CS466 Project 1

Cracking and Creating Ciphers

Dr. Andrew Scott

Introduction:

For this project you will work in pairs to try and crack some cipher text as well as write a One-Time Pad cipher, and a Stream Cipher. This document begins by describing the operations of a Shift, Substitution, Vigenère, One-Time Pad, and Stream Cipher, then come the tasks. For the first task you will attempt to crack some provided ciphertext, to deduce its plain text or at least some characteristics of the ciphertext or the plain text is was derived from. For the second task you will try your hand at writing a One-Time Pad cipher which will be a stepping stone to the third task which is to write a stream cipher.

Shift Cipher: (Key type, shift amount)

A shift cipher, also known as a Ceasar cipher is one of the simplest and most widely known encryption techniques. It is sometimes called the Cesar Cipher because it was purported to be used by Julias Cesar for the exchange of sensitive miliary messages.

The shift cipher works by shifting the replacing each letter of the plain text with another one a fixed number of places down or up the alphabet. For example, the following has a shift of three.

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	.	
	.	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

Therefore, an encrypted message will have looks follows.

COBBYZLCCBBYQLYQEBYCFOPQYMBOPLKYQRYTLOHYQEFPYLRQ

In the above examples, we just used the letters A-Z and a period and space. However, this shift could compass all valid ASCII characters or more. For example, with ASCII characters and shift of -3 my name Andrew Scott, becomes: `>kaobt.|P`lrr`

Although the ASCII table contains extra control characters and non-visible characters such as delete, we ignore those to only include offsets 32 to 126, and wrap around so that with a shift of -3 ! becomes {, or a shift of 5 } becomes #.

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_					

Substitution Cipher: (Key type: substitute alphabet)

A substitution cipher is another form of simple cipher where one symbol is exchanged for another at random.

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

Therefore, with the above table, my name ANDREW SCOTT would produce the cipher text: HKYTSANEXGUU.

Like the shift cipher, this table could be expanded to encompass all 94 visible characters of the ASCII table or more such as Unicode characters.

Vigenère Cipher: (Key Type: text >= message length)

The Vigenère cipher is a method of encrypting alphabetic text where each letter of plain text is encoded with a different shift cipher, for which the increment is determined by the corresponding letter of another text, called the key. If the recipient knows the key, they can decipher the plain text.

For example, if the plain text is `HAMMER TIME` and the key is OSCILATINGFANSAREEVIL, then:

- The first letter of the plain text is shifted by 14 positions in the alphabet because the first letter of the key O, the 14th letter of the alphabet (counting from O), producing V.
- The second letter is shifted by 18, because the second key value is S, producing S
- The third letter is shifted by 3, because the third key value is C, yielding Q.
- The fourth letter is shifted by 9, because the fourth key value is I, producing W.

```
Plain text : HAMMER TIME
Key text   : OSCILATINGFANSAREEVIL
Cipher text: VSQWQBTAWTK
```

If for example the key value was Z, and the next character of plain text was a U. The shift would be 25 and thus with wrap around this would yield S.

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	26	27

With this cipher the length of the key should be equal to the message length or greater. Our key says something memorable, but more likely a random sequence should be used.

The One-Time Pad: (Key Type: text >= message length)

The one-time pad (OTP) is an early example of an encryption technique that is considered uncrackable. It requires the sender and the recipient to be in position of a securely distributed pad in which each page of both books contain identical random sequences. Each sheet of the pad will contain a random sequence and may also contain a serial number, or a date. To send a message the sender and recipient must agree on which sheet to use. The idea is for each to use the sheet and its key once for encryption and decryption of a message, and then to destroy the page.

With this technique each letter from the pad will be combined in a predetermined way with one letter of the message. It is common to assign each character a numerical value such as the following:

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	26	27

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	,	;
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	48	49	50	51	52	53	54	55	56	57	58

0	1	3	4	5	6	7	8	9
59	60	61	62	63	64	65	66	67

In our example below the value of the keys are combined with modular addition. Therefore, both parties need to agree on what modulus value to use. In this example, the message is "Death", the key is BDcKt, and the modulus value is 67 (67 is used as it's the size of the character list). This produces the cipher text Eh;1s. The modulo here is used so that if the two values sum to more than 67 (the last character in the table above), that it wraps around to the start.

```
                    MESSAGE:     3(D)     32(e)         28(a)       50(t)      35(h)
                  KEY VALUE:     1(B)      3(D)         30(c)       10(K)      50(t)
      MESSAGE + KEY_VALUE:        4         35           58          60         85
(MESSAGE + KEY_VALUE) mod 67: 4(E)       35(h)       58(;)       60(1)       18(S)
                CIPHERTEXT:     E          h           ;           1          S
```

The message `Eh1ls` is what would be sent. The recipient would then receive this message. Knowing the key BDcKt (via the pad), and the mod 67, the message can be deciphered using subtraction. To account for the modulus value, if the value of the ciphertext – key_value is < 0, then add the modulus value to it.

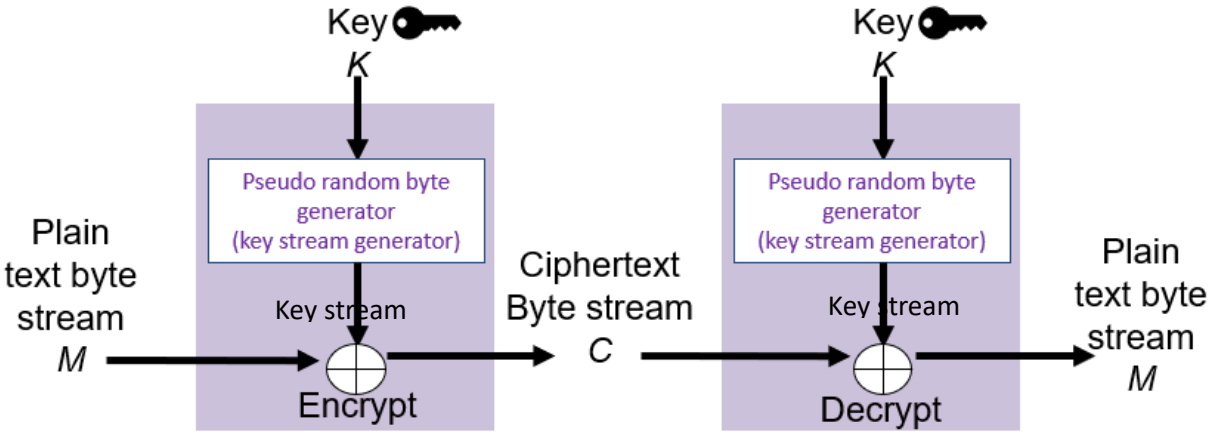
```
                CIPHERTEXT:     4(E)      35(h)       58(;)       60(1)       18(S)
                 KEY_VALUE:     1(B)       3(D)       30(c)       10(K)       50(t)
  CIPHERTEXT – KEY_VALUE:       3(D)      32(e)       28(a)       50(t)        -32 + modValue = 35(h)
                   MESSAGE:      D          e           a           t           h
```

The one significant limitation of the OTP is that the message length is limited by the size of the key used. Cycling of the values in the chosen OTP page could be achieved used, but conceivably this might reduce security.

Stream Cipher: (Key Type hex)

A stream cipher is similar to the OTP, except that it uses a pseudo random byte (or other size) generator with the key as input. It works like a seed value to then produce a pseudo random byte each time one is needed requested based off of that key. The values coming from the PRBG as needed is called the key stream.

The key stream is used in both the encryption and decryption process as the values generated are XORed with the plain text or cipher text.



The XOR operation in encryption:

The OTP style ciphers bear a similar trait to other ciphers, except that it uses modular addition. Instead many ciphers may use the XOR operation on the values of the plain text and key when both are represented as binary. For example, if we take the plain text DEAD, the key FACE, and use the table of the previous section to generate numbers represented in binary, our cipher text becomes HECH, when we XOR each value of the message with each corresponding value of the key.

```
   Message: 0011(D)  0100(E) 0000(A)   0011(D)
       Key: 0101(F)  0000(A)  0010(C)  0100(E)
   M XOR K: 0111(H)  0100(E)  0010(C)  0111(H)
Ciphertext:    H         E        C       H
```

What makes XOR interesting is, from the above three sequences of Boolean values, we only need two of the three binary values to the other value by XORing them together. So, if we now take our ciphertext and XOR it with the key, we can produce the original message. XAND has a similar property too. Below is an example with XOR.

```
Ciphertext:    0111(H)  0100(E)  0010(C)  0111(H)
       Key:    0101(F)  0000(A)  0010(C)  0100(E)
   Message:    0011(D)  0100(E) 0000(A)   0011(D)
```

Tasks:

For this lab, you will attempt to crack some encrypted text, as well as write your own encryption algorithms. Each pair has been provided with four cipher texts **alpha.enc**, **beta.enc**, **gamma.enc**, and **delta.enc**. The contents of these files will differ from team to pair, but all are based on the following character set. Your programs are also expected to utilize this character set, where A is 0 and \n is 77 a length of 78 characters. We will refer to this as our alphabet.

```
['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', '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', '(', ')', ':', '=',
'-', '+', '*','!','/','?', '\'' ,'\n']
```

Notice, the \' in the second to last element, that is just an apostrophe proceeded with an escape character so that python treats it as a character. See the next page for the description of the 3 tasks you will perform for this project.

Task 1: Crack as Many As You Can

Can you crack you're the four files to discover the plain text? Each team has been provided with an encrypted file which when decrypted contains a **zip** file with five files containing cipher text **alpha.enc, beta.enc**, **gamma.enc**, **delta.enc** and **epsilon.enc**. For each team the content of these files may differ greatly. Each file uses a different cipher, and the ciphers used are selected from those described in this handout. Crack as many of them as you can for which you may have to write some code which **must** be written in Python.

For each file whether successful or unsuccessful describe below your approach, and if successful the contents of the file.

Alpha.enc:

Paste plain text below if successful at cracking it:

In detail, describe your approach to cracking the file, what did you try and what (if anything) succeeded:

Beta.enc:

Paste plain text below if successful at cracking it:

In detail, describe your approach to cracking the file, what did you try and what (if anything) succeeded:

Gamma.enc:

Paste plain text below if successful at cracking it:

In detail, describe your approach to cracking the file, what did you try and what (if anything) succeeded:

Delta.enc:

Paste plain text below if successful at cracking it:

In detail, describe your approach to cracking the file, what did you try and what (if anything) succeeded:

Task 2: Write a One Time Pad Cipher:

Using the Python 3 language, write a python module that perform encryption and decryption using a one-time pad, as well as generation of a one-time pad. The is intended to be called from the command line and will work as follows:

Generating a one-time pad text file:

The following code will generate a one time pad of **100** values in the file **pad1.otp. g** must come first in the arguments list, -s denotes a size is coming next, -o denotes the output file name is coming next, and -s with their following value can be in any order.

```
python one_time_pad.py g  -s 100  -o pad1.otp
```

The values of your one-time pad file, must contain only the letters contained in our alphabet selected at random. . In the otp file a character from our alphabet can appear at random 0 to many times.

Note: Python's random library is not considered safe for cryptographic purposes. Therefore use Python's **Secrets** library and its **randbelow** function to select statistically random numbers below a specified integer.

Enciphering Text:

The following code will generate a ciphertext from a specified plain text and one time pad. Where -k tells your program a key is coming next, -f tells your program the name of the input file is coming next, and -o the name of the output file is coming next. Assume the text files and ciphertext files are in the same directory as the Python module. Assume the `-k value`, `-f value`, and `-o value` parameters can come in any order, but `e` must be first.

```
python one_time_pad.py e -k pad1.otp -f plain1a.txt -o cipher.enc
```

Deciphering Text:

The following code will decrypt a ciphertext from a specified file and one time pad. Where -k tells your program a key is coming next, -f tells your program the name of the input file is coming next, and -o the name of the output file is coming next. Assume the `-k`, `-f` , and `-o` parameters with their following values can come in any order, but `d` must be first.

Assume the text files and ciphertext files are in the same directory as the Python module.

```
python one_time_pad.py d -k pad1.otp -f cipher.enc -o plain1b.txt
```

Task 3: Write a stream cipher:

For this task, it is recommended you complete task 3 first and use its code as your basis. You will build a stream cipher. This cipher should not depend on the alphabet shown earlier, but accept any UTF-8 character.

To help you I will provide you with a pseudo random bite generator in the file **prb.py**. The PRBG was originally developed by John Clark Craig. It has been documented and commented by myself to help you understand its function and use. You can read more about this at either of the following two links below:

https://jccraig.medium.com/python-pseudorandom-bytes-143462fb0afe

click for alternate link:

Write your stream cipher so that not only can it be used from the command line, but also other programmers can use your code directly to encipher and decipher text in the way a stream cipher is often used. In this way once the programmer has initialized your cipher (called a function in your code to create the PRBG and pass it the key), they can encrypt and decrypt text all at once or on demand. Therefore functions should exist for encrypting and decrypting one character at a time, as well as a whole message (code for reuse).

Generating a Key:

Generate a key file for your stream cipher. The key_file should result in 64 bit key (8 byte) hexdecimal value. For example: 7affbdeb990f6ff7. To generate the key the user should type.

```
python stream_cipher.py g  -f key.stc
```

For this purpose, take a look at the Python library **Secrets**. In particular, it has a **token_hex** function. When generating this value you will notice an output is a hexadecimal value, the output of which can be written to a file as text.

Enciphering Text:

The following code will generate a ciphertext from a specified plain text and stream cipher.

```
python stream_cipher e -k key.stc -f plain.text -o cipher.enc
```

To make this work, you must employ a pseudo random byte generator that receives your key, and uses this to make random bytes which can be XORed with each character of the pain text to create your cipher text.

Deciphering Text:

The following code will decrypt a ciphertext into plain text using a key.

```
python stream_cipher d -k key.stc -f cpher.enc-o plain.txt
```

To make this work, you must employ a pseudo random byte generator that receives your key, and uses this to make random bytes which can be XORed with each character of the cipher text to create your plain text. For this

Task 3 Guidance:

In order to perform the **XOR** operation, you will need to represent the characters as bytes or integers (in Python they are treated similarly). You can use the **ord()** function to convert a character to an integer value representing its code point. Know too that the **char()** function is the opposite or ord.

Store the output of each enciphered character in a **bytearray,** and then using the **hex** function of byte arrays for saving the ciphertext as hexadecimal values. When loading the hexadecimal ciphertext in, use the **bytes.fromhex(hex_string)** function to convert it to a **bytearray**.

Grading:

In all areas of your work that requires coding you will be judged on the functional outcomes, code quality (think SOFA), code readability, reuse, documentation and commenting.

Task 1: Crack the ciphers: 0..30%

0..20% for cracking the shift ciphertext and code/method used to crack it.

0..10% for cracking the substitution ciphertext and code/method used to crack it.

0..+5 bonus for cracking the One Time Pad ciphertext and code/method to crack

0..+5 bonus for cracking the Vigenere ciphertext (must have code/method to crack)

0..+5 bonus for cracking the stream ciphertext (must have code/method to crack)

Task 2: One Time Pad Cipher 0..20%

Task 3: Stream Cipher 0..50%

Dr. Scott will use the Git Logs to ascertain the activity of both team members so ensure you **BOTH** commit and push often, as this will be an indication you are pulling your weight.

Submission Via GitHub Classroom and ~~Handin~~ Canvas as a Backup:

When you begin your assignment you and your partner will use the below GitHub Classroom link to create or join a repository. Make sure you do not both create a repository by seeing if the repository has already been created. The handout **project1_pairs.docx** on canvas contains the prescribed team names you must use.

https://classroom.github.com/a/iylzZeOc

See appendix A at the end of the document for guidance on using GitHub classroom.

When ready to submit create a branch called **submission** it is that branch I will grade. In the absence of a submission branch I will reduce your grade by 5 points and look in **main/master**.

Your repository should be clearly organize, containing your work for each task in its own folder. A copy of this word document with your answers to task 1 typed in red.

As a backup (because GitHub Classroom is new to us), also submit your project directory as a gzipped tar file via handin on agora. The tar file will contain your README file, your tests files (plain text, cipher and keys), and your source files. If your project is in a directory named project1 and you are in its parent directory, then you can use:

```
tar -cvzf project1.tar.gz project1
```

to create the file project1.tar.gz. The 'c' means to create a tar file, the 'v' means verbose, the 'z' means to turn the tar file into a gzipped file, and the 'f' specifies that the next command line argument is the name of the resulting file. Then use the following command to submit the file.

```
handin.466.1 1 project1.tar.gz
```

Not having an accessible repository, and having to use an Agora submission will reduce your project grade by 10 points.

Project 1 is due Wednesday October 21st at 11:59pm via Handin and GitHub classroom

Also, with your hand in submission you are to submit a two to six (ish) sentence confidential attestation statement outlining your and your partner's contribution to the project and whether you though the workload was acceptably evenly balanced between the two of you.

Late Policy:

You can submit the project late for 5% of the maximum possible grade each day upto a maximum of 25%, after which it is not taken. This means if it comes in on October 3rd at 12:01pm it will be considered one day late and lose 5%, and for every 24 hours past that point an additional 5% will be accrued up to a maximum of. Saturday and Sunday together count as a single day.

Appendix A: Using GIT Hub:

Set Up

For this project you will be working with **GitHub.** On Canvas you will see a link to the GitHub Project Repository Invitation which I have also copied below:

https://classroom.github.com/a/iylzZeOc

You will then be expected to sign in to the WCU organization.

The following describes the steps you will go through when setting up your repository and Project once you have signed in:

1) Once you have signed up you will be prompted to link an existing account or create a new one. When creating or linking an account, ensure that the name used for your account is representative of your name.

2) So see who you are working with and what you should call your project/team, see handout **Project1_Pairs.docx** on our course's Canvas account. You will notice that have prescribed a team name that you and your partner **must** use when creating a team.

3) Next you and your partner will first need to create or join your project/team repository. To do that use the link below, choose your roster name (your first and last name), and then select to create a team, or join the one named in the handout if it exists.
https://classroom.github.com/a/iylzZeOc

4) Next you and your partner must create or join the repository to be used for your project.

5) At this point Your repository should be a fork of my template, and contain a README and a folder with several example crypto keys. GitHub should now allow you to clone this repository to your local machine, and/or Agora with authentication via your PAT, or SSH if you have set that up with GitHub.

Note, that GitHub classroom uses Personal Access Tokens instead of passwords when working with your repositories as a remote. Other than this the use of a Git Hub repository within Android Studio is the same. If you experience difficulty with this set up do reach out to