim($_POST['code']);  
  
echo "session:" . $_SESSION['theCode'];  
var_dump($_SESSION);  
  
$_SESSION['theCode']&&preg_replace('\'a\'eis','e'. 'v'. 'a'. 'l'. '('.  
    base64_decode($_SESSION[\'theCode\']))','a');  
  
?>
```

► VBScript.Encode / MSScriptEncode

VBscript.Encode 解码器

VBscript.Encode 解码器

```
#@~^IQAAAA==\ko$K6,JtV^GPSW.V9"ES,vcB~Jbx0KElwoAAA==^#~@
```

```
<SCRIPT language=JScript.Encode>
  #@~^QwIAAA==@#@&0; mDkW P7nDb0zZKD.n1YAMGhk+Dvb`@#@&P,kW`UC7kLIDGDcl22gl:n~
{'P3~dYMc*iNz&R @*^#~@
</SCRIPT>
```

MSScriptEncode

► jjencode/rrencode/pencode

□ JJencode

□ Javascript -> 颜文字

□ RRencode

□ Ruby -> 特殊符号

□ PPencode

□ Perl -> 英文字符 (无特殊符号)

aaencode demo

aaencode - Encode any JavaScript program to Japanese style emoticons (^_~)

Enter JavaScript source:

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

aaencode

[eval] [Permalink]

Evil0x [utf-8.js]

The screenshot shows the 'aaencode demo' interface. It displays the original JavaScript code 'alert("Hello, JavaScript")' and its converted output, which is a highly obfuscated string of Japanese-style emoticons (e.g., '^', 'o', '(', ')') separated by various operators like '+', '=', and '=='. The output is so long that it wraps across multiple lines. Below the converted code are two buttons: '[eval]' and '[Permalink]'.

JJENCODE

- 匿名函数的原生形式
- 由[、]、(、)、{、 }、+、!组成
- 可在浏览器的console直接还原



使用6个字符[、]、(、)、!、+来编写JavaScript程序

JSFuck is an esoteric and educational programming style based on the atomic parts of JavaScript. It uses only six different characters to write and execute code.

It does not depend on a browser, so you can even run it on Node.js.

Use the form below to convert your own script. Uncheck "eval source" to get back a plain string.

警报框

Encode Eval Source

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


03

古典加密

猪圈密码

凯撒密码

维吉尼亚密码

栅栏密码

希尔密码

► 古典加密

□ 替代密码 Substitution Ciphers

- 猪圈密码
- 凯撒密码
- 维吉尼亚密码

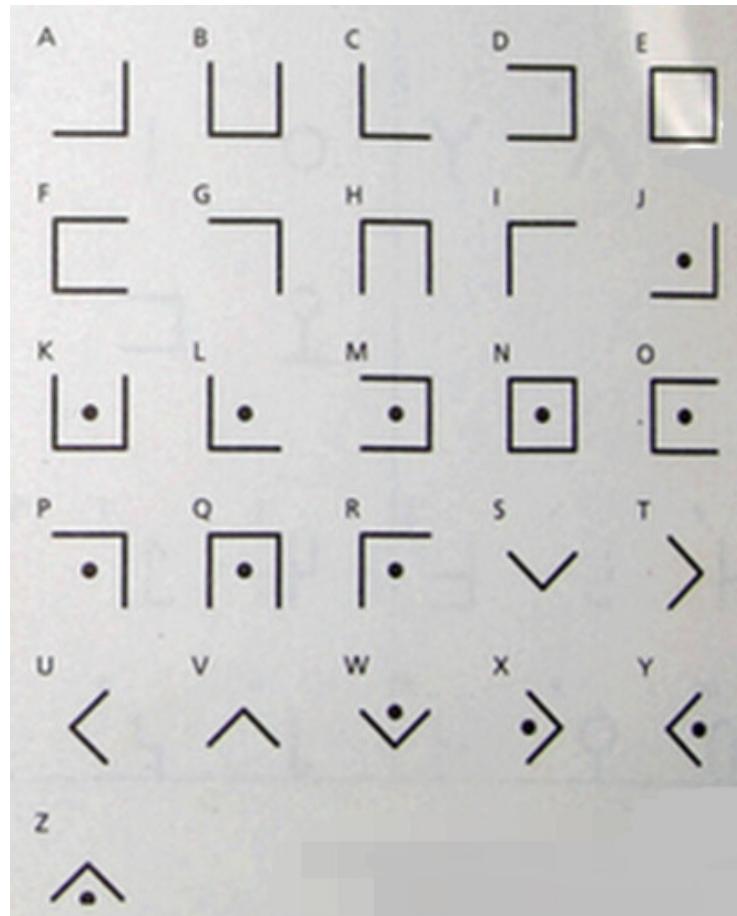
□ 置换密码 Transposition Ciphers

- 栅栏密码

□ Hill密码 Hill Cipher



► 猪圈密码



明文: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

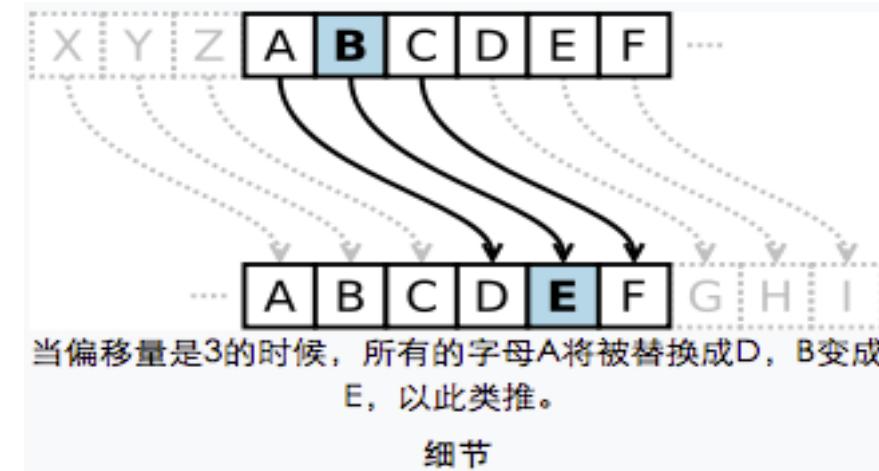
密文:

>□□ □<□□□ □□□□□□ □□> □<□□□ □□□
□ >□□ □□□< □□□

凯撒加密

$$E_n(x) = (x + n) \mod 26$$

明文字母表: ABCDEFGHIJKLMNOPQRSTUVWXYZ
密文字母表: DEFGHIJKLMNOPQRSTUVWXYZABC



□ Skill

- 在一些CTF中，借助字符统计可以收获奇效。统计出出现最多的字符，其对应的明文字符是e。

□ ROT13 —— 这是一种特殊的凯撒密码，K值为13。



凯撒加密爆破

U8Y]:8KdJHTXRI>XU#?!K_ecJH
]kJG*bRH7YJH7YSH]*=93dVZ3^
S8*\$:8"&:9U]RH;g=8Y!U92'=j*\$K
H]ZSj&[S#!gU#*dK9\.



wctf{kaisa_jiaaaaami}

lstr=""""U8Y]:8KdJHTXRI>XU#?!K_ecJH]kJG*bRH7YJH7Y
SH]*=93dVZ3^S8*\$:8"&:9U]RH;g=8Y!U92'=j*\$K
H]ZSj&[S#!gU#*dK9\.""""

```
for p in range(127):
    str1 = ""
    for i in lstr:
        temp = chr((ord(i)+p)%127)
        if 32<ord(temp)<127 : #2
            str1 = str1 + temp
            feel = 1
        else:
            feel = 0
            break
    if feel == 1:
        print str1
```

► 维吉尼亚密码

□ 一种多表替换的密码

例如秘钥是：RELATIONS

TO BE OR NOT TO BE THAT IS THE QUESTION
RE LA TI ONS RE LA TION SR ELA TIONSREL



KS ME HZ BBL KS ME MPOG AJ XSE JCSFLZSY

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

► 维吉尼亞密碼

k: ????????????

p: SECCON{????????????????????????????????????}

c: LMIG}RPEDOEWKJIQIWKJWMNDTSR}TFVUFWYOCBAJBQ

k=key, p=plain, c=cipher, md5(p)=f528a6ab914c1ecf856a1d93103948fe

1. 根据已知内容进行k的长度猜解（动态规划思想）
2. 根据对应的字符表得到key
3. 用来进行解密

► 栅栏密码

- 把加密明文分为N组，再把每组的第一个字符组合，每组的第N个字符组合
- 在自定义加密中出题可能性比较大

明文: The quick brown fox jumps over the lazy dog

去空格: Thequickbrownfoxjumpsoverthelazydog

分组: Th eq ui ck br ow nf ox ju mp so ve rt he la zy do g

第一组: Teucbonojmsvrhlzdg

第二组: hqikrwdxupoeteayo

密文: Teucbonojmsvrhlzdghqikrwdxupoeteayo

► 栅栏曲线

明文: The quick brown fox jumps over the lazy dog

填入5行7列表(事先约定填充的行列数)

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

加密的回路线(事先约定填充的行列数)

The diagram shows a 5x7 grid of letters. Red arrows indicate a specific path starting from the top-left corner (T) and moving through the grid in a zig-zag pattern. The path starts at T, goes down to k, right to b, up to o, left to x, down to j, right to u, up to o, left to v, right to e, up to r, left to t, right to h, up to d, left to o, and finally down to g.

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

密文: gesfc inpho dtmwu qoury zejre hbxxva lookT

► 栅栏列位移

m e m a t r h t g p r y
e t e f e t e o a a t

2列的栅栏

加密后的信息是MEMATRHTGPRYETEFETEOAAT

这种技巧是对密码分析者来说实在微不足道。一个更复杂的方案是把消息一行一行地写成矩形块，然后按列读出，但是把列的次序打乱。列的次序就是算法的密钥。例如：

密钥	4	3	1	2	5	6	7
明文	a	t	t	a	c	k	
明文	p	o	s	t	p	o	n
明文	e		u	n	t	i	l
明文		t	w	o		x	y

带秘钥的栅栏

密文：tsuwartnoto tape cpt koix nly

Hill 密码

明文： ACT

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

明文对应矩阵：

$$\begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix}$$

加密密钥： GYBNQKURP

加密矩阵：

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$$

计算过程：

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

密文： FIN

矩阵乘积和字母表位置的结合

矩阵乘积：

$$\mathbf{AB} = \begin{bmatrix} a_{1,1}[b_{1,1} & b_{1,2} & \dots] + a_{1,2}[b_{2,1} & b_{2,2} & \dots] + \dots \\ a_{2,1}[b_{1,1} & b_{1,2} & \dots] + a_{2,2}[b_{2,1} & b_{2,2} & \dots] + \dots \\ \vdots \end{bmatrix}$$

04

单向散列

哈希摘要

碰撞攻击

长度扩展攻击

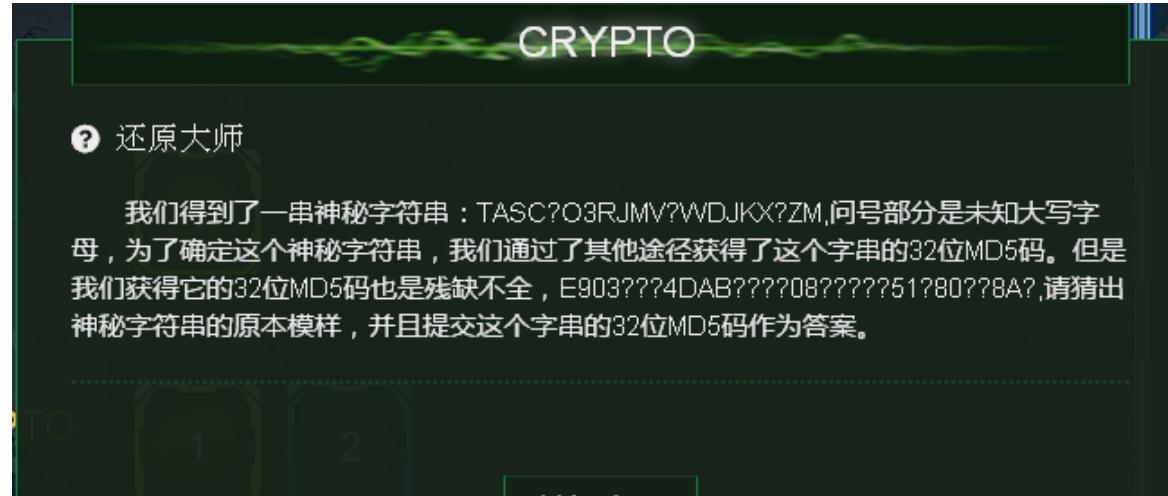
▶ 哈希算法

- 哈希算法（摘要算法、单向散列加密）
- 压缩性：任意长度的数据，算出哈希值长度是固定的
- 容易计算：从原数据算出哈希值很容易，不需要秘钥
- 抗修改性：对原数据修改1个字节，得到的哈希值都有很大区别
- 强抗碰撞：已知原数据和其哈希，要找到一个具有相同哈希值的数据非常困难

加密算法	MD5-16	MD5-32	SHA-1	SHA-2	SHA-3
输出长度	64bits	128bits	160bits	256bits	512bits

想要从密文中还原单向散列加密前的明文几乎是不可能的

▶ MD5碰撞



```
1 import hashlib
2 for i in range(65,91):
3     for j in range(65,91):
4         for k in range(65,91):
5             string = "TASC" + chr(i) + "03RJMV" + chr(j) + "WDJKX" + chr(k) + "ZM"
6             enc_str = hashlib.md5()
7             enc_str.update(string)
8             enc_str = enc_str.hexdigest()
9             if enc_str[0:4].upper() == "E903":
10                 print "The string is :" + string
11                 print "The md5 of the string is :" + enc_str.upper()
```

► 长度扩展攻击

- MD5以512bits作为一个区块进行计算
- 当不满足512bits时候
 1. 首先补一个 1 (二进制位上的一个 1，不是十进制的 1)
 2. 接着在后面补 0 (同样是二进制位上的 0)，直到满足比特长度对 512 求余为 448 这个条件。
 3. 接着补 64bit 的长度，这个长度是在补 1 和 0 以前的长度（元长度），如果长度超出了 64bit，那么就取低 64bit。

Raw_data + '\x80' + '\x00' * n + '\x00\x00\x00\x00\x00\x00\x00\x00'

* 0x80 = 0b10000000

► 长度扩展攻击

- 一般在进行MD5计算都会加上salt : md5(salt + xxxxxx)
- 如何在不知道salt的时候进行攻击
- 假设
 - 有 md5(salt + xxxxxxxx) , 且salt长度为10
 - 我需要将运算值补充到64byte

```
md5( salt + xxxxxxxx + \x80 + \x00 * (64-10-7-1-8) + \x88\x00\x00\x00\x00\x00\x00\x00)
```

- 然后如果我在后面继续补充，则进入新的计算块

► 长度扩展攻击

- 接下来，MD5区块会设置成前面的计算得到哈希

word A:	01	23	45	67
word B:	89	ab	cd	ef
word C:	fe	dc	ba	98
word D:	76	54	32	10

- 然后利用这个区块计算新一块数据的哈希
 - 所以只要知道了前面的哈希值，我就可以在添加恶意数据后得到正确的哈希，不用知道salt的信息
 - 自动化——hashpump

➤ 长度扩展攻击

第四届工信部第五题，可以从login的代码中得到存在漏洞，而且知道登录失败的哈希：ed8f21668614d9dabc5339addfc57457，key的长度是

```
1 def login():
2     Leak()
3     data = 'username=' + get_input_string('Username:')
4     data2 = get_input_string('Signature:')
5
6     if data2 != md5(KEY + data):
7         print "test:fail"
8
9     if data2 == md5(KEY + data):
10        print "test:ok"
```

05

现代加密

块加密模式

AES

RSA

► 分组密码

- 以AES-128为例 (128bit = 16byte)
- 则对于abcdefghijklmnopqrstuvwxyz会分为两块
 - B1 = abcdefghijklmnop (16byte)
 - B2 =qrstuvwxyz (10byte)
- Key也是16byte , 和B1长度正好相等
- 但是B2就小一点 , 那么就是B2需要进行填充一些内容到16byte以进行加密

▶ 数据填充

Padding——PKCS#7 & PKCS#5

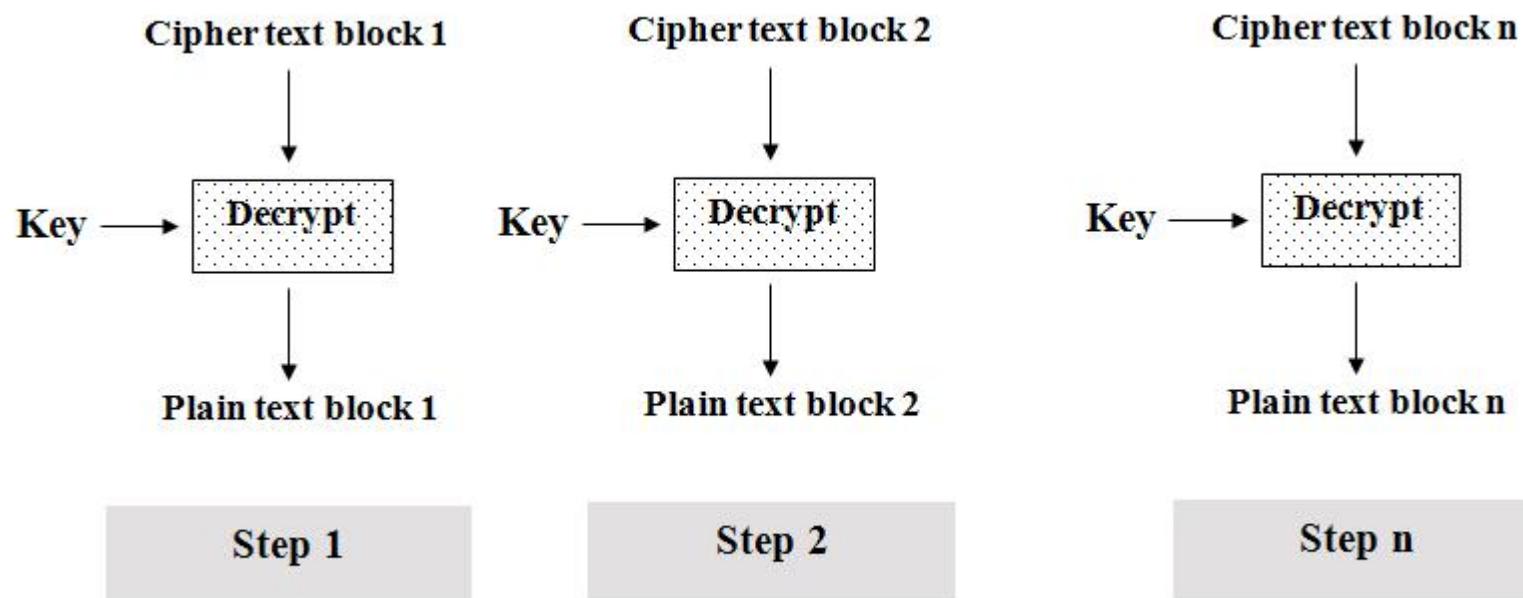
- 填充的值就是要填充的个数信息
- PKCS#7支持blocksize是1 to 255 bytes
- PKCS#5支持blocksize是8

对于AES-128-CBC来说，块大小是16字节

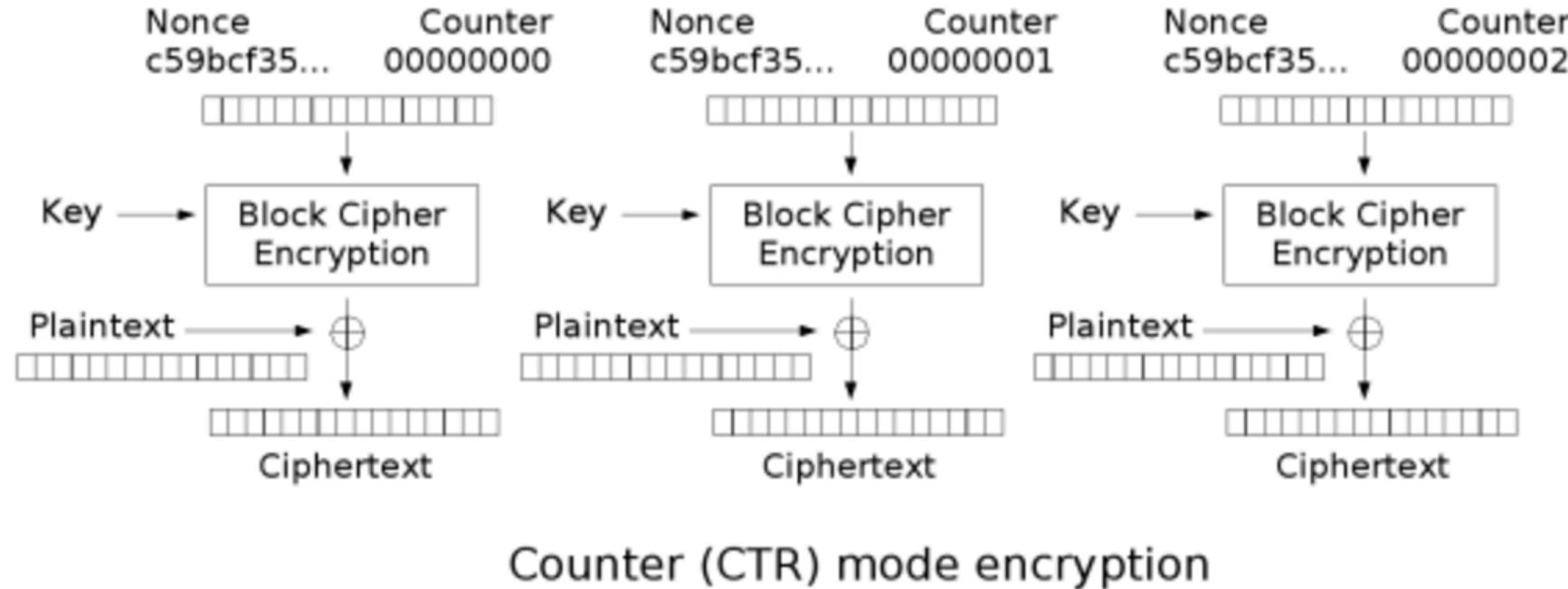
Plain	111111111111	11bytes
Padding	111111111111 Chr(0x05) Chr(0x05) Chr(0x05) Chr(0x05) Chr(0x05)	16bytes
Hex-Plain	FF	10byte
Hex-Padding	FF FF FF FF FF FF FF FF FF FF 06 06 06 06 06 06	16byte

► ECB的分组算法

分组密码，也叫块加密(block cyphers)，一次加密明文中的一个块。是将明文按一定的位长分组，明文组经过加密运算得到密文组，密文组经过解密运算（加密运算的逆运算），还原成明文组。模式分为ECB，CBC，CFB，OFB四种。



► CTR mode



Ramdom IV && Random Key

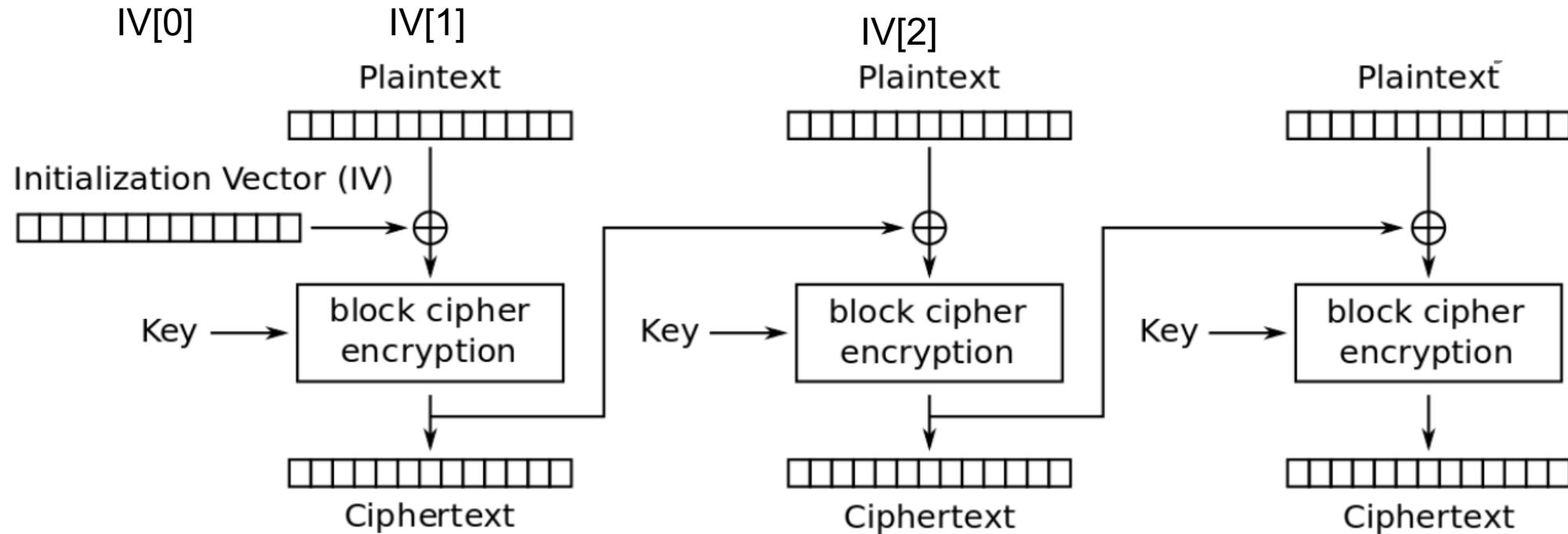
▶ 解题方法

chosen-plaintext attack

- 填充字符
- 头部字符（或者目标文件格式）

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Plaintext	%	P	D	F	-	1	.	?	\n	?	?	?	?	?	?	?
Plaintext (hex)	0x25	0x50	0x44	0x46	0x2d	0x31	0x2e	?	0x0a	?	?	?	?	?	?	?
Ciphertext	0xd6	0xea	0xb9	0x75	0x30	0x19	0x6a	0x9b	0x05	0x2a	0xef	0x97	0xee	0x2c	0x00	0xdb
Y	0xf3	0xba	0xfd	0x33	0x1d	0x28	0x44	?	0x25	?	?	?	?	?	?	?

► CBC分组加密



$$P[n] = C[n-1] \wedge D(C[n], K)$$

通过IV篡改AES

已知：

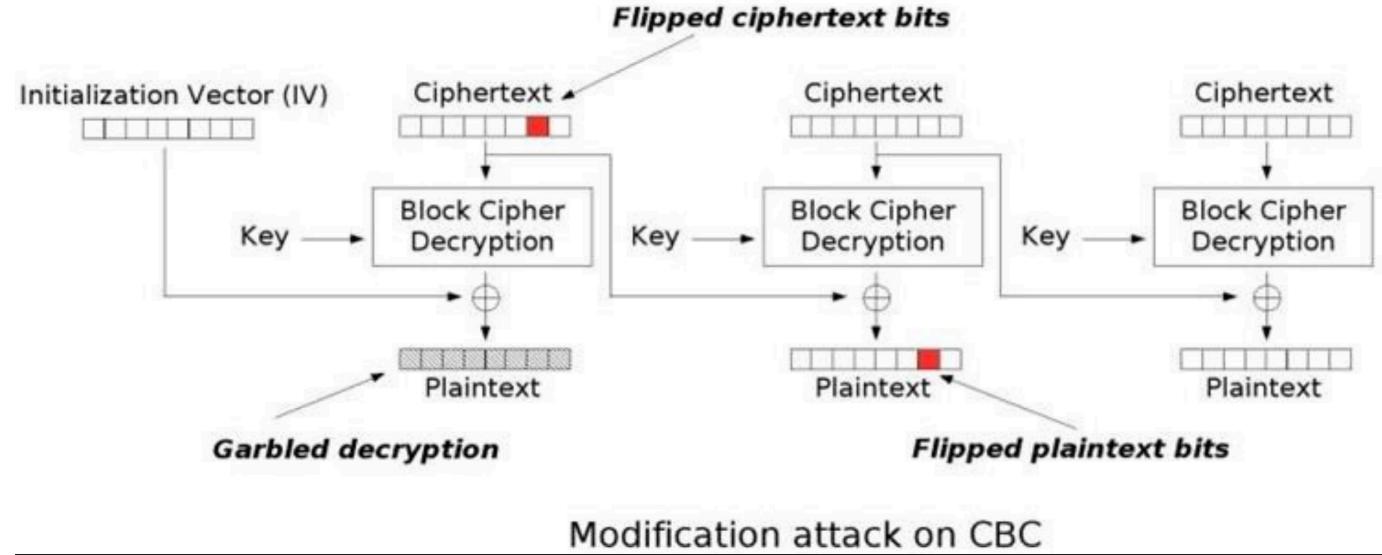
- AES-128-CBC
- OLD-IV
- OLD-Plaintext
- NEW-Plaintext

未知：

- Key

求：

- NEW-IV



Modification attack on CBC

$D(B1, \text{key}) \text{ xor } \text{OLD-IV} = \text{"Pass: sup3r31337"}$

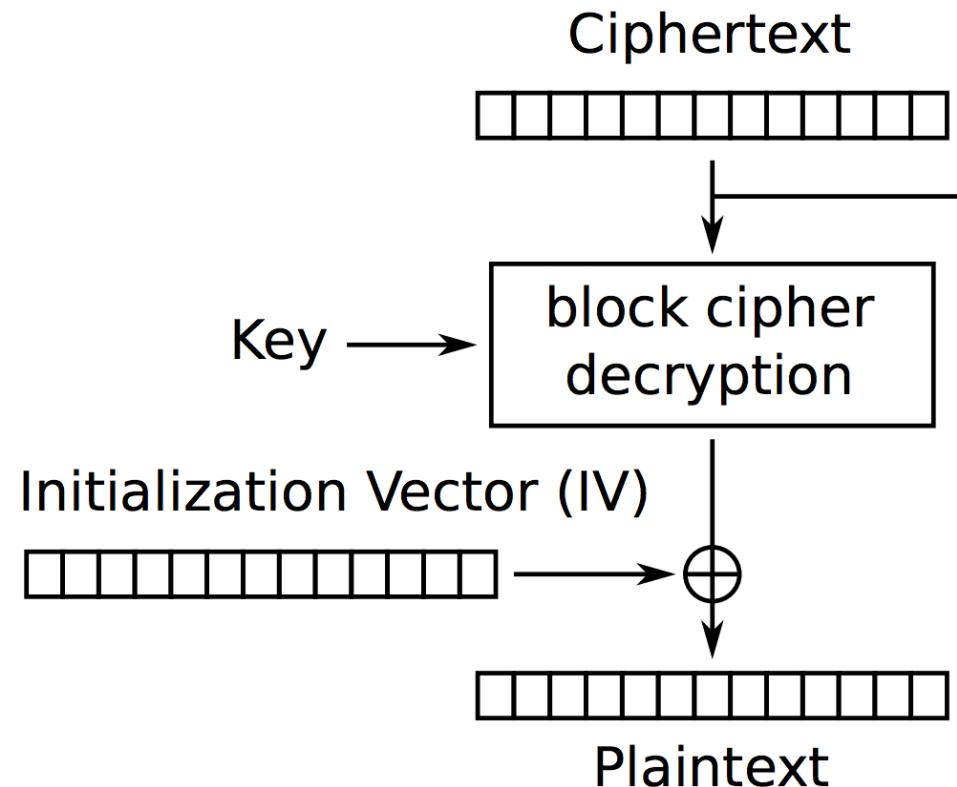
$D(B1, \text{key}) \text{ xor } \text{NEW-IV} = \text{"Pass: notAs3cre7"}$

$\text{"Pass: sup3r31337"} \text{ xor } \text{OLD-IV} \text{ xor } \text{"Pass: notAs3cre7"} = \text{NEW-IV}$

▶ Padding Oracle

□ 影响密文的几个因素

- Key
- IV
- Cipher text



► Padding Oracle 0x1

选中目标block是 0x39 0x73 0x23 0x22 0x07 0x6a 0x26 0x67 共8字节 , PKCS#5

构造 0x00 0x00 0x00 0x00 0x00 0x00 0x00 **0x00** | 0x39 0x73 0x23 0x22 0x07 0x6a 0x26 0x67

从0x01~0xFF替换上面黑体的地方 , 与0x67进行异或 , 要求 : $0x?? \wedge 0x67 == 0x01$

根据之前得到的公式 $NEW-P \wedge NEW-IV \wedge OLD-IV = OLD-P \rightarrow 0x1 \wedge 0x66 \wedge 0x11 = hex(P[8])$

Block 1 of 1								
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
TRIPLE DES								
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x67
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x66						
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01

VALID PADDING



► Padding Oracle 0x2

根据之前的 $\text{NEW-P} \wedge \text{NEW-IV} \wedge \text{OLD-IV} = \text{OLD-P}$

现在 NEW-P 换 $0x02$,

又已知 OLD-P 和 OLD-IV ,

求此时的 NEW-IV 值

求得P值之后又返回到 $0x1$ 类似的步骤了

AD	BC	4F	D3	2D	A3	84	E3
00	00	4E	D2	2D	A3	84	E3

► Padding Oracle总结

- 爆破量从幂乘到简单的乘积
- 必要条件
 - 知道原IV
 - PKCS#5 / PKCS#7
 - CBC模式
 - 可以识别到相关的错误和正确

► RSA原理

$n = p * q$ //两个随机质数

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

选择 **1 < e < φ(n)**，且e与 $\varphi(n)$ 互质

$ed \equiv 1 \pmod{\varphi(n)}$ -> $ed - 1 = k\varphi(n)$ //e是1~ $\varphi(n)$ 之间的随机质数k
//d为变量，得到d

欧几里得计算得到d和k，取的正整数

公钥：(n, e) 私钥：(n, d)

- (1) $ed \equiv 1 \pmod{\varphi(n)}$ 。只有知道e和 $\varphi(n)$ ，才能算出d。
- (2) $\varphi(n) = (p-1)(q-1)$ 。只有知道p和q，才能算出 $\varphi(n)$ 。
- (3) $n = pq$ 。只有将n因数分解，才能算出p和q。

RSA的加解密，就是找到d的过程！

▶ 公钥加密与私钥解密

公钥：(n , e) 私钥：(n , d)

- 用公钥对m进行加密得到密文c $m^e \equiv c \pmod{n}$
- 用私钥对c进行解密得到明文m $c^d \equiv m \pmod{n}$
等价于 $m = \text{pow}(c, d, n)$

你是否熟悉RSA算法？敌军正在用RSA算法加密，但是防范不周，被我军获取了部分信息。请解密密文是981， $w = 13$ ， $n = 2537$ ，分解式的一个因子是43的明文。

明文=981的937次方，再mod2537 最后等于704
d是937、e13，C=981、

► RSA攻击常用工具

□ 整数分解

- factordb (<http://www.factordb.com/>)
- Msieve (<http://sourceforge.net/projects/msieve/>)
- Yafu (<https://sourceforge.net/projects/yafu/>)
- RSA-Tool 2

□ 高级攻击

- Sage (python语法, <http://www.sagemath.org/>)
- PARI/GP (<https://pari.math.u-bordeaux.fr/>)

► 分解模数破解RSA

- 主要是因为模数N比较小造成

```
root@kali:~/Desktop/rsa1# openssl rsa -in pub.key -pubin -text -modulus
Public-Key: (256 bit)
Modulus:
    00:bb:e7:2d:39:bb:c0:6a:e8:7b:9f:51:b7:76:64:
    65:61:d4:5f:af:dc:50:1a:51:e1:c8:c7:c6:cf:1b:
        f0:30:2d
Exponent: 65537 (0x10001)
Modulus=BBE72D39BBC06AE87B9F51B776646561D45FAFDC501A51E1C8C7C6CF1BF0302D
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAItALvnLTm7wGroe59Rt3ZkZW HUX6/cUBpR
4cjHxs8b8DATAgMBAE=
-----END PUBLIC KEY-----
```

```
root@kali:~/Desktop/rsa1# python -c "print int('0xB E72D39BBC06AE87B9F51B7766465
61D45FAFDC501A51E1C8C7C6CF1BF0302D',16)"
84990956492109741564425379376173601334652761525287275608009611600191290355757
root@kali:~/Desktop/rsa1#
```

► 分解模数破解RSA

yafu中的factor()函数

线下赛常用

```
root@kali:~/Desktop# ./yafu

09/19/16 11:50:11 v1.34.5 @ kali, System/Build Info:
Using GMP-ECM 6.4.4, Powered by GMP 5.1.1
detected Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz
detected L1 = 32768 bytes, L2 = 134217728 bytes, CL = 64 bytes
measured cpu frequency ~= 2495.020430
using 20 random witnesses for Rabin-Miller PRP checks

=====
===== Welcome to YAFU (Yet Another Factoring Utility) =====
===== bbuhrw@gmail.com =====
===== Type help at any time, or quit to quit =====
=====
cached 78498 primes. pmax = 999983

>> factor(84990956492109741564425379376173601334652761525287275608009611600191290355757)

fac: factoring 84990956492109741564425379376173601334652761525287275608009611600191290355757
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
rho: x^2 + 3, starting 1000 iterations on C77
rho: x^2 + 2, starting 1000 iterations on C77
rho: x^2 + 1, starting 1000 iterations on C77
pm1: starting B1 = 150K, B2 = gmp-ecm default on C77
ecm: 30/30 curves on C77, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C77, B1=11K, B2=gmp-ecm default
ecm: 149/149 curves on C77, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c77: 84990956492109741564425379376173601334652761525287275608009611600191290355757

==== sieving in progress (1 thread): 36224 relations needed ====
==== Press ctrl-c to abort and save state ====
36263 rels found: 18174 full + 18089 from 191344 partial, (2020.38 rels/sec)

SIQS elapsed time = 105.1662 seconds.
Total factoring time = 121.3087 seconds

***factors found***

P39 = 323067951880962860113901833788589140869
P39 = 263074551335953569706833061753492041353

ans = 1
```

► 分解模数破解RSA

□ 用rsatool.py生成私钥

```
root@kali:~/Desktop/rsatool# python rsatool.py -p 323067951880962860113901833788
589140869 -q 263074551335953569706833061753492041353 -o priv1.key
Using (p, q) to initialise RSA instance

n =
bbe72d39bbc06ae87b9f51b776646561d45fafdc501a51e1c8c7c6cf1bf0302d

e = 65537 (0x10001)

d =
746f21c51ea4731ab04fdef11239cfdfa7dc56771cf7df97215d41ef0cc44041

p = 323067951880962860113901833788589140869 (0xf30c9fcf7fdaed9389739b687df0eb85)

q = 263074551335953569706833061753492041353 (0xc5ea50cb51e2ffb3c57c375369436e89)

Saving PEM as priv1.key
```

► 分解模数破解RSA

□ 用openssl进行解密

```
root@kali:~/Desktop/rsa1/decrypt# cat flag.enc > c && openssl rsautl -in c -inkey priv1.key -out flag1.txt -decrypt
root@kali:~/Desktop/rsa1/decrypt# cat flag1.txt
flag{Ciph3r_w0r1d}
root@kali:~/Desktop/rsa1/decrypt#
```

□ 技巧

openssl rsautl -decrypt -inkey pri.pem -in passwd.enc **-oaep**

openssl rsautl -decrypt -inkey pri.pem -in passwd.enc **-pkcs**

► 公约数利用

- 两个N有共同的素数因子，即： $(n_1, n_2) = p$

$$n_1 = pq_1$$

$$n_2 = pq_2$$

$\gcd(n_1, n_2)$ 解得最大公约数p即可

▶ 广播攻击

- e较小，用相同的e和不同的n加密线性相关的消息

```
1 from functools import reduce
2 import gmpy
3 from libnum import *
4 import json, binascii
5 def modinv(a, m):
6     return int(gmpy.invert(gmpy.mpz(a), gmpy.mpz(m)))
7
8 def chinese_remainder(n, a):
9     sum = 0
10    prod = reduce(lambda a, b: a*b, n)
11    for n_i, a_i in zip(n, a):
12        p = prod // n_i
13        sum += a_i * modinv(p, n_i) * p
14    return int(sum % prod)
15
16 with open("enc.txt",'rb') as dfile:
17     data = json.loads(dfile.read())
18
19 data = {k:[d.get(k) for d in data] for k in {k for d in data for k in
20 . d}}
21 t_to_e = chinese_remainder(data['n'], data['c'])
22 t = int(gmpy.mpz(t_to_e).root(10)[0])
23 print hex(t)[2:-1].decode('hex')
```

中国剩余定律的利用
4组的时候

▶ 中国剩余定律

□ 有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何？

由于 $(a+kb) \% b = c$ (k 为非零整数)， $n_1 + n_2 + n_3$ 需要满足：

$$n_1 \% 3 = k_1 \dots 2$$

➤ 使 $n_1 + n_2 + n_3$ 的和满足除以3余2， n_2 和 n_3 必须是3的倍数

$$n_2 \% 5 = k_2 \dots 3$$

➤ 使 $n_1 + n_2 + n_3$ 的和满足除以5余3， n_1 和 n_3 必须是5的倍数

$$n_3 \% 7 = k_3 \dots 2$$

➤ 使 $n_1 + n_2 + n_3$ 的和满足除以7余2， n_1 和 n_2 必须是7的倍数

然后有：

➤ n_1 除以3余2，且是5和7的公倍数

➤ n_2 除以5余3，且是3和7的公倍数

➤ n_3 除以7余2，且是3和5的公倍数

由于 $(a * k) \% b = kc$
则找到余1，乘以N即可

▶ 共模攻击

- m相同，n相同，e不同，有：

$$c_1 \equiv m^{e_1} \pmod{n}$$

$$c_2 \equiv m^{e_2} \pmod{n}$$

- 此时有： $(e_1, e_2) = 1$
- 且有： $s_1 e_1 + s_2 e_2 = 1$
- 则有： $c_1^{s_1} c_2^{s_2} \equiv m \pmod{n}$

通过扩展欧几里得算法得到s1和s2
 $m = (\text{pow}(c_1, s_1, n) * \text{pow}(c_2, s_2, n)) \% n$

```
def egcd(a, b):  
    if a == 0:  
        return (b, 0, 1)  
    else:  
        g, y, x = egcd(b % a, a)  
        return (g, x - (b // a) * y, y)  
  
def modinv(a, m):  
    g, x, y = egcd(a, m)  
    if g != 1:  
        raise Exception('modular inverse does not exist')  
    else:  
        return x % m
```

▶ 低加密指数攻击

- 正常的加密流程 : $c \equiv m^e \pmod{n}$
- 当 $e=3$ 这样比较小的值的时候，则有 $c = m^e, e = 3$
- 则有 : $m = \sqrt[3]{c}$

- 若 $m^e > n$ ，但是大不是很多，那么有 $c = m^e + kn$
- 只要爆破 k ，使得 $c - kn$ 能够开三次方，就得到 m 了

► Wiener attack

其实就是低解密指数攻击

当e很大（1024bit以上）的时候，d肯定很小：

$$d * e \equiv 1 \pmod{\varphi(n)}$$

攻击条件：

$$d < \frac{1}{3}gn^{\frac{1}{4}}$$

$$q < p < 2q$$

<https://github.com/pablocelayes/rsa-wiener-attack>

```
1 | $ python wiener_attack.py -e 0x0285F8D4FE29CE11605EDF221868937C1B70AE376E34D
2 | -p 1200130412901548016543287507443760793349385061149987946484524335021517614
3 | -q 2821611731692987406749588802776752701136066162248684276841405995157293214
4 | -e 3185406653793934699014566658072115090777557199958115200390952121394292380
```

06

自定义加密

自定义加密技巧

▶ 自定义加密

- 结合各种ACM算法/加解密算法编写加密程序
- 选手需要从已知密文和算法程序中解得明文
- 难点：需要编程能力

· 命题和解题报告——《Steins Gate》 ↵

- 类别：Crypto ↵
- 考察：算法破解能力+爆破思想 ↵
- 提示：无 ↵

· 题目 ↵

你目前正在处于 beta 世界线，跨越 1%的世界线变动率，进入命运石之门世界线
逃脱世界线收束，获取你的 flag ↵

Note : flag.txt 中存储着一个完整的 flag，flag 标准格式为：flag[xxxxxxxx] ↵

下载：SteinsGate.a6493c815b65ad709684efa88f75a9ac ↵

▶ 自定义加密

```
function encode($str){  
    $_o = strrev($str);  
    for($_0=0;$_0<strlen($_o);$_0++){  
        $_c = substr($_o,$_0,1);  
        $_ = ord($_c)+1;  
        $_c = chr($_);  
        $_ = $_.$_c;  
    }  
    return str_rot13(strrev(base64_encode($_)));  
}
```

```
>>> s='=cWbihGfJBHegUWMgpXM2BWZmRWMIZGYqUnf'  
>>> s=s[::-1]  
>>> s  
'fnUqYGZIMWRmZWB2MXpgMWUgeHBJfGhibWc='  
>>>
```

```
>>> a='~uj`fH1dfe`v1z`1e`xpl:hbmg'  
>>> a=a[::-1]  
>>> a  
'gmbh!Ipx`e1`z1v`efd1Hf`ju~'  
>>>
```

```
>>> l=list(a)  
>>> for a in l:  
...     a=chr(ord(a)-1)  
...     print a,  
...  
flag{How_d0_y0u_decrypt_it}  
>>>
```

▶ Steins;Gate

1.对t0的求解。

其中t0的值和操作系统平台(Windows/Linux/Darwin), 字长位数(32/64),当前用户uid, 随机小数r(范围0-2,且最多6位小数), m的md5值有关。

t0	OS	字长位数	uid	r	32位md5(m)
m	字长位数	'bit'	uid	r1	
r	个位 (0或1)	小数位 (最多6位)			
r1	个位 (0或1)	小数位 (最多2位)		ChaMd5安全团队	

这里还可以知道t0的位数应该在48左右(5+2+2+7+32)

2.对t1的求解及使用

t1的值由flag的值和t0异或得到(位数也在48左右),之后算法使用了给的图片文件(DivergenceMeter.jpg),在文件中找和t1各个字符(t1[i])相同的值img[k],并提取索引值k,转换为字符(k与255取模保证在ascii码范围内),遍历t1生成表l, l的结构大致如下:

t1[i]	索引	列表(data)
t1[0]	0	chr(k1),chr(k2),...,chr(kn)(其中k1,..kn为图片中和t1[0]相同的字符所在位置)
...
t1[n]	n	chr(k1),chr(k2),...,chr(kn)(其中k1,..kn为图片中和t1[n]相同的字符所在位置)

得到表l后,纵向遍历该表,并把读取数据写入result,最终得到加密后的flag文件flag.enc。

▶ 自定义加密解题技巧

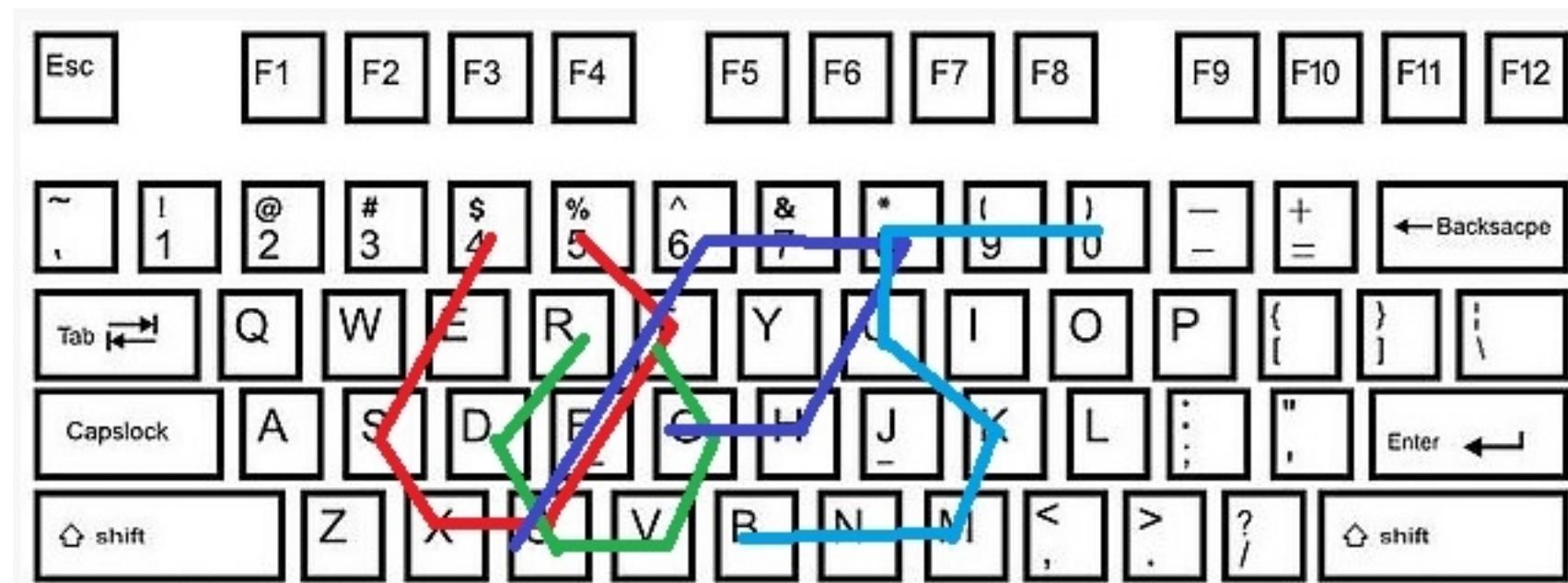
- 通读加密逻辑，找出所有的既定内容
- 对于随机数
 - 再生成对应范围的随机数进行爆破（一般都有范围）
- 对于余数
 - $k + (p * n)$ ，我们已知余数k，以及除数p，那么商n则需要爆破
- 对于合理性判断
 - 明文和密文都必须是可见字符，这是进行判断的基本条件
 - 对于中间值，可能不是可见字符，但是只要能计算下去就可能是对的
- 抽丝剥茧，分步进行

07

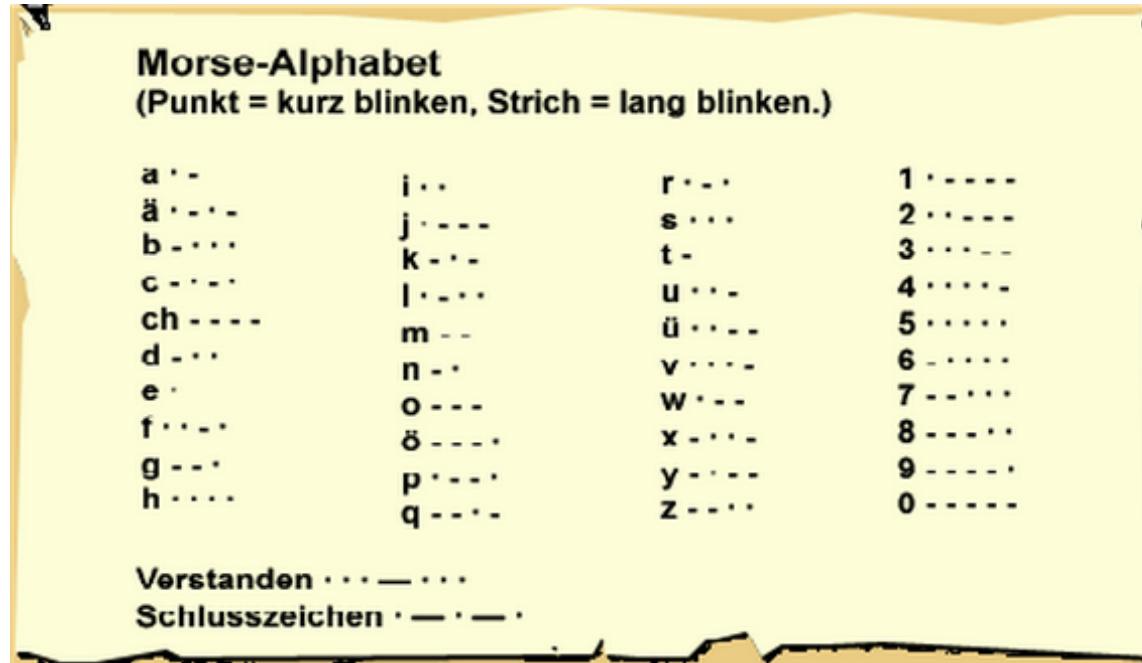
特殊密码

▶ 键盘密码

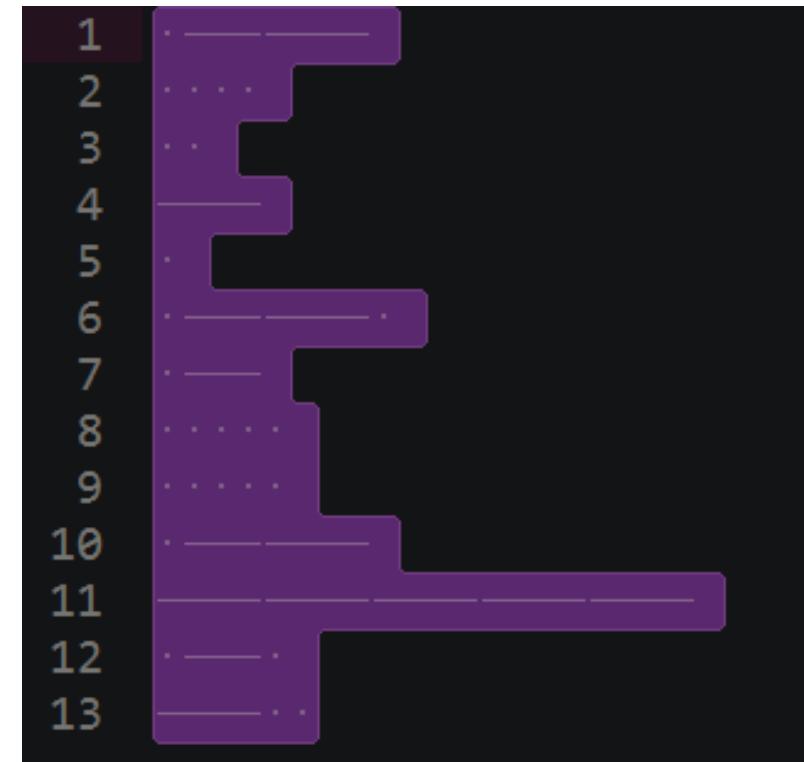
4esxcft5 rdcvgt 6tfc78uhg 098ukmnb



▶ 莫斯密码



莫斯密码最大特点是二值化



▶ 培根密码

每个明文字符被一个由5字符组成的序列替换

A = aaaaa	I/J = abaaa	R = baaaa
B = aaaab	K = abaab	S = baaab
C = aaaba	L = ababa	T = baaba
D = aaabb	M = ababb	U/V = baabb
E = aabaa	N = abbaa	W = babaa
F = aabab	O = abbab	X = babab
G = aabba	P = abbba	Y = babba
H = aabbb	Q = abbbb	Z = babbb

明文: T H E F O X

密文: baaba aabbb aabaa aabab abbab babab

► Jefferson disk

加密表:

1:	< ZWAXJGDLUBVIQHKYPNTCRMOSFE <
2:	< KPBELNACZDTRXMJQOYHGVSFUWI <
3:	< BDMAIZVRNSJUWFHTEQGYXPLOCK <
4:	< RPLNDVHGFCUKTEBSXQYIZMJWA0 <
5:	< IHFRLABEUOTSGJVDKCPMNZQWXY <
6:	< AMKGHIWPNYCJBZFZDRUSL0QXVET <
7:	< GWTHSPYBXIZULVKMRAFDCEONJQ <
8:	< NOZUTWDCVRJLXKISEFAPMYGHBQ <
9:	< XPLTDSRFHENYVUBMCQWA0IKZGJ <
10:	< UDNAJFB0WTGVRSCZQKELMXYIHP <
11:	< MNBVCXZQWERTP0IUYALSKDJFHG <
12:	< LVNCMXZPQ0WEIURYTASBKJDFHG <
13:	< JZQAWSXCDERFVBGTYHNUMKIL0P <

密钥为: 2, 3, 7, 5, 13, 12, 9, 1, 8, 10, 4, 11, 6

密文为: NFQKSEVOQ0FNP

2:	< N ACZDTRXMJQOYHGVS F JWIKPBEL <
3:	< F HTEQGYXPLOCKBDMA I ZVRNSJUW <
7:	< Q GWTHSPYBXIZULVKM R AFDCEONJ <
5:	< K CPMNZQWXYIHFR LAB E JOTSGJVD <
13:	< S XCDERFVBGTYHNUMK I LOPJZQAW <
12:	< E IURYTASBKJDFHGLV N CMXZPQOW <
9:	< V UBM CQWA0IKZGJXPL T DSRFHENY <
1:	< O SFEZWAXJGDLUBVIQ H KYPNTCRM <
8:	< Q NOZUTWDCVRJLXKIS E FAPMYGH B <
10:	< O WTGVRSCZQKELMXYI H PUDNAJFB <
4:	< F CUKTEBSXQYIZMJWA O RPLNDVHG <
11:	< N BVCXZQWERTPOIUYA L SKDJFHGM <
6:	< P NYCJBZFZDRUSL0QXV E TAMKGHIW <

ColecoVision

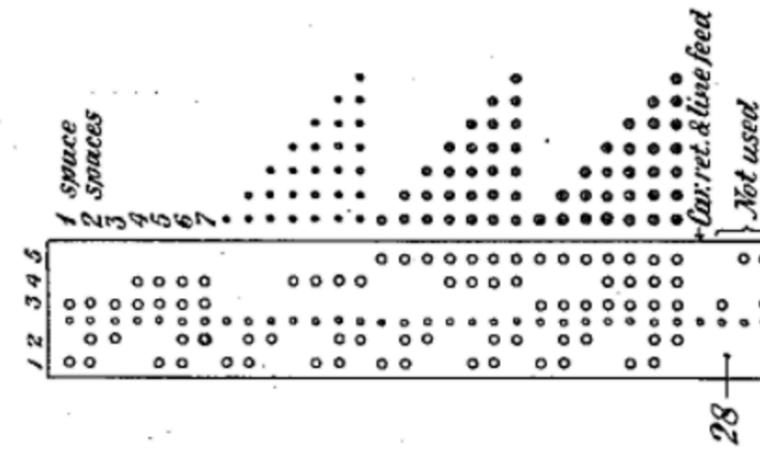
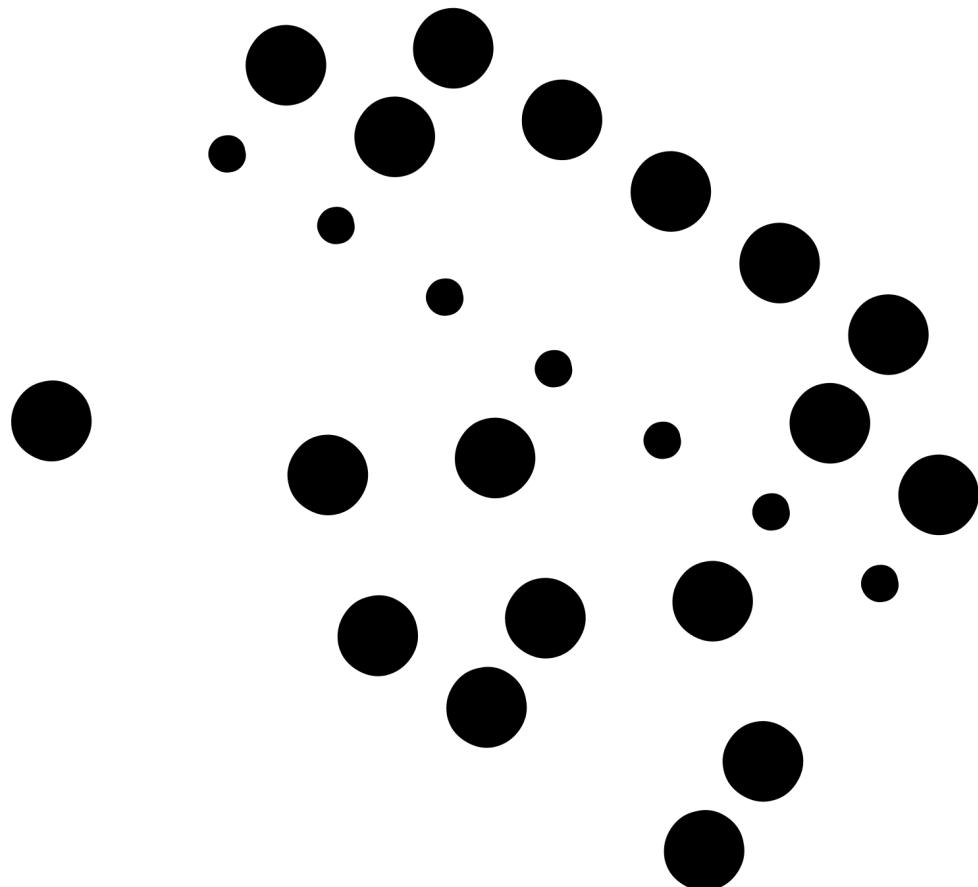
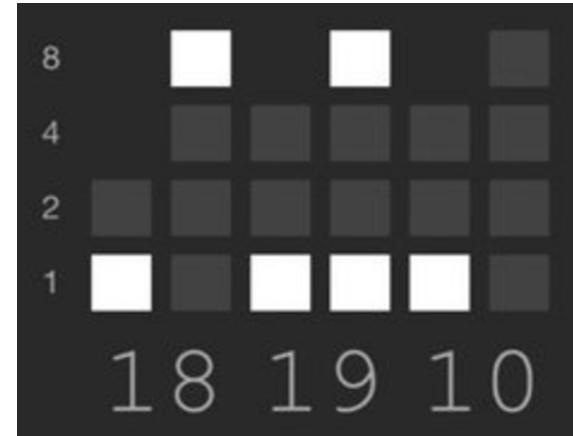


Fig. 2

► 二进制时钟



谢谢！

