Cryptographic Methods for Deciphering/Identifying Ciphers in MATLAB

[1]Christian Pinto, BTech, [2]Harrison Carranza, MIS
[1]*The New York City College of Technology – CUNY, USA, christian.pinto@mail.citytech.cuny.edu*
[2]*Marist College, USA, harrison.carranza2@marist.edu*

Mentor: Aparicio Carranza, PhD
The New York City College of Technology – CUNY, USA, acarranza@citytech.cuny.edu

Abstract* – *Cryptographic methods are used worldwide to keep information safe. Monoalphabetic, Polyalphabetic or Polygraphic substitutions, etc., are methods that can make the information secure. Once encrypted, the receiver should be able to understand the message with a given key. If there is no key, the formula called Index of Coincidence (IC) can help deciphering. It narrows down the search for the method used with the result obtained from the IC formula. Using the MATLAB IDE, we will implement the ciphers mentioned above and along with the IC formula, determine the cipher if no key is present.

Keywords – *Cipher, Decryption, Encryption, Monoalphabetic, Polyalphabetic, Polygraphic*

I. Introduction

Sending important information can be risky for many different reasons. Now a days, communications can be intercepted and if not secure, the interceptor can see everything you sent. This is where cryptology comes in. Cryptology is the study of codes and ciphers. It allows us to hide a message and send it without worries. As defined by David Kahn, "Cryptology is protection, it is to that extension of modern man – communications - what the carapace is to the turtle, ink to the squid, camouflage to the chameleon" [1]. When cryptography first started, there were only simple ciphers but no one understood what the messages meant because it was new [2]. Now cryptography has gotten so complex that even if someone were to get a hold of the message, they will not necessarily understand it. When talking about cryptography, it is important to understand how it works. Cryptography has two parts: *ciphering/encryption* and *deciphering/decryption*. Encryption is taking a message and disguising the message with any cryptographic method. Decryption is the opposite of encryption where you go from the hidden message to the original message. Each method used has a different level of security.

Monoalphabetic, Polyalphabetic, Polygraphic Ciphers, and public key ciphers are the main groups of ciphers. In this paper we will be focusing mainly on Monoalphabetic, Polyalphabetic, and Polygraphic Ciphers. Within these general forms, there are different methods within each one. Monoalphabetic contains the Additive Cipher, Multiplicative Cipher, and Affine Cipher. The polygraphic cipher contains the Vigenere Square and the Polyalphabetic contains the Hill cipher. With the knowledge of cryptography, we have created a program that will decipher messages as long as the user provides the key. Another part of the program can identify the cipher of a message. This part is still a work in progress. In order to understand how the program works, understanding how deciphering messages is the first step.

II. Deciphering Ciphertext

When you decipher messages, you will need to have the key used to encrypt them. This is important because without it you cannot get the original message sent. Since there are multiple methods, we will take it one by one starting with the Caesar (additive) cipher.

A. Additive/Caesar Cipher

In order to decrypt messages, you will take the additive inverse with mod(26). Let us take the example, "**kl pb qdph lv dolfh**" and using the key of 19 to decipher the message. First we need to get the additive inverse of 19 which is -19 mod 26. The result from this is **7** mod **26**, which we will use to decrypt. With this key, we add it to the Ciphertext position and come back to the plaintext message.

Table 1 - Inverse Additive Cipher

Ciphertext	A	b	f	r
Position	1	2	6	18
Plaintext	H	i	m	y
Position	8	9	13	25

If done mathematically:

$$1 + 7(\text{mod } 26) = 8\text{mod}(26) = H$$
$$2 + 7(\text{mod } 26) = 9\text{mod}(26) = i$$
$$6 + 7(\text{mod } 26) = 13\text{mod}(26) = m$$
$$18 + 7(\text{mod } 26) = 25\text{mod}(26) = y$$

When we decipher the entire message we end up with "**Hi my name is Alice**". For this reason, the additive cipher is the least secure cipher. Someone can find the key used sooner or later.

B. Multiplicative Cipher

The multiplicative cipher works in the same manner. Once you have the key, take the multiplicative inverse and

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2016.1.2.069
ISBN: 978-0-98228996-9-3
ISSN: 2414-6390

you end up with the plaintext message. Once you have the key you would multiply the cipher text by the inverse key $k^{-1} mod(26)$. The inverse would be a number that makes this statement true

$$A*K^{-1} = 1 \bmod (26) \quad (1)$$

The easy way to decipher the message with the key would be to look up the table with all the inverses. In the additive decipher, we used the key 11 to encipher the message. Now let us use that key to decipher the message. According to Table 1, the inverse of 11 is 19 [3].

Table 1 - Multiplicative inverse chart

Number	1	3	5	7	9	11	15	17	19	21	23	25
Multiplicative inverse	1	9	21	15	3	19	7	23	11	5	17	25

Now let us decipher the message "**Ju mo xkmc ua kbugc**" with the inverse key 19. To decrypt the cipher text we multiply each cipher text by the multiplicative inverse and use mod (26).

C1 = 10 * 19mod(26) = 190 mod(26) = 8 = h
C2 = 21 * 19mod(26) = 399 mod(26) = 9 = i
C3 = 13 * 19mod(26) = 247 mod(26) = 13 = m
C4 = 15 * 19mod(26) = 285 mod(26) = 25 = y

Table 2- Multiplicative inverse applied

Ciphertext	j	u	m	o
Position	10	21	13	15
Plaintext	H	i	m	Y
Position	8	9	13	25

As you go on, you will end up back at the message "**Hi my name is Alice**". Just like with the additive cipher, the person can brute force and find the multiplicative inverse. They will eventually find the right key and decipher the message. The next cipher will be the Affine cipher which is a bit more secure.

C. Affine Cipher

In order to decipher this message, you will need the multiplicative key and the additive key. The first step is to take multiplicative inverse of the key and apply it to the ciphertext. Once that is done, you will need to apply the additive inverse to the message to get the plaintext [2]. Let us take the example, "**Hello world**". If you encrypt it using the additive key of 2 and multiplicative key of 3, we will end up with the message:

After additive key applied
"**jgnnq yqtnf**"
After multiplicative key
"**duppy wyhpr**"

Now that we know the ciphertext, we can apply the inverse to get back to the original formula. We will do so by first applying the multiplicative key. Since we are using the multiplicative key of 3 we know that the inverse is 9 from Table 1.

Table 3 - Multiplicative Inverse Key Applied

Cipherttext	d	u	p	P
Position	4	21	16	16
Ciphertext (Multiplicative inverse)	j	g	n	n
Position	10	7	14	14

As observed in Table 3, we are getting back to the additive ciphertext when its all done. Once finished, you end up with, "**jgnnq yqtnf**" as it is expected. Now, if we apply the additive inverse of 2, which is 24, we get the following:

Table 4 - Additive Inverse Key applied

Ciphertext (Multiplticative inverse)	J	g	n	n
Position	10	7	14	14
Plaintext	h	e	l	l
Position	8	5	12	12

Table 4 is the result when finished, we get the original message "**Hello world**". We can also use the following formula to decrypt the message where **s** is the multiplicative key and **r** is the additive key.

$$p = r^{-1} + (s^{-1} * c) \ (mod\ 26) \quad (2)$$

If we apply the key from the example, we get the following:

C1 = 24+(9* 4)(mod26) = 24+(36)mod(26) = 60mod(26) = 8 = h
C2 =24+(9*21)(mod26)= 24+(189)mod(26) = 213mod(26) = 5 = e
C3 = 24+(9*16)(mod26) = 24+(144)mod(26) = 168mod(26) = 12 = k

This cipher is a bit more secure because if someone tries to brute force this with an additive decipher/multiplicative decipher, they will get nothing understandable. They would have to do each possible combination, which would take a long time. This cipher is not completely safe since it is a monoalphabetic cipher, then, let us take a look at the polyalphabetic cipher Vigenere square.

D. Vigenere Square

When you encrypt the message you will use the table seen in Figure 1. You use the top row for the plaintext and the left column for the keyword. Where the two intersect is your ciphertext. Deciphering works in a different manner. You will put your keyword on the top row and scroll down until you reach the letter of the Ciphertext and the letter on the left column is your plaintext. For example, if we decrypt the message, "**pqvc vw fcgb rmw**" and use the keyword "**city**", we will get the result seen in Figure 1.

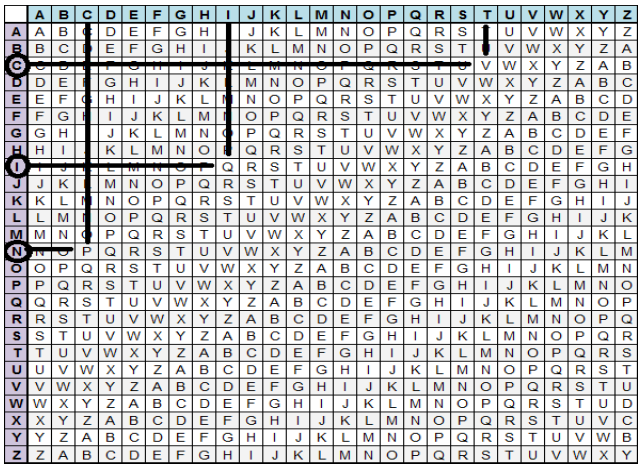

Figure 1: Deciphering Vigenere square

Table 5 - Deciphering of the Vigenere Square

Ciphertext	p	q	V
Keyword	c	I	t
Plaintext	n	i	c

When you continue all the way you will end up with the message "**Nice to meet you**". This cipher is pretty secure because without the key word, you will get nowhere. There are ways to determine the length of the keyword but it will not assist much because of the many different combinations of letters that you can put together. The next cipher we will look at is Hill's System.

E. Hills system

This cipher is typically one of the longer ones to decipher because you can only do two letters at a time and each time perform a matrix multiplication. When we encrypted, we use a 2 x 2 matrix. We will do the same when we decipher. The first step is to get the inverse of the 2 x 2 matrix. Next, you will multiply that matrix to the inverse of ad – bc (*determinant*). This will give us the inverse matrix that we will use to multiply later on.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \begin{matrix} d & -b \\ -c & a \end{matrix}$$

Determinant = ad – (-b * -c)

First find the modular inverse of the determinant. Similar to finding the multiplicative inverse.

$$\left(\begin{matrix} d & -b \\ -c & a \end{matrix}\right) * \text{modular inverse of determinant}) \bmod 26$$

$$= \text{Inverse matrix } \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1}$$

Now that we have the inverse matrix, the formula to compute the plaintext is the following where **p** is the plaintext and **c** is the ciphertext:

$$\begin{matrix} pi \\ pi+1 \end{matrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} * \begin{matrix} ci \\ ci+1 \end{matrix} \quad (3)$$

$$\bmod(26)$$

Let's use an example encrypted using hills system. The encrypted message is "**fcuxfwgvuiyvob**" and the plaintext is "**Nice to meet you**". When encrypted, we used the key a = 9, b = 4, c = 5, and d = 7.

The first step is to apply the inverse function to the matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \begin{matrix} d & -b \\ -c & a \end{matrix}$$

$$\begin{pmatrix} 9 & 4 \\ 5 & 7 \end{pmatrix}^{-1} = \begin{matrix} 7 & -4 \\ -5 & 9 \end{matrix}$$

Now we compute the determinant:

Ad – bc = (9*7) – (-4*-5) = 63 – 20 = 43

To find the modular inverse you can use an online calculator to make the process faster. For this example, the inverse of 43 is 23. Now, we multiply the inverse of the determinant with the inverse matrix.

$$\begin{matrix} 7 & -4 \\ -5 & 9 \end{matrix} * 23 \bmod(26) = \begin{matrix} 5 & 12 \\ 15 & 25 \end{matrix}$$

Now that we have our inverse matrix, we can apply it to the ciphertext "**fcuxfwgvuiyvob**" and get the plaintext.

$$\begin{matrix} p1 \\ p2 \end{matrix} = \begin{matrix} 5 & 12 \\ 15 & 25 \end{matrix} * \begin{matrix} 6 \\ 3 \end{matrix} = \begin{matrix} 14 \\ 9 \end{matrix} = \begin{matrix} n \\ i \end{matrix}$$

$$\begin{matrix} p3 \\ p4 \end{matrix} = \begin{matrix} 5 & 12 \\ 15 & 25 \end{matrix} * \begin{matrix} 21 \\ 24 \end{matrix} = \begin{matrix} 3 \\ 5 \end{matrix} = \begin{matrix} c \\ e \end{matrix}$$

When we continue all the way, we will get "**Nice to meet you**". When deciphered, there will be an extra letter at the end because the message was odd numbered. We need an even number in order to do the matrix multiplication. The extra letter was taken out so the message made sense. This is another secure cipher because of the key. It is a bit more limited than the Vigenere cipher because the key has to be prime to the modulus. The last two methods are much safer than monoalphabetic ciphers. Now that we know how to decrypt a messages, let us see the steps to analyze a message without any knowledge of the key which is called cryptanalysis.

III. CRYPTANALYSIS

Cryptanalysis is when we take an encrypted message and analyze it to see any patterns. This is what people do when they intercept a message. This is used not only in a bad way. It can also help us to find flaws in an encryption method and obtain new ways to make things more secure. The way to analyze a message is by using a frequency table. A frequency table consists of every occurrence of a letter in a message. By knowing the most frequent used letter in the ciphertext, you can compare it with the most used letter in English. In Figure 2, we can see the frequency of the most used English letters.

Letter frequency distribution for a sample English text					
letter	number of occurrences	frequency	letter	number of occurrences	frequency
E	8,915	.127	Y	1,891	.027
T	6,828	.097	U	1,684	.024
I	5,260	.075	M	1,675	.024
A	5,161	.073	F	1,488	.021
O	4,814	.068	B	1,173	.017
N	4,774	.067	G	1,113	.016
S	4,700	.067	W	914	.013
R	4,517	.064	V	597	.008
H	3,452	.049	K	548	.008
C	3,188	.045	X	330	.005
L	2,810	.040	Q	132	.002
D	2,161	.031	Z	65	.001
P	2,082	.030	J	56	.001

Figure 2 - Most frequently used English letters

As we can see, the letter "**e**" is the most used letter in English. Knowing the most used letter in a Ciphertext, you can map that letter with the letter "**e**". This would typically work well with monoalphabetic ciphers because there can only be one letter that corresponds to another letter. There are also charts for the frequency of digraphs and trigraphs as seen in Figure 3.

digraph	relative freq. (%)	digraph	relative freq. (%)
TH	3.319	ES	1.213
HE	2.859	TO	1.213
IN	2.081	NT	1.200
ER	1.596	EA	1.059
ED	1.493	OU	1.047
AN	1.430	NG	1.034
ND	1.430	ST	1.034
AR	1.302	AS	0.996
RE	1.302	RO	0.996
EN	1.289	AT	0.983

trigraph	relative freq. (%)	trigraph	relative freq. (%)
THE	1.82	EAR	0.26
AND	0.77	HAT	0.24
ING	0.68	OFT	0.22
HER	0.50	WAS	0.21
NTH	0.40	EST	0.21
ENT	0.36	HEN	0.20
THA	0.35	IVE	0.20
INT	0.30	ALL	0.20
ERE	0.29	THI	0.20
DTH	0.28	HIN	0.20

Figure 3 - Frequency table for digraphs and trigraphs

These tables are mainly used for higher level ciphers such as Hills system or the Vigenere cipher. The question is what do we do with this information?. There is a formula called the Index of Coincidence (**IC**) that can possibly determine whether or not the cipher is monoalphabetic or not. The Index of Coincidence formula is the following:

$$\sum_{i=1}^{26} \frac{ni(ni-1)}{n(n-1)} \qquad (4)$$

In this formula, n_i is the corresponding letter in the alphabet. For example, N_1 = "A", N_2 = "B" and so on. N is equal to the length of the message. Then we take the sum and get a result depending on the type of cipher that has been used. When we get the results there are two options. If the result is close to .065, then the message has probably been encrypted with a monoalphabetic cipher. If the result is less than .065, it has probably been encrypted with a higher level cipher. Although this result is not all so certain, it helps narrow down your search. For example, let us take the message "**Welcome to the poster session**". If we use a basic additive cipher and shift it by 6. We will get the following message, "**ckriuskzuznkvuyzkxykyyout**". If we create a frequency table we will get the following:

Table 11: Frequency Table for message

Letter	c	i	k	n	o	r	s	t	u	v
Frequency	1	1	5	1	1	1	1	1	4	1

Table 12: Frequency table continued

Letter	x	y	z
Frequency	1	4	3

From this table we can see that k is the most frequent letter. When we use the IC formula the result is: 0.0833330. Since it is higher than 0.065, it would mean it is a monoalphabetic cipher which is correct. Since k is the most frequent letter in the message, we can assume it is the letter **e** in the plaintext. We know that the position of **e** is 5 and k is 11. In order to get from **e** to **k** we must shift by 6. Since, we are decrypting the message we would apply the inverse additive key which is -6 and we end up with the message, "**Welcome to the poster session**". If it did not work, we would have to map **k** with the next most frequent English letter which would be "**t**" and use the same method or it may not be an additive cipher.

Now if we end up getting the result that is less than .065, we will need to determine the length of the keyword/key. There is a test called the Kasiski Test that can help figure out the length for the Vigenere Square which states, "if a string of characters appears repeatedly in a polyalphabetic ciphertext message, it is possible (though not certain), that the distance between the occurrences is a multiple of the length of the keyword" [2].

For this test, you will need to keep a chart with the repeated string, the position of the first letter in the string, the distance of each of the occurring stings, and the prime factorization of the distance. With the third column, you can make an assumption that it is a multiple of the keyword [2]. An example of this is shown in Figure 4.

Kasiski Test Results

Sequence	Frequency	Distances	Prime Factors
GBANY	2	36	2, 2, 3, 3
KZNAL	2	90	2, 3, 3, 5
BANY	2	36	2, 2, 3, 3
CEFE	2	135	3, 3, 3, 5
GBAN	2	36	2, 2, 3, 3
JBMI	2	522	2, 3, 3, 29

Figure 4 - Chart for the Kasiski Test

With this test, it can help us to come to a conclusion on the length. We can also use the following formula to calculate the length of the keyword which is:

$$r \approx \frac{0.027n}{(n-1)IC - 0.038n + 0.065} \qquad (5)$$

In the above formula, n is the variable for the length of the message. If we apply this formula and say the Kasiski test led you to the assumption that the multiple of the keyword is 7 and you get 15.7 from the result, most likely the keyword length is 14.

This same exact cryptanalysis and decryption process is what we have implemented. The cryptanalysis portion is still being developed.

IV. Implementation

For the implementation of our cryptographic solution in MATLAB, we started with the monoalphabetic ciphers. Before starting with our MATLAB implementation, we needed to learn about its some important features. When we assign to a variable a character, it will return you its ASCII code.

```
A(1) = 'e';
disp(A(1));
    101
```

As it is seen above, the letter “e” corresponds to the decimal number 101 on the ASCII table which can be seen in the ASCII table in Figure 5.

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]

Figure 5 - Complete ASCII table

We first began with the additive cipher and worked our way up to the Hill’s cipher which was last. We will start off with the first part of the program which is when we know the key.

Part 1

Additive Cipher

This cipher is fairly straightforward as you know from the previous sections. The first step is to get the user message as a string. The next step is to get the key from the user. Once we have the key, we will need to take its additive inverse and apply it to the message which is what we did.

IM = char(rem(IM-i-96+26,26)+96);

In this line, IM stands for the input message, rem stands for remainder which is similar to mod, and char converts a number to a letter based on their ACII code. How that line works is it takes the input message where each character is converted to decimal, and then subtract the key and subtract 96. We will then add 26 to it to be able to get the mod 26 afterwards with the rem function. Once we get the remainder, add 96 back to the remainder so we can return to the original position on the ASCII table. Once done, we convert it back into characters using char [4].

Multiplicative Cipher

We apply the same basic idea with the multiplicative cipher. We take the user's input and then get its inverse key which we will then use it to multiply with the input message and get its corresponding letter. To get the multiplicative inverse or modular inverse, we used the following code:

```
[d,a,b] = gcd(k,26);
in = mod(a,26);
g = gcd(in,26);
```

With this result, we can calculate the inverse of the multiplicative key. Once we get that value, we test it if it is prime to the modulus which in this case will be 26. If the value does end up being 1, we will use the same method that we used for the additive cipher. We need to take the input message and subtract it by 96 then multiply it by the inverse key. Once we get that result, we use the mod function to calculate the modulus of the value and then add 96. Once we get the ASCII value we use the char command to convert it to letters.

IM = char(mod((IM – 96)* in, 26) + 96);

Affine Cipher

The first step is to get the multiplicative key and the additive key from the user. Using the key, we implement what we did in the previous two other programs. As we know from decrypting, you apply the multiplicative key then the additive key. We apply the following code to find the multiplicative inverse:

```
[d,a,b] = gcd(k,26);
in = mod(a,26);
```

Then we check if the gcd of the inverse and the modular is 1

```
g = gcd(in,26);
```

If the gcd is 1, then we will be able to decipher the message. We put together multiplicative cipher then the additive cipher code.

```
IM = char(mod((IM – 96)* in, 26) + 96);
IM = char(rem(IM-i-96+26,26)+96);
```

Vigenere Square

The first step is to take in the user's keyword as a string. The second step is to convert the strings from its ASCII code to the position 1 – 26.

```
Key = key – 96;
IM = IM – 96;
```

The next step is to get the key index. This is done by creating an array from 0 to the length of the input message and subtract the length by one. Then you will take the length of the keyword as the modulus. You will add one to that result so it is between one and length of the keyword [5].

```
Keyindex = mod(0: (length(IM) – 1) , length(key)) +1;
```

Once we have the key index, we will use it to identify the position of the keyword which will repeat throughout the array [5].

```
K = key(keyword);
Disp(k);
```

If we display the result we will get the following:

```
Input Message: pqvcvwfcgbrmw
input key: city
  Columns 1 through 12

    3     9    20    25     3     9    20    25     3     9    20    25

  Column 13

    3
```

Figure 6 - ACII Position for key

As you can see after every four slots it will repeat itself. This is what we do when we decipher. We repeat the keyword each time we reach the end of the keyword. Now once we have this, the next step is to subtract the input message from the keyword or "k" in this case. We will subtract the keyword position from the input position.

```
Plaintext = IM – k;
```

Now if the result is less than 0, you will need to add 26 to it. This will let it return to its position. From time to time, the result will be less than 0. To correct the positions we do the following:

```
index = plaintext < 0;
plaintext(index) = plaintext(index) + 26;
```

The index line lets the program find the position of where the plaintext is less than 0. The following line adds 26 to the corresponding position. This corrects the position. The last step in this program is to convert it from Ciphertext to plaintext by using the char command after adding 96 to the Ciphertext.

```
Plaintext = char(plaintext + 96);
```

Hills System

The first step is to analyze the input to see if the length of the message is even. If it is not just add an extra letter in the end. Now once we have an even input, the next step is to change the array into a 2 x (length of message divided by 2) matrix. This is done by the reshape command in MATLAB.

```
IM = reshape(IM,2,Size/2);
```

This is needed so you can perform the matrix multiplication. You cannot multiple a 2 x 2 with a 1 x (length of message). As usual, you will need to convert it from its ASCII code to 1 – 26. We need to get the key and put it in a 2 x 2 array. Once we have the matrix, we need to take the inverse.

```
Inm = [d –b;-c a];
```

Once done, the next step is to get the determinant of the matrix.

```
X = (a*d)-(b*c);
```

Next, we need to find the modular inverse of the determinant. Now if the inverse is prime to 26, then you will multiply the modular inverse with the inverse matrix. The second to final step is to multiply the inverse matrix with the input message array. This will give you the result but first you will need to reshape it again so when you display it, it will be correct and in order. We turn it back into a single row and to its original length with the following:

```
IM = reshape(y,1,Size);
```

Once that is done the final step is to convert it back with char. Now we will see how to implement the cryptanalysis.

Part 2

In this part of the program we will implement the IC formula and the frequency table to examine an input message. First you will get the users input. The next step is to create a table which holds all the occurrences of each letter. This can be done in different ways but we did the following:

```
A = zeros(1, 26);
for i = 1:Size
  for j = 1:26
    A(j);
  end
if(IM(i) == 'a')
 A(1) = A(1)+1;
```

End
(b – z)
end

For each iteration, the corresponding letter will increase by 1. Once we have this, we implement the IC formula to determine the whether it is monoalphabetic or another higher level cipher. This can be done by using the sum() function in MATLAB.

sum((A(k).*(A(k)-1))/(Size*(Size - 1)));

If the result states it's a monoalphabetic cipher, we will test each cipher. The same will happen with the higher level ciphers. For now we will only work with monoalphabetic ciphers. Now if the value is close to 0.065, we will run tests from the first monoalphabetic cipher which is the additive and then to the multiplicative. Now we will pass the values of the matrix A, the input message (IM) and the Size of the message. Now we will take the array A and determine which value is the most frequent. Once we have the index value, we will subtract e (5) – I (index of most frequent letter value (1-26)). This will give us the key if it was an additive cipher. We would apply mod(26) just in case it wasn't from 1 – 26. Now we will apply this key, r:

IM = char(mod(IM + r - 96,26)+96);

To check the output, we use a function called spellcheck which will check the spelling. This is as far as we got implementing the cryptanalysis. The following will be the results of the program.

V. Results

As mentioned before, this program has two parts. The first part is where the user enters the key and the other is where the user just inputs a message

Part 1

```
Do you know the key for cipher?(Y/N): y
Choose Cipher
1.Monoalphabetic
2.Polyalphabetic
3.Polygraphic
```

Figure 7 - Main menu decryption with key

Once here, you can choose which type of cipher you would like to decipher. We will go through each one and see if the result is correct. We will start with the monoalphabetic ciphers first.

```
Select Monoalphabetic Cipher
1.Additive(Shift)
2.Multiplicative
3.Affine
```

Figure 8 - Monoalphabetic Menu

If the user inputs 1, it will take them to the Monoalphabetic menu. It will prompt them for the message and then the key. Let's begin with the additive cipher.

Additive/ Caeser Cipher

We will use the Ciphertext from the example we used before "**Ab fr gtfx bl Tebvx**" with the key of 19.

```
Input Message: abfrgtfxbltebvx
Enter Key: 19
himynameisalice
```

Figure 9 - Output from Additive Program

Although the letters aren't separated, you can still see the original message.

Multiplicative Cipher

Let's take the example, "Ju mo xkmc ua kbugc", with the multiplicative key of 11:

```
Input Message: jumoxkmcuakbugc
Enter Key: 11
himynameisalice
```

Figure 10 - Multiplicative Ciphertext Deciphered

Affine Cipher

This part takes in the inputs for the multiplicative key and additive key. We will use the example "gqkthmr yc dmffmz" with the multiplicative key of 3 and additive key of 8.

```
Input Message: gqkthmrycdmffmz
Enter Multiplicative Key: 3
Enter Additive Key: 8
complexisbetter
```

Figure 11 - Affine Deciphered

For this cipher we used a different message which was "Complex is better". For the next section, the user would need to input 2 in the main menu to get to the Vigenere Square.

Vigenere Square

We will use the Ciphertext message, "pqvc vwfc gbrm w" and the keyword "city" to decipher the message.

```
This is the Vigenere Decipher
Input Message: pqvcvwfcgbrmw
input key: city
     3     9    20    25

nicetomeetyou
```
Figure 12 - Vigenere Square Program

The outputs you see displayed on the screen are the position of the keyword letters. This time our message was “Nice to meet you”.

Hills System

Once the user inputs the message, it will determine if it is even or odd. If it is even it will decipher, else you will need to add an extra letter. Let’s take the message “fcuxfwgvuiyvob” with the key a = 9, b = 4, c = 5, d = 7. For this situation, we removed letter at the end to see what happened if it was odd.

```
Hills System Decipher
Input Message: fcuxfwgvuiyvo
The length of the message is:   13
Enter another letter to make it even
```
Figure 13 - Situation where message is odd

Now when I put back the last letter and the same key this is the result I get.

```
Hills System Decipher
Input Message: fcuxfwgvuiyvob
The length of the message is:   14
Input a: 9
Input b: 4
Input c: 5
Input d: 7
     9     4
     5     7

nicetomeetyouo
```
Figure 14 - Even input and output

As you can see the program gave me the correct output “**Nice to meet you**” even though it has the extra letter at the end. When situations like these happen, you can ignore the last letter.

Part 2

In the main menu, if you entered “N” it will perform the cryptanalysis. The example we will use is a monoalphabetic cipher. We took the message, “Welcome to the poster session” and used the additive key of 6 to encrypt it. Our output is, “**ckriuskzuznkvuyzkxykyyout**”. We will use this example for our input message in our program.

The first cipher it will test is the additive cipher. If we input the cipher from above, our output will be the following:

```
Do you know the key for cipher?(Y/N): n
Input Message: ckriuskzuznkvuyzkxykyyout
The length of the message is: 25  Columns 1 through 12

     0     0     1     0     0     0     0     0     1     0     5     0

  Columns 13 through 24

     0     1     1     0     0     1     1     1     4     1     0     1

  Columns 25 through 26

     4     3

Index Of Coincidence is: 0.0833330.5
The message was encrypted with a monoalphabetic substitution
Spellcheck output is:  1
The sender used the Additive Cipher
```

Figure 15 - Frequency Table/ IC formula result

For the spellcheck function, 1 means it was spelled correctly and 0 means it’s not. Now if it was a messaged encrypted with a multiplicative, the first result would be wrong and move on to the next cipher. To show this, let’s take the example, “***Welcome to the poster session***”, and apply a multiplicative key of 3 to the message, “**qojismohshxovsehobeoeeasp**”. If we input this to the program, we won’t get an output, but we can see it move on to the next cipher. This is because we haven’t fully completed this part of the program.

```
Do you know the key for cipher?(Y/N): n
Input Message: qojismohshxovsehobeoeeasp
The length of the message is: 25  Columns 1 through 12

     1     1     0     0     4     0     0     3     1     1     0     0

  Columns 13 through 24

     1     0     5     1     1     0     4     0     0     1     0     1

  Columns 25 through 26

     0     0

Index Of Coincidence is: 0.0833330.5
The message was encrypted with a monoalphabetic substitution
Spellcheck output is:
It wasnt an additive cipher
Multiplicative test
```

Figure 16 - Output from multiplicative program

Since it hasn’t been completed it can’t determine if a multiplicative cipher was used. This is what we plan to accomplish in the future.

VI. Future Research

We were able to implement deciphering messages in MATLAB. As of right now we have the additive cipher implemented but we would like to continue implementing cryptanalysis to determine the rest of the ciphers. A conflict with this is when you get to the higher level ciphers where the key could be anything like the Vigenere Square and Hills Cipher.

VII. Conclusion

We successfully implemented the decrypting methods: Additive, Multiplicative, Affine, Vigenere Square, and Hills system using MATLAB. If you input the correct key, it will

display the correct output. With our knowledge, we can continue going further then just decrypting with a key.

REFERENCES

[1] R. Spillman, "Introduction to cryptology," in *Classical and ContemporaryCryptology*, NJ, Pearson,2004,ch 1. sec 1.0 – 1.9, pp.1 - 12.

[2] R. Lewand, "Preface,"in *Cryptological Mathematics*, Washington D.C, MAA, 200, ch 0, pp. xi.

[3] C. Christensen,"Cryptography of Multiplicative Ciphers", NKU, Highland Heights, KY, 2006.

[4] H. Boas.(2005,April 5).*Introduction to Matlab*[Online]. Avaliable: http://www.math.tamu.edu/~boas/courses/math696/matlab-introduction.html.

[5] T. Booher.(2014, September 13).*Cryptography*[Online].Available: http://www.theboohers.org/cryptography/

[6] V.Neale.(2012 September).*An Introduction to Modular Arithmetic*[Online]. Available: http://nrich.maths.org/4350

[7] D. Arnold.(1996, September 7).*Deciphering Hill Ciphers* [Online. Available: http://msemac.redwoods.edu/~darnold/math45/Activities/HillCiphers/decipher.pdf

[8] Matlab Programming Fundamentals, MathWorks, [Online] September 2015, https://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf (Accessed: 5 October 2015)

[9] G.Kessler.(2015 Decemeber 5).*An Overview of Cryptography*[Online]. Available: http://www.garykessler.net/library/crypto.html

[10] Index of Coincidence, Thonky, [Online] 2015,Available: http://www.thonky.com/kryptos/index-of-coincidence (Accessed: 4 November 2015).

[11] Everything You Need to Know About Modular Arithmetic, Cornell, [Online] 2006, http://www.math.cornell.edu/~morris/135/mod.pdf (Accessed: 12 December 2015.