CMSC 201 Fall 2015
Lab 10 – Caesar Cipher

Assignment: Lab 10 – Caesar Cipher
Due Date: During discussion, November 2nd through November 5th
Value: 1% of final grade

Part 1: Introduction to Information Security

Protecting data and even more importantly information is a concept that is common throughout headlines today. Articles ripped from today's headlines often read like this: "Apple apps ranked as biggest security risk on the PC" or "Are people really your biggest cyber security risk?" So what can we do as computer scientists to help reduce the risk of our data being compromised?

There are three main concepts that are at the heart of information security (InfoSec). They are Confidentiality, Integrity, and Availability (or CIA for short). Figure 1 shows these information security goals.

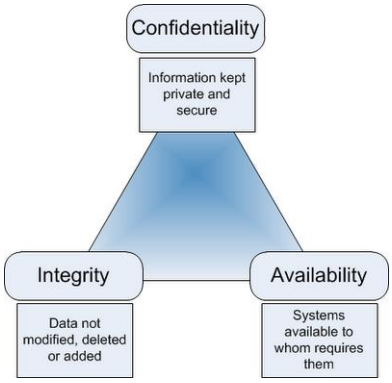

Figure 1. Information Security Goals (Williams, 2012)

Confidentiality: Information kept private and secure.
Integrity: Data not modified, deleted, or added.
Availability: Systems available to whom requires them.

For this particular lab, we are going to focus on confidentiality which is related to keeping our information private and secure. Many current system vulnerabilities are related to not keeping our data confidential such as when Target had a data breach or Ashley Madison was hacked.

Part 2: Cryptology, Encryption, and Decryption

As this is just a general introduction to some of the most basic ways to help secure our data, we are not going to delve into the complex methods of protecting data that are currently employed in the information assurance sector. Rather, we will introduce some of the key concepts, and the lab will entail coding a simple encryption/decryption system to protect our data.

Cryptology: The practice and study of protecting our data from unwanted users is what we consider to be cryptology. We have been employing simple cryptologic methods for securing our information for thousands of years. Our lab will use an encryption method that was documented to have been used by Julius Caesar as far back as 44 B.C.E.

Encryption: This is one way to help secure our data, by converting it from something readable to something that appears to be nonsense. Converting the word “Hello” to “Lfmmp” is an example of a simple 1 character shift (or Caesar Cipher). H->L, e->f, l->m, l->m, and o->p.

Decryption: In order to be able to read something that has been encrypted, we need to reverse the algorithm so that we can read it again. To do this, we decrypt it. In our previous example, we did a 1 character shift from Hello to Lfmmp. To decrypt it, we would need to shift the encrypted message back from Lfmmp to Hello.

As you can see, the idea of basic encryption and decryption isn’t particularly complicated. Cartoons and cereal boxes have been using things like secret decoder rings as prizes or toys for decades.

Part 3: ASCII

As we have discussed in class, computers are not particularly good at storing characters or strings directly in memory (mostly because they can only speak binary!) so they convert the characters into a number. This is particularly interesting because they using a *character-encryption* scheme to do this.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[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	[NEGATIVE 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	[ENG OF 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	[SUBSTITUTE]	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	[DEL]

Table 1. ASCII Table (Wikipedia, 2015)

Now, the notion of character-encryption sounds more complicated than it really is. What we are doing is very simple. We are mapping each character on a standard keyboard to a specific number. If you take a look at Table 1 above, you will see that the first 32 characters are control characters and therefore do not print anything. The next 95 characters are printable and are the ones that we are most familiar with.

If you look at Table 1, we can see the decimal equivalent that each character is converted to. For example, an “8” is 56. The letter “T” is 84. The character “y” is 121. Another way to think about this is that the number “8” is stored in 7-bit binary as 0011 1000. The letter “T” is 0101 0100. The “y” is 0111 1001.

If you would like to see the entire ASCII chart with the decimal or binary equivalent, check out this page: https://en.wikipedia.org/wiki/ASCII

Part 4: Useful Python Functions

If we were interested in knowing what the decimal equivalent of a character was, we could use a built-in functions to convert it. Similarly, if we wanted to find out the character equivalent of a decimal number, we could use a built-in function to convert it.

If we want to convert a number to a character, we could use the built-in function called **`chr()`**

If we want to convert a character to a decimal number, we could use the built-in function called **`ord()`**

Let's look at an example:

Start with a simple example ord.py

```
# FILE: ord.py
# PURPOSE: Convert integers into characters

a = 66
b = 65
c = 71

print(chr(a)+chr(b)+chr(c))
```

```
bash-4.1$ python ord.py
BAG
bash-4.1$
```

Notice how the interpreter took an integer and converted it to a character.

We can also start with the same values using a character and convert it to a decimal using **`ord()`**

```
# FILE: chr.py
# PURPOSE: Convert characters into integers

a = "B"
b = "A"
c = "G"
print(ord(a),ord(b),ord(c))
```

```
bash-4.1$ python chr.py
66 65 71
bash-4.1$
```

The numbers print out as expected. One thing that we need to be careful of is performing concatenation when using **`ord()`**. Because **`ord()`** returns an integer, using the **`+`** will return the sum of those numbers!

```
# FILE: chr2.py
# PURPOSE: Convert characters into integers
#          Use concatenation when printing

a = "B"
b = "A"
c = "G"
print(ord(a)+ord(b)+ord(c))
```

```
bash-4.1$ python chr2.py
202
bash-4.1$
```

In this example, we are concatenating using the **`+`** sign. As mentioned, **`ord()`** returns integers so when we add up 66+65+71 = 202. We don't want the sum!

Part 5: Caesar Ciphers

As we have mentioned previously, we have utilized ciphers to communicate secrets across distances for millennia. Why did they use them? Well, if we have a soldier on the other side of the country and are relaying a message that might put their life at risk, do we want that message falling into the wrong hands? No! So what can we do? The simple answer is to encrypt the message.

In modern times, we encrypt almost all data that is transmitted. Packets that are sent back and forth from your phone or your laptop are encrypted and sent to your router. Data is sent back to your device and decrypted. It happens all the time and, for the most part, we do not even know that it is happening. However, because computers are always improving and our ability to "crack" encrypted messages is always improving, we are also constantly trying to improve our encryption methods.

Before you can plunge in and understand things like AES and RSA encryption, we need to start with something basic like Caesar ciphers. For Caesar ciphers, we are going to encrypt normal strings and decrypt encrypted strings.

A Caesar cipher is pretty simple. It is a substitution cipher that people like to think was used by Julius Caesar. (There is even some documentation that suggests he did use it!). In this cipher we follow one formula to encrypt a message and another to decrypt. Let us take a look at what that means mathematically.

Here is our encryption formula:

$$E_n(x) = (x + n)$$

```
if En(x) > "Z":
    En(x) = En(x) - 26
```

Where x is the character and n is the modifier. For our encryption formula, we need to make sure we don't go "beyond" the end of the alphabet. So we check if the newly encrypted letter is "after" the letter "Z", adjusting it if it is.

Here is our decryption formula:

$$D_n(x) = (x - n)$$
$$\text{if } D_n(x) < \text{"A"}:$$
$$\quad D_n(x) = D_n(x) + 26$$

Where x is the character and n is the modifier. For our decryption formula, we need to make sure we don't go "before" the beginning of the alphabet. So we check if the newly decrypted letter is "before" the letter "A", adjusting it if it is.

So how would we implement this from a coding standpoint? As we have discussed in the previous parts of the lab, we can convert a character into an integer. A string, of course, is a series of characters, which means we can also convert it into a series of integers.

At this point, we will have a whole sequence of integers. As these are integers, we can choose a value to change the integers.

For example, say we are trying to encrypt the character "U".

1. First, we could convert the "U" to an integer using the **`ord()`** function.
2. We could take the integer equivalent to "U" (in this case, 85) and add a specific number to the integer equivalent. Say we try and add 2 to the integer 85. Now we have 87.
3. Now we can use the **`chr()`** command to convert the 87 back to a character. The decimal equivalent to 87 is "W".

Think about it. As long as we know that we modified it by 2, we could reverse the formula and return the original value.

Part 6: Creating a Caesar Cipher

For the hands-on part of the lab, we are going to use our techniques to try and generate the code required to create a Caesar Cipher. For this effort, you will be building a **`main()`** function along with two additional functions called **`encrypt()`** and **`decrypt()`**.

To keep things simple, your program only needs to work with capital letters, and does not need to handle spaces.

Below, we have provided the pseudocode to build each of the two functions.

`encrypt(n)`

1. The call to **`encrypt(n)`** will need to take in one argument: an integer representing by how much you would like to shift the string.
2. Next, **`encrypt(n)`** will ask the user for the string to encrypt.
3. Then **`encrypt(n)`** will loop through each character in the input string.
4. During the loop, each character will need to be shifted by the number of characters that was passed to the encrypt function (n). This would probably look something like:
   `ord(current character) + amount to shift by`
5. After shifting each character, you will need to check that you haven't gone past the "end" of the alphabet. You can do this by checking to make sure that the shifted value is not greater than **`ord("Z")`**.
   a. If it is greater than **`ord("Z")`**, you should subtract 26 from the value to get it back "into" the alphabet.
6. Finally, you should print the results of the encryption.

```
bash-4.1$ python lab10.py
What would you like to do?
1. Encrypt String
2. Decrypt String
Enter 1 or 2: 1
How much would you like to shift? (1-26): 2
Enter plain text message: ATTACKATDAWN
Your encrypted message is: CVVCEMCVFCYP
bash-4.1$
```

`decrypt(n)`

1. The call to **`decrypt(n)`** will need to take in one argument: an integer representing by how much you would like to shift the string. (You should ask the user for a positive integer, but you will use it to shift "backwards".)
2. Next, **`decrypt(n)`** will ask the user for the string to encrypt.
3. Then **`decrypt(n)`** will loop through each character in the input string.
4. During the loop, each character will need to be shifted by the number of characters that was passed to the encrypt function (n). This would probably look something like:

   `ord(current character) - amount to shift by`

   (Notice that we are now ***subtracting*** the shift amount from the ord.)
5. After shifting each character, you will need to check that you haven't gone past the "beginning" of the alphabet. You can do this by checking to make sure that the shifted value is not ***less than*** **`ord("A")`**.
   a. If it is less than **`ord("A")`**, you should ***add*** 26 to the value to get it back "into" the alphabet.
6. Finally, you should print the results of the decryption.

```
bash-4.1$ python lab10.py
What would you like to do?
1. Encrypt String
2. Decrypt String
Enter 1 or 2: 2
How much would you like to shift? (1-26): 2
Enter encrypted message: CVVCEMCVFCYP
Your plain text message is: ATTACKATDAWN
bash-4.1$
```

Part 8: Completing Your Lab

To test your program, first enable Python 3, then run **`lab10.py`**. Test your code to make sure that it runs both encryption and decryption functions. Take the output of the encryption function to make sure that the decryption function returns what you originally input.

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave.

References:

Wikipedia (2015). "ASCII". Retrieved from
https://simple.wikipedia.org/wiki/ASCII
Williams, G. (2012). "CIA & InfoSec". Retrieved from
http://geraintw.blogspot.com/2012/09/cia-infosec.html