Introduction to Modern Cryptography

Michele Ciampi

(Slides courtesy of Prof. Jonathan Katz)

Lecture 01 Part 3

Shift Cipher

The Shift Cipher

- Consider encrypting English text
- Associate $\boldsymbol{a} \leftarrow \mathbf{0}$, $\boldsymbol{b} \leftarrow \mathbf{1}$, ... $\boldsymbol{z} \leftarrow \mathbf{25}$
  - $\boldsymbol{k} \in \mathcal{K} = \{\mathbf{0}, \ldots, \mathbf{25}\}$
- To encrypt using key $\boldsymbol{k}$, shift every letter of the plaintext by $\boldsymbol{k}$ positions (with wraparound)
- Decryption just does the reverse

```
helloworldz
ccccccccccc
jgnnqyqtnfb
```

Modular arithmetic

- $x = y \mod N$ if and only if N divides $x - y$
  - $[x \mod N]$ = the remainder when x is divided by N
  - i.e. the unique value $y \in \{0, \ldots, N-1\}$ such that $x = y \mod N$
- $25 = 35 \mod 10$
- $25 \neq [35 \mod 10]$
- $5 = [35 \mod 10]$

The Shift Cipher, formally

- $\mathcal{M}$ = strings over lowercase English alphabet
- Gen: choose uniform $k \in \{0, \ldots, 25\}$
- $\mathsf{Enc}_k(m_1 \ldots m_t)$: output $c_1 \ldots c_t$, where

$$c_i = [m_i + k \mod 26]$$

- $\mathsf{Dec}_k(c_1 \ldots c_t)$: output $m_1 \ldots m_t$, where

$$m_i = [c_i - k \mod 26]$$

Is the Shift Cipher secure?

Brute-force Attack

- No – only **26** possible keys!
- Given a ciphertext, try decrypting with every possible key
- Only one possibility will "make sense"
- Example of a **brute-force** or **exhaustive-search** attack

Brute-force Attack on Shift Cipher

Example

- Ciphertext `uryybjbeyq`
- Try every possible key and decrypt:
  - message under key **1** is: `tqxxaiadxp`
  - message under key **2** is: `spwwzhzcwo`
  - ...
  - message under key $\boldsymbol{i}$ is: `helloworld`
  - ...

Byte-wise Shift Cipher

- Alphabet of **bytes** rather than (English, lowercase) letters
- Works natively for arbitrary data!
- Use `XOR` instead of modular addition
- Essential properties still hold

Hexadecimal (base **16**) Notation

Hex	Bits	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Hex	Bits	Decimal
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Hexadecimal (base **16**) Notation

0x10

- `0x10 = 16*1 + 0 = 16`
- `0x10 = 0001 0000`

0xAF

- `0xAF = 16*A + F = 16*10 + 15 = 175`
- `0xAF = 1010 1111`

ASCII

- American Standard Code for Information Interchange
- Character encoding standard
- Byte-wise Shift Cipher: encode characters in ASCII
- **1** byte = **1** character = **2** hex digits
- Encoded using the **ASCII table**

ASCII table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

https://hubpages.com/technology/What-Are-ASCII-Codes

Useful observations

- Only **128** valid ASCII chars (**128** bytes invalid)
- Only `0x20-0x7E` printable
- `0x41-0x7A` includes upper/lowercase letters
- Uppercase letters begin with `0x4` or `0x5`
- Lowercase letters begin with `0x6` or `0x7`

Byte-wise Shift Cipher, Formally

- $\mathcal{M}$ = strings of bytes
- Gen: choose uniform $k \in \mathcal{K} = \{\texttt{0x00} \ldots \texttt{0xFF}\}$ i.e. there are **256** possible keys
- $\textsf{Enc}_k(m_1 \ldots m_t)$: output $c_1 \ldots c_t$, where

$$c_i = m_i \oplus k$$

- $\textsf{Dec}_k(c_1 \ldots c_t)$: output $m_1 \ldots m_t$, where

$$m_i = c_i \oplus k$$

Is this scheme secure?

- No – only **256** possible keys!
- Given a ciphertext, try decrypting with every possible key
- If ciphertext is long enough, only one plaintext will ”make sense”
- Can further optimize
  - First nibble of plaintext likely `0x4,0x5,0x6,0x7` (assuming letters only)
  - Recover **2** key bits and reduce exhaustive search by a factor of 4.

Sufficient key space principle

Crypto Design Lesson One

- The key space must be large enough to make brute-force attacks impractical

Spoiler: necessary, but not sufficient

End

Reference: Section 1.3 of the book