

Using MATLAB as a tool for encryption and decryption

Authors

Andrew Dittmer

Elaine Freed

John Swanson

Raafay Uqaily

Submitted To

Mr. Michael Brewster

Engineering 102 - H04

Freshman Engineering

Statler College of Engineering and Mineral Resources

West Virginia University

Morgantown, WV

Sunday, 26th April, 2020

Signatures: By signing this, we agree that we have not received any unauthorized assistance on any aspect of this project. We have each read and edited this report.

Andrew Dittmer: Was a part of writing the following sections (Abstract, 1.2, 2.3, 3.1, 3.2, 3.3, 5) Andrew D.

Elaine Freed: Was a part of writing the following sections (1.2, 2.1, 2.2, 2.3, 3.1, 3.2) ELAINE FREED

John Swanson: Was a part of writing the following sections (1.1, 1.2, 2.3, 3.1, 3.2) John Swanson

Raafay Uqaily: Was a part of writing the following sections (1.1, 1.2, 2.3, 3.1, 3.2, 3.3, 4, 5.1) Raafay Uqaily

Abstract

The goal of the project was to create five individual MATLAB script files that would be capable of successfully encoding and decoding a message using a unique type of cipher. Various conditional and branching techniques were incorporated in the creation of a tabula recta, determination of a key, and the general encryption and decryption process. Specifically, for-loops, while-loops, if-statements, and switch-cases were used to accomplish these goals. Each cipher generated its key in a unique way, resulting in a different encrypted message for each script file. However, each script successfully generated the same decrypted message when the same initial message and key was inputted by the user for both the encoding and decoding process. From the five ciphers, the Running-Key cipher was determined to be the safest for encryption, primarily due to its lengthy and unique keys, which were often titles or phrases from books and movies. The scripts accomplished the required tasks while incorporating a series of advanced MATLAB methods as part of the final algorithm, thus meeting the initial goal of providing secure end-to-end encryption to the user.

Table of Contents

Abstract.....	i
1 Introduction.....	1
1.1 Problem Statement.....	1
1.2 Background.....	2
1.2.1 History of Cryptography.....	2
1.2.2 Current uses of Cryptography.....	3
1.2.3 Research in Areas of Cryptography.....	3
1.2.4 Types of Encryption.....	4
1.2.5 Cryptanalysis.....	8
1.2.6 Future applications of Cryptography.....	9
1.2.7 Ethics of Encryption.....	10
2 Methods and Materials.....	12
2.1 Materials.....	12
2.2 General Coding Methods.....	12
2.2.1 Creating the Tabula Recta.....	13
2.2.2 Inputs and Formatting.....	13
2.2.3 Encryption.....	14
2.2.4 Decryption.....	15
2.3 Cipher-Specific Coding Methods.....	16
2.3.1 The Vigenere Cipher Key Generation.....	16
2.3.2 The Running-Key Cipher Key Generation.....	17

2.3.3	The Auto-Key Cipher Key Generation.....	18
2.3.4	Beaufort Cipher Encoding Process.....	19
2.3.5	Porta Cipher Encoding Process.....	20
3	Results.....	21
3.1	Encryption Results.....	21
3.1.1	Vigenere Cipher Encryption Output.....	22
3.1.2	Running-Key Cipher Encryption Output.....	22
3.1.3	Auto-Key Cipher Encryption Output.....	23
3.1.4	Beaufort Cipher Encryption Output.....	23
3.1.5	Porta Cipher Encryption Output.....	24
3.2	Decryption Results.....	24
3.2.1	Vigenere Cipher Decryption Output.....	25
3.2.2	Running-Key Cipher Decryption Output.....	25
3.2.3	Auto-Key Cipher Decryption Output.....	26
3.2.4	Beaufort Cipher Decryption Output.....	26
3.2.5	Porta Cipher Decryption Output.....	27
3.3	Condensed Output for all Ciphers.....	27
4	Discussion.....	29
5	Conclusion.....	30
5.1	Future Work.....	31
6	References.....	32

List of Figures

Figure 1: General Structure of a Secure Storage System (Agudo 192).....	4
Figure 2: The Tabula Recta (“Cryptography”).....	5
Figure 3: The Porta Cipher’s Edited Tabula Recta (Lyons).....	7
Figure 4: Beaufort Cipher Encoding Process.....	8
Figure 5: Code for the Tabula Recta.....	13
Figure 6: Removing Special Characters.....	14
Figure 7: General Encryption Process.....	15
Figure 8: General Decryption Process.....	16
Figure 9: Vigenere Cipher Key Editing Process.....	17
Figure 10: Running-Key Cipher Key Editing Process.....	18
Figure 11: Autokey Cipher Keyword Modification Process.....	18
Figure 12: Beaufort Cipher Message to Numbers.....	19
Figure 13: Beaufort Cipher Encryption Process.....	20
Figure 14: Code for Porta Cipher Encryption Table.....	21
Figure 15: Vigenere Cipher Encryption Output.....	22
Figure 16: Running-Key Cipher Attempt to Input Small Key.....	22
Figure 17: Auto-Key Cipher Encryption Output.....	23
Figure 18: Beaufort Cipher Encryption Output.....	24
Figure 19: Porta Cipher Encryption Output.....	24
Figure 20: Vigenere Cipher Decryption Output.....	25

Figure 21: Running-key Cipher Decryption Output.....	25
Figure 22: Auto-key cipher decryption output.....	26
Figure 23: Beaufort Cipher Decryption Output.....	26
Figure 24: Porta Cipher Decryption Output.....	27

List of Tables:

Table 1: Elimination of Special Characters.....	27
Table 2: Encryption Outputs.....	28
Table 3: Decryption Outputs.....	28

1 Introduction

MATLAB code was produced in order to effectively encrypt and decrypt a secret message using the Vigenère, Running-key, Auto-key, Beaufort, and Porta ciphers. This work was necessary as it helped secure cyberspace, which was declared as one of the fourteen grand challenges of the 21st century by the National Academy of Engineering. The project not only positively affected the individual using the code, but it also benefited the entire globe in terms of securing cyberspace.

1.1 Problem Statement

The problem faced in this project was to ensure that a user could safely transmit a secret message through cyberspace and whoever was on the receiving-end could successfully decrypt the message as well. The primary challenge of the project was to write code in MATLAB that could adequately encrypt and decrypt a message using various ciphers. Overarching issues included understanding how to determine the key for each of the particular ciphers.

There were several major objectives that were accomplished by conducting this project. The first objective was to conduct sufficient background research to better understand cryptography, cryptanalysis, and different encryption techniques. The next objective was to develop a MATLAB script that would compute an encrypted or decrypted message, depending on what the user wanted. Lastly, each cipher had to be analyzed to determine the most reliable encryption technique.

1.2 Background

1.2.1 History of Cryptography

Encryption was the name given to the process of translating a message into a script that was unintelligible to all except the people who were intended to receive it (Jackob). This name stemmed from the Greek word, *Kryptos*, which meant “hidden or secret” (Jackob). Ciphers were the set of steps performed to encrypt and decrypt these messages (“Cyphers”).

There were three general eras in the history of cryptography (“Ciphers”). The classical era included the oldest ciphers all the way until the 1950’s (“Ciphers”). Of the known accounts of ancient cryptography, the first was found in Egypt (Jackob). Around 1900 BC, a portion of altered hieroglyphics were used for inscription (Jackob). Different methods of encryption and decryption were used, including encryption tables, keys, shifts, and even a cipher disk, like the one invented in 1466 by Leon Battista Alberti (Jackob). Additionally, the Assyrians even applied encryption techniques to keep their processes for making pottery secret (Jackob).

The mechanical era dawned during World War II when people began using geared machines to encrypt messages (“Ciphers”). These machines had much more complex “keys” which had to do with the settings of the gears on the machines (“Ciphers”). A famous example from this era was the Enigma machine created by the Germans during the war (Jackob).

The modern era included all techniques after the mechanical era, which used current technology (“Ciphers”). Because computers made performing cyphers much quicker, the steps became increasingly complicated, to the point that they would be nearly impossible to do by hand (“Cyphers”). This, however, was vital to their security, as computers also made it very easy

to cryptanalyze (decryption when you don't know the key) the more simple ciphers of the classical era ("Cyphers").

1.2.2 Current uses of Cryptography

Cryptography can be applied in a variety of ways in society today. In fact, it is used in nearly every field of study. For example, one researcher from the Government College of Engineering in India explained how cryptography was applied to protect image-based secrets (Hasnet et al. 4491). This was applied within "video-conferencing, defense database, banking, finance, mobile computing, personal communication, and much more" (Hasnet et al. 4492). Furthermore, cryptography is also currently being used within cancer research. Obviously, as data is transferred at a global scale in order to gain more understanding of various cancers, it is imperative to keep confidential information protected. In order to keep information secure, cryptography can be applied within "national-level deduplication among state or province-based cancer registries, sharing of confidential data across cancer registries to support case aggregation across administrative geographies, secure data linkage, and cancer cluster investigation and surveillance" (Jacquez et al. 199). Additionally, cryptography even has applications within online purchases. Some IBM researchers alluded to this in a journal written regarding a specific cryptographic coprocessor: "Data encryption is a vital part of today's business processes and information systems to safeguard high-value business data, sensitive client information, online transactions, and electronic commerce" (Arnald et al. 1).

1.2.3 Research in Areas of Cryptography

Cloud computing and storage is widely used to run programs and save data using the internet and a third party (Agudo 190). It is very useful because it provides higher reliability,

easier access, and is cost effective, among other things (Agudo 190). However, there are some new concerns it raises, including the possibility of lost or altered data by unauthorized users (Agudo 190).

In order to use cryptography to secure data effectively, the data must be encrypted before it reaches the storage point, and the key must be saved elsewhere, as seen in Figure 1 (Agudo 191). This, however, raises questions about how to access the data without having to decrypt all of it, especially when you have multiple users (Agudo 191). If a user only wants to access a small part of the data for a short period of time, how can the user search for the data without telling the third party the key (Agudo 195)? Searchable encryption schemes allow a user to search for a word without the server knowing anything about the plaintext result or the word that was being searched (Agudo 195). There are several different types of searchable encryption techniques that have been developed as the advancements in computer technology have aided their creation (Agudo 195).

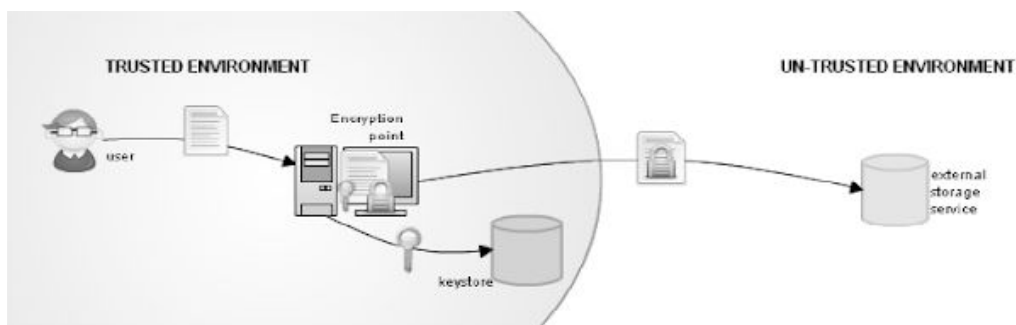


Figure 1: General Structure of a Secure Storage System (Agudo 192)

1.2.4 Types of Encryption

There are two types of basic encryption: asymmetric and symmetric. The main difference between both types is that symmetric encryption uses the same key for both encrypting and

decrypting, while asymmetric encryption uses two different keys, one for encryption and the other one for decryption. Asymmetric encryption often uses a “public key” and a “private key” which is kept secret by the person who originally performed the encryption (“What Types of Encryption Are There?”).

For basic symmetric encryption, there are five different ciphers that operate in similar ways but yield different results. The five types of ciphers commonly used in encryption are the Vigenère, Running-key, Auto-key, Beaufort, and the Porta. Each cipher uses a key and the tabula recta, which was used by Julius Caesar, to encode messages (Cruise 1). The tabula recta is a table that begins with each letter of the alphabet and then replaces that letter with the next letter of the alphabet in the row below. This process is repeated until the table arrives at the letter before the original, as seen in Figure 2.

	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	B
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	C
E	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
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	E
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	F
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	G
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	H
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	I
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	J
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	K
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	L
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	M
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	N
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	O
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	P
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	Q
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	R
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	S
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	T
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	U
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	V
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	V	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	X
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

Figure 2: The Tabula Recta (“Cryptography”)

Each type of cipher uses a similar process in which it begins with a key and a message that needs to be encoded. The idea is to find the first letter of the key in the first column and then

find the first letter of the message in the first row and then trace those two to their intersection point which yields the first letter of the now encrypted message (Lyons 2). For example, if the key was “KEYA” and the message was “BETA” we would find that “K” and “B” intersect at the letter “L”. This is the basic idea of all types of encryption, but the main difference between them is how the key is created.

The Vigenère cipher uses any word imaginable as the key and whenever the key is too short for the message, the key repeats. For example, if the key was “BETA” and the message being encrypted was “RETAINER” the key word would need to repeat twice in order to be the same length as the message, yielding “BETABETA”. The key can also be shortened if it is longer than the message using the same process.

The Running-key cipher is similar to the Vigenère cipher except the goal for this cipher is to ensure that the key is long enough for the particular message being encrypted. The key is often a passage from a book or a long line from a movie, which makes sure that it will be long enough for the message being encrypted (Lyons 1). For this reason, the Running-Key cipher is more secure than the Vigenère since the key does not repeat.

The Auto-key cipher has a short keyword as the key, like the Vigenère does, but when the key is too short for the message, instead of repeating itself, the remainder of the message being encrypted becomes the final portion of the key. For example, if the key was “BETA” and the message being encrypted was “RETAINER” instead of the key being “BETABETA”, like it was for the Vigenère, the key would become “BETARETA”. This makes the Auto-key cipher one of the easiest encryption methods to crack because only the first few letters of the message need to

be decrypted; after that, the remainder of the key is actually the original message that was intended to be hidden.

The Porta cipher uses the exact same process as the Vigenère cipher except that the first column is reduced to 13 options instead of 26, as seen in Figure 3. The process of encrypting is still the same, except now two different first key letters can yield the same result. For example, in the first column, “E” and “F” are now side by side so they no longer have their own unique rows as they did before because they now share a row. This narrows down the difficulty of cracking the code because there are 13 different key options instead of 26.

Keys	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	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
C,D	o	p	q	r	s	t	u	v	w	x	y	z	n	m	a	b	c	d	e	f	g	h	i	j	k	l
E,F	p	q	r	s	t	u	v	w	x	y	z	n	o	l	m	a	b	c	d	e	f	g	h	i	j	k
G,H	q	r	s	t	u	v	w	x	y	z	n	o	p	k	l	m	a	b	c	d	e	f	g	h	i	j
I,J	r	s	t	u	v	w	x	y	z	n	o	p	q	j	k	l	m	a	b	c	d	e	f	g	h	i
K,L	s	t	u	v	w	x	y	z	n	o	p	q	r	i	j	k	l	m	a	b	c	d	e	f	g	h
M,N	t	u	v	w	x	y	z	n	o	p	q	r	s	h	i	j	k	l	m	a	b	c	d	e	f	g
O,P	u	v	w	x	y	z	n	o	p	q	r	s	t	g	h	i	j	k	l	m	a	b	c	d	e	f
Q,R	v	w	x	y	z	n	o	p	q	r	s	t	u	f	g	h	i	j	k	l	m	a	b	c	d	e
S,T	w	x	y	z	n	o	p	q	r	s	t	u	v	e	f	g	h	i	j	k	l	m	a	b	c	d
U,V	x	y	z	n	o	p	q	r	s	t	u	v	w	d	e	f	g	h	i	j	k	l	m	a	b	c
W,X	y	z	n	o	p	q	r	s	t	u	v	w	x	c	d	e	f	g	h	i	j	k	l	m	a	b
Y,Z	z	n	o	p	q	r	s	t	u	v	w	x	y	b	c	d	e	f	g	h	i	j	k	l	m	a

Figure 3: The Porta Cipher’s Edited Tabula Recta (Lyons)

Unlike the other ciphers, the Beaufort cipher is different. The Beaufort cipher uses a short key and tabula recta table just like the Vigenère and the others, except the process of encoding is completely different. Instead of finding the first character of the key and the message in the first column and row, it finds the first character of the message being encoded in the first row and traces that down the column until it intersects the first letter of the key. Then, it traces that back to the first character of that row and that becomes the first character of the encoded message (Lyons 3). For example, if the key was “BETA” and the message being encoded was

“RETAINER” the key would repeat the same way the Vigenère did, yielding “BETABETA”.

Then, as seen in Figure 4, the code begins in the first row at the location of the first letter of the message “R” which is the box highlighted green. Finally, it travels down that column until it finds the first letter of the key which is “B” and highlighted red. From there, the first character of the encrypted message is located in the first column of that row, which is “K” and highlighted red. This process repeats for each character of the message that is being encoded.

	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
1	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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
10	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
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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
20	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
21	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
22	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
23	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
24	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
25	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
26	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

Figure 4: Beaufort Cipher Encoding Process

1.2.5 Cryptanalysis

Cryptanalysis encompasses analyzing and finding the solution of ciphers, placing an increased emphasis on understanding the method and system rather than the actual code (Gaines 1). The main goal of this type of study is to discover the hidden aspects even if the key or main algorithm is unable to be cracked (Martin 13). While the objective of cryptanalysis is to find shortcomings in encryption or otherwise cracking encoded messages, cryptanalysts' research results are used by cryptographers to improve and strengthen or replace flawed algorithms (Rouse).

When cryptanalyzing ciphers, several 'candidate' keys are tried until a key is found that produces a relatively readable output. This method is known as text characterisation and is a way to determine how close a piece of text is to the English language. Some of the common techniques of text characterization include the chi-square statistic, the index of coincidence, unicity distance, and frequency counts (“Crypto”).

There are many different types of cryptanalysis attacks and techniques, which vary depending on how much information the analyst has about the ciphertext being analyzed. Some common techniques include: plaintext, differential, integral, and side-channel attacks (Rouse). Although the goal is to find weaknesses in the algorithm itself, successful attacks on systems include attacks on the system administrators or users as well, where attackers gain access through subterfuge, susceptibility to greed, or through physical or threat of violence (Giovanni). Cryptanalysis requires a strong grasp of mathematics in terms of predicting the number of possible combinations, however, there are several tools that can be used for doing such analysis (Rouse). For example, CryptoBench is a program that can be used to do cryptanalysis of ciphertext generated with many common algorithms (Rouse).

1.2.6 Future applications of Cryptography

“In an attempt to predict what cryptology may look like hundreds of years from now, it is fair to break up the subject into three categories: mathematics, computational complexity, and computing technology” (Diffie 84). That’s exactly what one researcher did when he pointed out several facts concerning these three subcategories. To begin, calculus was just invented a little over 300 years ago (Diffie 84). In regard to the fast pace at which math has evolved in just the last 300 years, it becomes obvious that with it, cryptography might very well do the same in the

coming century. Similarly, Diffie describes how the topics of computational complexity and computing technology have also changed significantly in such a short amount of time (Diffie 85). All of these facts led researchers to believe cryptography may evolve at an absurd rate over the next few decades or that humans might soon discover something better (Diffie 86).

Assuming Cryptography will still be used, some of its future applications include: “improved digital signatures, privacy-enhancing technologies, new hash functions, and new ciphers” (Callas 16-20). Each of these topics could potentially have an impact across every career path and could have day-to-day implications on how people live their lives. For example, every business would use advanced forms of cryptography to protect information being sent within the company; therefore, many employees will take advantage of these systems. Also, when a person orders anything online, these new technologies will play a huge role in keeping their information safe.

In the case that Cryptography does evolve into something better, Quantum cryptography might even replace cryptography as a whole (Callas 21). Many refer to this as “quantum secrecy” because it is a way to use “quantum effects to entangle photons so that two photons are bound together, thereby keeping messages from being intercepted” (Callas 21). This is a very elaborate process that has a lot of room for improvement, with significant progress made over the past few years.

1.2.7 Ethics of Encryption

Cryptography raises several ethical concerns since it provides individuals the freedom to protect the privacy of their personal communications at the cost of decreased government monitoring. Using encryption is necessary to protect confidential information in branches of the

government along with the intellectual property of people like researchers and scientists.

However, it can be harmful by making it harder for intelligence agencies to track terrorists or carry out certain law enforcement activities in order to protect national interests (Matthews).

There are several examples, such as tactical military operations, that show that cryptography is in fact morally justified. Confidential information about military tactics can prove to be fatal if they accidentally fall into the hands of the enemy. If an armed force would not use encrypted communication systems, it would be risking its own soldiers lives. The military in such a situation would be morally obligated to use cryptography in order to protect the lives of both soldiers and civilians alike (Rogaway 34).

There are several realistic examples of such ethical dilemmas regarding the ethics of cryptography as well. In the 1990s, the National Security Agency (NSA) restricted the export of “most cryptographic hardware and software to products using a 40-bit or shorter key (Pfleeger 63).” Using a key that short made the brute-force attack method of cracking encrypted messages significantly easier. At the time, it was believed that this limitation was aimed at making it easier for the agency itself to hack into encrypted information. However, by intentionally implementing such a practice, the agency was exposing citizens to cyberattacks from foreign entities as well (Karsten and West).

More recently, in 2019, Senior Trump Administration officers met to discuss the prohibition of end-to-end encryption to make it easier for law enforcement agencies to access suspects’ data (Geller). Passing such legislation would affect the privacy and security of tens of millions of consumers and in a way force tech and security companies like Apple and Google to reduce security features on their devices. This potential legislation received severe backlash from

those who believed prohibiting encryption meant the government would be infringing upon their right to privacy (Raicu).

As mentioned by Falk, “Cryptography is in general morally permissible to use. At no time should cryptographic technologies be withheld from the general population because there are some situations in which the use of cryptography is a duty, and preventing such a fulfillment of duty is unethical in and of itself (Falk 45).” Ultimately, the general opinion about the use of cryptography is that the unethical use by a small number of people is no reason to deprive everyone else from the many of the benefits of cryptography in general.

2 Methods and Materials

2.1 Materials

The project was completed using MATLAB, a numerical computing program which can be used to manipulate matrices and run algorithms. MATLAB was used to create scripts for each of the five ciphers. Each script created a tabula recta, reformatted the inputted key and message, and encrypted or decrypted the inputted message (as prompted).

2.2 General Coding Methods

Because of the similarities in the algorithms for each cipher, the mechanisms for much of the MATLAB code were very similar. Several techniques such as while-loops, for-loops and conditional if-statements were used in order to maximize efficiency. The following sections outline the basic structure of the five scripts.

2.2.1 Creating the Tabula Recta

The variable (*MAT*, for example) used for the tabula recta was set equal to the alphabet. Then a for-loop was used to concatenate rows into *MAT*. The loop index variable (*k*) was set equal to the numbers two through 26, indicating the remaining rows of the tabula recta. In the body of the loop, the next line of *MAT* was referenced. *MAT* of *k* was set equal to the 2nd through last elements of the previous row, followed by the 1st element. These two arguments were then concatenated to form the tabula recta. Figure 5 shows sample code for this process.

```
51 % make the matrix-----
52 - MAT = [alphabet];
53 %the first row in the matrix is a ceasar shif of 0, which is the alphabet.
54 %after that, each row can be added by taking the last row and copying the
55 %everything from the second element to the last, followed by the first
56 %element
57 - for k = 2:length(alphabet)
58 -     MAT(k,1:26) = [MAT(k-1,2:26) MAT(k-1,1)];
59 - end
```

Figure 5: Code for the Tabula Recta

2.2.2 Inputs and Formatting

Each script gathered the following information using input commands: the key, the message, and whether the user wanted to encrypt or decrypt the inserted message. Several different methods such as a while-loop, switch-case, or menu option were used to determine whether the user wanted to encrypt or decrypt the message.

The ‘upper’ function was used on both the key and the message, which switched any lowercase letters to uppercase. Nested loops were then used to evaluate each character in the string to see if it matched a letter in the uppercase alphabet; if it wasn’t a match it was deleted. The first loop went through letters of the message, and the second loop went through letters of the alphabet. An if-statement was then used to identify characters that matched with the alphabet,

and if they did, those letters were stored in a new string, as seen in Figure 6. After going through each of the letters in the message, the new string variable would contain only uppercase letters from the old message, with no spaces, punctuation, or special characters.

```
%Step 1-Eliminating special characters for both the keyword and the message
while Letter>=1 %this is because we want the loop to run as long as we are at the 1st character, once
    Is_Letter=0;
    for u=1:26
        if Message(Letter)==Alphabet(u) %trying to remove everything thats not a letter
            Is_Letter= 1;
        end
    end %we want to start at the final character and work backwards because whether t
    if Is_Letter==0 %it's important that I am starting at the end of the phrase and working my w
        Message(Letter)= ''; %this eliminates any character that is not a letter in the alphabet whic
    end
    Letter=Letter-1; % this ensures that we move Backwards to the next letter if the first letter check
end
Keyword=upper(Keyword); %changes the keyword to all uppercase to ensure that the letters of the keywo
Letter=length(Keyword);
while Letter>=1 %this is because we want the loop to run as long as we are at the 1st charact
    Is_Letter=0;
    for u=1:26
        if Keyword(Letter)==Alphabet(u) %trying to remove everything that's not a letter
            Is_Letter= 1;
        end
    end %we want to start at the final character of the keyword and work backwards beca
    if Is_Letter==0 %it's important that I am starting at the end of the phrase and working m
        Keyword(Letter)= ''; %this eliminates any character from the key that is not a letter in
    end
    Letter=Letter-1; % this ensures that we move Backwards to the next letter if the first lett
end
```

Figure 6: Removing Special Characters

Next, the code would include a section which formatted the key to match the specifications of the cipher.

2.2.3 Encryption

If the user wanted the message to be encrypted, the following code was performed. The loop index variable of the for-loop for encryption was set equal to one through the length of the message. A second for-loop was nested within the first loop and was run to evaluate each of the rows of the tabula recta. In the body of the second for-loop, an if-statement was placed. The if-statement set a new indexing variable equal to the row number of the tabula recta for which the first element of that row was equal to the letter of the key that corresponded to the letter of the secret message being encrypted via the first for-loop. A second for-loop, with a second

if-statement, set another new indexing variable equal to the column number within the tabula recta for the first element in the column that was equal to the letter of the secret message being encrypted. After both nested for-loops, but inside the body of the first, the encrypted message was created. This was done by replacing each letter of the original message with the element of the tabula recta that was at the intersection of the row of the first new index variable and the column of the second new index variable. Figure 7 displays an example structure of the loops and if-statements for this section of code.

```

70 - for Letter=1:Length_of_Message
71 -     for i=1:26 %determine the row
72 -         if Alphabet(i)==Key(Letter)
73 -             current_row=i;
74 -         end
75 -     end
76 -     for i=1:26 %determine the column
77 -         if Alphabet(i)==Message(Letter)
78 -             current_column=i;
79 -         end
80 -     end
81 -     Encoded_Message(Letter)=Start(current_row,current_column);
82 -     %index the position of the tabula recta using the determined row
83 -     %and column
84 - end

```

Figure 7: General Encryption Process

2.2.4 Decryption

If the user's goal was to decrypt a message, the following code was performed. The loop index variable of a for-loop was set equal to one through the length of the message. Inside this loop, two more for-loops were created: the first one running through the rows of the tabula recta, and the second running through the columns, similar to the encryption process. The difference was that in this case, the first loop located the correct row that corresponded to the letter of the key that was being used for decryption, but the second loop searched through the row determined by the first loop to find the letter of the message that needed to be decrypted. Finally, the new

letter of the decoded message was set equal to the letter in the first row of the column that corresponded to the encrypted letter within the row number of the key. The details of this process are shown in Figure 8.

```

152 - for Letter=1:Length_of_Encoded_Message
153 -     for i=1:26 %determine which row we are going to be searching in to find the index of our encoded message
154 -         if Alphabet(i)==Key(Letter)
155 -             current_row=i;
156 -         end
157 -     end
158 -     for i=1:26 %determine which column we are going to be searching in to find the index of our Decoded message
159 -         if Start(current_row,i)==Encoded_Message(Letter)
160 -             current_column=i;
161 -         end
162 -     end
163 -     Decoded_Message(Letter)=Start(1,current_column);
164 - end

```

Figure 8: General Decryption Process

2.3 Cipher-Specific Variations

2.3.1 The Vigenère Cipher Key Generation

The Vigenère cipher script used a series of if-statements and a for-loop to ensure that the length of key was the same length as the message. The first if-statement checked to see if the key was already the same length as the message; if it was, the key was not altered. The “elseif” portion of the if-statement shortened the key if it was too long for the message. Then, the remaining “else” portion of the if-statement repeated the key until it was the appropriate length. This was done using a for-loop that ran until the length of the key matched the length of the message. Within the for-loop, an if-statement was placed to determine if the script needed to repeat the key entirely (possibly even multiple times), or if only a portion of the key needed to be repeated. If the difference between the key and the message was the length of the key, the code simply doubled the original key. Then, in the “else” portion of the if-statement, as seen in Figure 9, when the difference between the number of characters in the key and the message was not the

exact length of the key, or larger, only a portion of the key was repeated. This ensured that the key and the message would be the same length.

```
%Step 3-Getting the Keyword to the same length as the message
[DONTLOOKAT,Length_of_Key]=size(Keyword); %the DONTLOOKAT variable is irrelevant because it represents
[DONTLOOKAT,Length_of_Message]=size(Message); %the DONTLOOKAT variable is irrelevant because it represents
if Length_of_Key==Length_of_Message %nothing needs to be done to the Keyword
    Key=Keyword;
elseif Length_of_Key>Length_of_Message %Keyword needs to be shortened
    Key= Keyword(1:Length_of_Message);
else %if the key is less than the message and needs to be repeated until it's the proper length
    Key=Keyword;
    for o=1+Length_of_Key:Length_of_Message %this ensures that the key starts to repeat when it should
        if rem(o,Length_of_Key)==0
            remainder=Length_of_Key; %point of this code is to determine how many characters need to be
        else
            remainder=rem(o,Length_of_Key);
        end
        Key(o)=Keyword(remainder); %this will repeat the key word until you come to the same length as t
    end
end
```

Figure 9: Vigenère Cipher Key Editing Process

2.3.2 The Running-Key Cipher Key Generation

Similar to the Vigenère cipher, the Running-Key cipher used if-statements and a loop to make the key and the message the same length. However, a major difference in the Running-Key cipher was that the key inputted by the user initially had to be longer than the message so it wouldn't repeat. Specifically, a while-loop was used with an if-statement within its body to first ensure the user inserted a key that was longer than the message, as seen in Figure 10. If the length of the key was greater than or equal to the length of the message, it would pass through the while-loop onto an if-statement. This additional if-statement would create a new key which was either equal to the key that passed through the first while-loop, or it was a shortened version of that original key, ensuring that the key and message were the same length before proceeding through the encryption and decryption code.


```

% Now, use if statement to ensure use inputs a keyphrase that
% is longer than the message.
if length(keyphrase) >= length(Secret_M)
    bad = 1; % When bad is not 1, this if statement will prompt the user to type in a longer keyphrase.
else
    disp('Try a keyphrase that is longer than the message') % Once user inputs a keyphrase such that len
end
end

% After removing all of the special characters ...
% Get keyphrase and secret message to be the same length
lengthofkeyphrase = length(keyphrase);
lengthofmessage = length(Secret_M);

if lengthofkeyphrase == lengthofmessage;
    newkey = keyphrase; % If already equal, perfect. newkey will be the same as keyphrase
elseif lengthofkeyphrase > lengthofmessage;
    newkey = keyphrase(1:lengthofmessage); % If keyphrase is greater, newkey will just be the terms 1 through
end

```

Figure 10: Running-Key Cipher Key Editing Process

2.3.3 The Auto-Key Cipher Key Generation

The Auto-Key cipher script used a series of conditional if-statements within a loop to ensure that the length of the key was similar to that of the message as seen below in Figure 11. If the key inserted by the user was the same length as the message, the key would simply remain unchanged. If the key was longer than the message, only enough letters of the key would be a part of the final key as many letters there were in the message. Lastly, if the key was shorter than the message, the final key would be the original key plus enough letters of the message concatenated to the end of the key to make the final key the same length as the message.

```

% Step 4 - Determine the Key -----
%(The goal is to make the Key the same size as the message)
if length(Keyword) == length(Secret_Message)
    Key = Keyword; %If the Keyword and message are the same length, the Key will simply be the Key
elseif length(Keyword) > length(Secret_Message)
    Key = Keyword(1:length(Secret_Message)); %If keyword is greater than message, only enough
elseif length(Keyword) < length(Secret_Message) %If the keyword has a length less than that of the
    difference = length(Secret_Message) - length(Keyword); %The difference between both le
    Remainder = Secret_Message(1:difference); %A new string remainder is created which is
    Key = [Keyword Remainder]; %The key is now the initial keyword + the "remainder" (lett
end
fprintf(' The Key is: %s\n',Key) %Displays the final key that is the same length as the message

```

Figure 11: Autokey Cipher key Modification Process

2.3.4 Beaufort Cipher Encoding Process

For the Beaufort cipher, the key was generated through the same process as the Vigenère cipher. However, this cipher had a completely different encryption process. First, a vector of numbers representing the secret message was created. This was done by setting a loop to run through the letters of the secret message (using the index variable, m , for example). Within the body of that loop, a second for-loop was placed with the loop index variable, a , which was set equal to one through the number of letters in the alphabet. Within this second loop, an if-statement evaluated whether the m 'th letter of the secret message was equal to the a 'th letter of the alphabet, and if it did, it set the m 'th letter of the new vector equal to the value of a . Example code is shown in Figure 12.

```
62 - SMN = [];  
63 - for m = 1:1th %for each letter in the message  
64 -     for a = 1:26 %for each letter in the alphabet  
65 -         if SM(m) == alphabet(a) %if they equal one another,  
66 -             SMN(m) = a; %take the number that was the position of the letter  
67 -             %of the alphabet that matched, and put it in the SMN vector  
68 -         end  
69 -     end  
70 - end
```

Figure 12: Beaufort Cipher Message to Numbers

After this process, another set of nested for-loops was coded, with the outer loop running through each number in the vector representing part of the secret message. The inner loop ran through the rows of the tabula recta. Within this loop, an if-statement evaluated whether the letter in the tabula recta at the intersection of the column of the number of the secret message vector and the row indicated by the nested for-loop matched the corresponding letter of the key. The code inside the if-statement used indexing to replace the letter of the plaintext secret

message with the letter of the alphabet that corresponded to the number of the row of the tabula recta for which the if-statement was true, as seen in Figure 13.

```

72 - for LN = 1:1th
73 -     for PR = 1:26
74 -         if MAT (PR, SMN (LN) ) == KEY (LN)
75 -             SM (LN) = alphabet (PR);
76 -         end
77 -     end
78 - end

```

Figure 13: Beaufort Cipher Encryption Process

Another difference between the script for this code was that the decryption process was the same as the encryption process. Therefore, no switch-case was used other than to display the final result in a statement that mentioned whether the message had been encrypted (in one case) or decrypted (in the other case).

2.3.5 Porta Cipher Encoding Process

The parts of the script that were unique for the Porta cipher were the code for the encryption table (an altered form of the tabula recta) and the lack of a decryption process, as it was the same as the encryption process. The key for this cipher was generated using the same method as for the Vigenère cipher. The encryption/decryption process was identical to the general encryption process for the Vigenère, Running-Key, and Auto-key ciphers.

The encryption table was created through the following steps. Two loops created two matrices (*MAT1* and *MAT2*, for example), which were then concatenated side by side into one matrix. The first row of *MAT1* was set equal to the last half of the alphabet. Then, a for-loop with a loop index variable equal to the 2nd through 13th rows of the matrix was established. In the body of the loop, rows were added to *MAT1* by concatenation. The new row, in brackets, would

include the 2nd through 13th elements of the previous row followed by the 1st element of the previous row. The first row of *MAT2* was set equal to the first half of the alphabet. Then a second loop assigned the rest of the rows in the same manner as the previous loop, with the only difference being that the new row referenced the 13th element of the previous row followed by the 1st through 12th elements, as shown in Figure 14.

```

140 - MAT1 = [alphabet(14:26)]; %the first row is just the second half of the
141     %alphabet, in order
142 - for k = 2:13
143     MAT1 = [MAT1 ; [MAT1((k-1),2:13) MAT1((k-1),1)]];
144 - end
145     %the second matrix is made pretty much like the first, just backwards
146     MAT2 = [alphabet(1:13)];
147 - for k = 2:13
148     MAT2 = [MAT2 ; [MAT2((k-1),13) MAT2((k-1),1:12)]];
149 - end
150 - MAT = [MAT1 MAT2];

```

Figure 14: Code for Porta Cipher Encryption Table

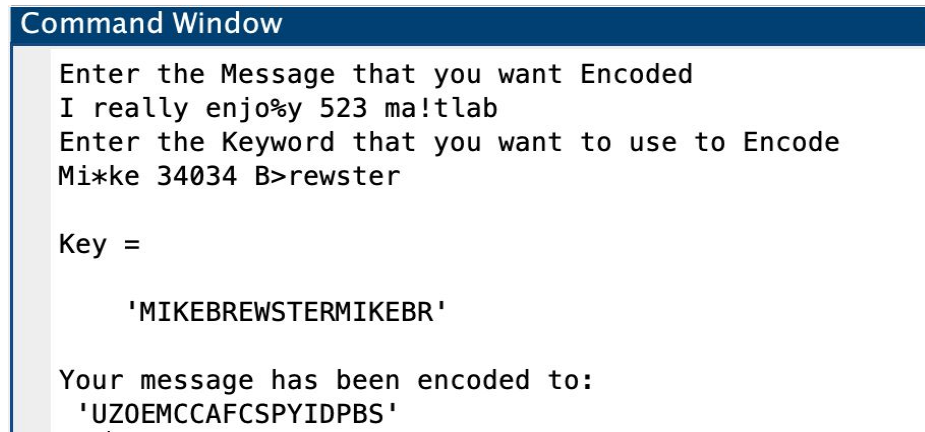
3 Results

3.1 Encryption Results

In order to maintain consistency and be able to accurately compare outputs, the same message and key was determined to be tested for all five ciphers. These inputs had numbers and misplaced special characters to test if the script could efficiently eliminate them as well from the final message and key. The desired message “I really enjo%y 523 ma!tlab” along with the key “Mi*ke 34034 B>rewster” was inputted. For all five ciphers, the script eliminated the special characters and changed all letters to uppercase, yielding “IREALLYENJOYMATLAB” and “MIKEBREWSTER” as the message and key to be used in each cipher-specific encryption process.

3.1.1 Vigenère Cipher Encryption Output

Once the inputted message and key were fixed, the key was adjusted to be the same length as the message and the corresponding key to be used for encryption was determined to be “MIKEBREWSTERMIKEBR.” The final encrypted message then read “UZOEMCCAFCSPIYPBS” as seen in Figure 15.

A screenshot of a MATLAB Command Window. The title bar is dark blue with the text "Command Window". The window contains the following text: "Enter the Message that you want Encoded", "I really enjo%y 523 ma!tlab", "Enter the Keyword that you want to use to Encode", "Mi*ke 34034 B>rewster", "Key =", "'MIKEBREWSTERMIKEBR'", "Your message has been encoded to:", "'UZOEMCCAFCSPIYPBS'".

```
Command Window
Enter the Message that you want Encoded
I really enjo%y 523 ma!tlab
Enter the Keyword that you want to use to Encode
Mi*ke 34034 B>rewster

Key =

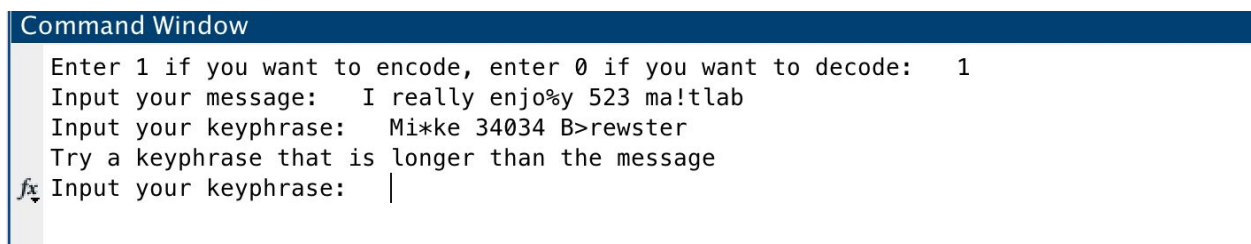
'MIKEBREWSTERMIKEBR'

Your message has been encoded to:
'UZOEMCCAFCSPIYPBS'
```

Figure 15: Vigenère Cipher Encryption Output

3.1.2 Running-Key Cipher Encryption Output

Once the inputted message and key were fixed, the command window displayed “try a keyphrase that is longer than the message” and proceeded to prompt the user to input a longer key as seen in Figure 16.

A screenshot of a MATLAB Command Window. The title bar is dark blue with the text "Command Window". The window contains the following text: "Enter 1 if you want to encode, enter 0 if you want to decode: 1", "Input your message: I really enjo%y 523 ma!tlab", "Input your keyphrase: Mi*ke 34034 B>rewster", "Try a keyphrase that is longer than the message", "fx Input your keyphrase: |".

```
Command Window
Enter 1 if you want to encode, enter 0 if you want to decode: 1
Input your message: I really enjo%y 523 ma!tlab
Input your keyphrase: Mi*ke 34034 B>rewster
Try a keyphrase that is longer than the message
fx Input your keyphrase: |
```

Figure 16: Running-Key Cipher Attempt to Input Small Key

To proceed, the first seven letters of the alphabet were added to the end of the original key to create a new key “MIKEBREWSTERABCDEFG.” The code ran and outputted “UZOEMCCAFCSMPBVOEG” as the final encrypted message.

3.1.3 Auto-Key Cipher Encryption Output

Once the inserted key and message were fixed, the key was adjusted to be the same length as the message and the corresponding key to be used for encryption was determined to be “MIKEBREWSTERIREALL.” The final encrypted message then read “UZOEMCCAFCSPURXLLM” as shown in below in Figure 17.

Encrypt

```
Enter your secret message: I really enjo%y 523 ma!t!ab
The Message entered is: IREALLYENJOYMATLAB
Enter your Keyword: Mi*ke 34034 B>rewster
The Keyword entered is: MIKEBREWSTER
The Key is: MIKEBREWSTERIREALL
The Encrypted Message is: UZOEMCCAFCSPURXLLM
```

Figure 17: Auto-Key Cipher Encryption Output

3.1.4 Beaufort Cipher Encryption Output

When prompted by the user to encrypt, and the same message and key were entered, the final key “MIKEBREWSTERMIKEBR” and the encrypted message “ERGEQGGSFQKQTAIRTBQ” were produced in the command window as seen in Figure 18 below.

```
Command Window
Enter message to be encrypted:
I really enjo%y ma!tlab
Enter key:
Mi*ke 34034 B>rewster
The encrypted message reads:
"ERGEQGGSEFKQTAIRTBQ"
```

Figure 18: Beaufort Cipher Encryption Output

3.1.5 Porta Cipher Encryption Output

When prompted to encrypt, and the same message and key were entered, the resulting key “MIKEBREWSTERMIKEBR,” and encrypted message “OAWPYTJPESMDSRBNNW” were produced. The resulting view of the command window is shown in Figure 19.

```
Command Window
Enter message to be encrypted:
I really enjo%y ma!tlab
Enter key:
Mi*ke 34034 B>rewster
The encrypted message reads:
"OAWPYTJPESMDSRBNNW"
```

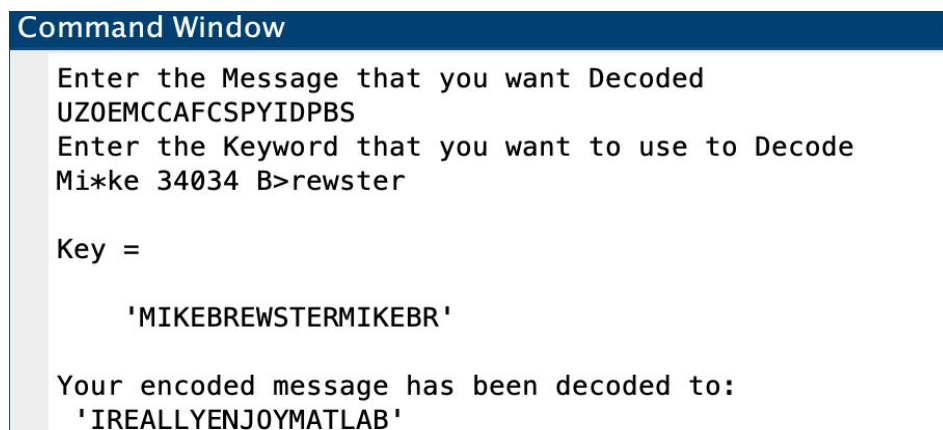
Figure 19: Porta Cipher Encryption Output

3.2 Decryption Results

For the decryption process, the final output from each cipher’s encryption script was used as the new inputted message with some misplaced special characters, along with the same key, “Mi*ke 34034 B>rewster.” Before proceeding to decryption, all special characters, numbers, and punctuation marks were removed and all letters were changed to uppercase as needed.

3.2.1 Vigenère Cipher Decryption Output

After eliminating special characters, the script yielded “UZOEMCCAFCSPIYPBS” as the message used for decryption and “MIKEBREWSTER” as the key. After modifying the key to be the appropriate length, the new key “MIKEBREWSTERMIKEBR” was generated, similar to the key used for encryption. The final decrypted message displayed was “IREALLYENJOYMATLAB” as seen in Figure 20.



```
Command Window
Enter the Message that you want Decoded
UZOEMCCAFCSPIYPBS
Enter the Keyword that you want to use to Decode
Mike 34034 B>rewster

Key =

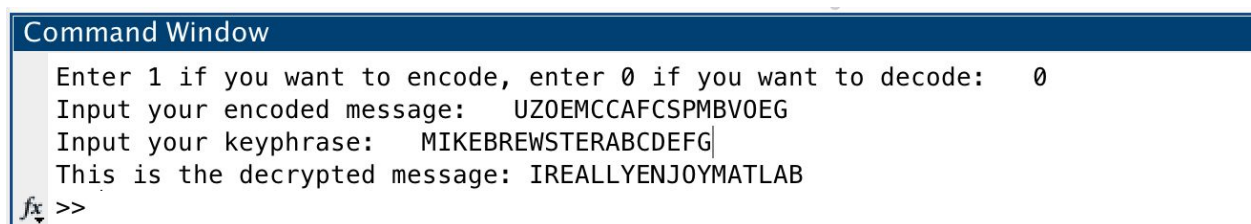
      'MIKEBREWSTERMIKEBR'

Your encoded message has been decoded to:
      'IREALLYENJOYMATLAB'
```

Figure 20: Vigenère Cipher Decryption Output

3.2.2 Running-Key Cipher Decryption Output

After eliminating special characters, the script yielded “UZOEMCCAFCSPIYPBS” as the message used for decryption and “MIKEBREWSTERABCDEFGH” as the final key. The final decrypted message displayed was “IREALLYENJOYMATLAB” as seen in Figure 21 below.



```
Command Window
Enter 1 if you want to encode, enter 0 if you want to decode:    0
Input your encoded message:  UZOEMCCAFCSPIYPBS
Input your keyphrase:  MIKEBREWSTERABCDEFGH
This is the decrypted message: IREALLYENJOYMATLAB
fx >>
```

Figure 21: Running-key Cipher Decryption Output

3.2.3 Auto-Key Cipher Decryption Output

After eliminating special characters, the script yielded “UZOEMCCAF CSPURXLLM” as the message used for decryption and “MIKEBREWSTER” as the key. After modifying the key to be the same length as the message, a new key “MIKEBREWSTERIREALL” was produced just like the key for encryption. The final decrypted message displayed was “IREALLYENJOYMATLAB,” as shown in Figure 22 below.

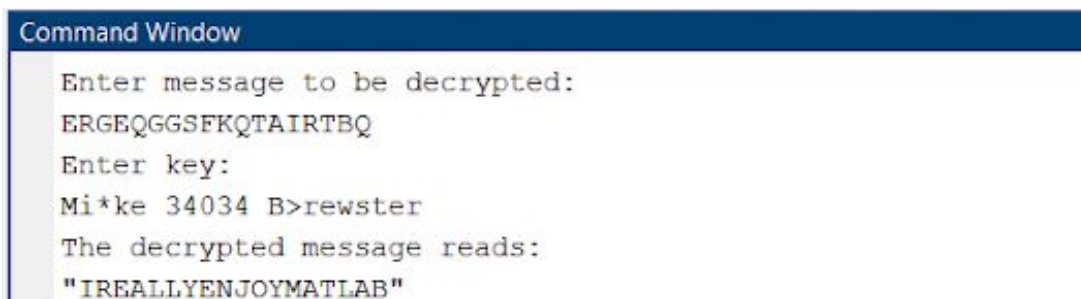
Decrypt

```
Enter your encrypted message: UZOEMCCAF CSPURXLLM
The Encrypted Message entered is: UZOEMCCAF CSPURXLLM
Enter your keyword: Mi*ke 34034 B>rewster.
The Keyword entered is: MIKEBREWSTER
The Key is: MIKEBREWSTERIREALL
The Secret Message is: IREALLYENJOYMATLAB
```

Figure 22: Auto-key cipher decryption output

3.2.4 Beaufort Cipher Decryption Output

When the encrypted message, “ERGEQGG SFKQTAIRTBQ” was entered with the same key, the result was “IREALLYENJOYMATLAB.” The key, after alteration, was “MIKEBREWSTERMIKEBR.” Figure 23 shows the display in the command window.



```
Command Window
Enter message to be decrypted:
ERGEQGG SFKQTAIRTBQ
Enter key:
Mi*ke 34034 B>rewster
The decrypted message reads:
"IREALLYENJOYMATLAB"
```

Figure 23: Beaufort Cipher Decryption Output

3.2.5 Porta Cipher Decryption Output

The encrypted message, “OAWPYTJPESMDSRBNNW” was entered, along with the same key. The result was “IREALLYENJOYMATLAB” as shown in Figure 24. The key, after alteration, was “MIKEBREWSTERMIKEBR.”

```
Command Window
Enter message to be decrypted:
OAWPYTJPESMDSRBNNW
Enter key:
Mi*ke 34034 B>rewster
The decrypted message reads:
"IREALLYENJOYMATLAB"
```

Figure 24: Porta Cipher Decryption Output

3.3 Condensed Output for all Ciphers

Table 1 below shows how each of the five ciphers successfully eliminated special characters, spaces, punctuation marks, and numbers from the inserted message and key for encryption.

Table 1: Elimination of Special Characters

Cipher	Initial Message	Edited Message	Initial Key	Edited Key
Vigenere	"I really enjo%y 523 ma!tlab"	"IREALLYENJOYMATLAB "	"Mi*ke 34034 B>rewster"	"MIKEBREWSTERMIKEBR"
Running-Key	"I really enjo%y 523 ma!tlab"	"IREALLYENJOYMATLAB "	"Mi*ke 34034 B>rewster"	"MIKEBREWSTERABCDEFGF"
Auto-Key	"I really enjo%y 523 ma!tlab"	"IREALLYENJOYMATLAB "	"Mi*ke 34034 B>rewster"	"MIKEBREWSTERIREALL"
Beaufort	"I really enjo%y 523 ma!tlab"	"IREALLYENJOYMATLAB "	"Mi*ke 34034 B>rewster"	"MIKEBREWSTERMIKEBR"
Porta	"I really enjo%y 523 ma!tlab"	"IREALLYENJOYMATLAB "	"Mi*ke 34034 B>rewster"	"MIKEBREWSTERMIKEBR"

The different encrypted messages for each of the five ciphers, along with the edited message and key is shown below in table 2.

Table 2: Encryption Outputs

Cipher	Edited Message	Edited Key	Encoded Message
Vigenere	“IREALLYENJOYMATLAB”	“MIKEBREWSTERMIKEBR”	“UZOEMCCAFCSPYIDPBS”
Running-Key	“IREALLYENJOYMATLAB”	“MIKEBREWSTERABCDEFG”	“UZOEMCCAFCSPMBVOEG”
Auto-Key	“IREALLYENJOYMATLAB”	“MIKEBREWSTERIREALL”	“UZOEMCCAFCSPURXLLM”
Beaufort	“IREALLYENJOYMATLAB”	“MIKEBREWSTERMIKEBR”	“ERGEQGGSFKQTAIRTBQ”
Porta	“IREALLYENJOYMATLAB”	“MIKEBREWSTERMIKEBR”	“OAWPYTJPESMDSRBNNW”

After removing any special characters from the encrypted message and key inputted by the user, the edited key and message were used to generate a decrypted message as shown below in table 3.

Table 3: Decryption Outputs

Cipher	Edited Encoded Message	Edited Key	Decoded Message
Vigenere	“UZOEMCCAFCSPYIDPBS”	“MIKEBREWSTERMIKEBR”	“IREALLYENJOYMATLAB”
Running-Key	“UZOEMCCAFCSPMBVOEG”	“MIKEBREWSTERABCDEFG”	“IREALLYENJOYMATLAB”
Auto-Key	“UZOEMCCAFCSPURXLLM”	“MIKEBREWSTERIREALL”	“IREALLYENJOYMATLAB”
Beaufort	“ERGEQGGSFKQTAIRTBQ”	“MIKEBREWSTERMIKEBR”	“IREALLYENJOYMATLAB”
Porta	“OAWPYTJPESMDSRBNNW”	“MIKEBREWSTERMIKEBR”	“IREALLYENJOYMATLAB”

4 Discussion

While all ciphers were successful in both encrypting and decrypting messages based on the user's preference, not all of the above tested ciphers were determined to be safe in order to provide end-to-end user encryption. All five ciphers were similar in the sense that they used the tabula recta to both encrypt and decrypt messages. However, the primary difference lies in the determination of the key for each cipher.

The foundation for all five of these ciphers was the Caesar cipher, a primitive encryption technique that used a simple shift in the alphabet several times. The simplistic foundation of these ciphers is what made them unreliable to provide secure communication through the internet in today's modern world. Nonetheless, each cipher was able to encrypt and decrypt messages as required by the user.

The Vigenère cipher formed a key that was easy to break. This was because the key repeated until it was as long as the message. Since the Porta and Beaufort cipher formed their keys the same way as the Vigenère, they were also easy to discover by cryptanalysts. However, the only thing making the Porta cipher easier to crack than the Vigenère was the fact that it had 13 rows in the tabula recta rather than the standard 26 rows.

The Auto-Key cipher was undoubtedly the easiest cipher to crack primarily since the key could include a portion of the actual message that was intended to be hidden, posing a greater risk for the invasion of privacy. In contrast, the Running-key cipher's key was always unique and never needed to repeat due to its long initial length. The fact that the Running-Key cipher used

complex keys like names of books or phrases from a movie proved it to be the best and most secure key that was difficult to guess or find patterns in.

Even though the methods for generating the key were unique for almost all of the ciphers, each cipher successfully yielded the same decrypted message when given the same initial message. This was because the decryption process for all ciphers simply undid the different encryption techniques. So, as long as the same initial message was entered, each cipher would output a unique encrypted message, but the exact same decrypted message.

Being able to encrypt secret messages was beneficial to the overall public safety and welfare since it provided users with a safe way to communicate with one another. This research effectively highlighted the weaknesses of ciphers based off of the primitive Caesar cipher and the negative implications it could have on the economy and the privacy of individuals if major data breaches occur as a result of poor encryption techniques. The global security industry was worth well over 130 billion dollars by the end of 2018 and this project certainly highlighted the overall importance of constantly finding weaknesses in encryption techniques to find newer and more secure networks for communication and storage of confidential information.

5 Conclusion

In conclusion, all five ciphers successfully completed the tasks of encrypting and decrypting the messages inputted by the user. Each script successfully eliminated undesired special characters, punctuation, and spaces and made all letters of the key and message uppercase. In addition, the final key was made the same length as the message being encrypted/decrypted using the appropriate process for each cipher. Finally, a series of for-loops and conditional if-statements successfully encrypted and decrypted the message using the tabula

recta as a reference. Each script and type of cipher helped the user successfully keep their private communication and information secure on the interweb.

5.1 Future Work

There are several advanced encryption techniques that provide a higher level of security than the ciphers used within the scope of this project. In the future, some of these more complex processes for encoding and decoding should be considered, particularly advanced quantum cryptography methods. Similarly, instead of encrypting messages, cryptanalysis and trying to crack encrypted messages should be considered in order to expose weaknesses of these ciphers.

In addition, the existing code could be simplified by editing the majority of for-loops and replacing them with the find function. This would shorten the scripts while making it easier for less experienced MATLAB users to understand the encryption and decryption process, while still being able to complete the same process. Doing so would enable the script faster to run as well.

Moreover, more display commands could be used throughout the code to make the process of the script easier to understand and interpret for the user. The script could be made more engaging and easier to use for the user by embedding user-friendly graphical interfaces within the algorithm, which could simplify the process of entering the key and the message.

In order to evaluate safe encryption techniques, instead of working solely on polyalphabetic substitution ciphers, playfair, transposition, and hill ciphers can also be evaluated in order to provide one with a better understanding of different ciphering mechanisms. Overall, there are a large number of possibilities for improvement and simplification that could be implemented in the future to secure confidential information from third parties.

6 References

- Agudo, Isaac, et. al. "Cryptography Goes to the Cloud." *Communications in Computer and Information Science*, vol. 187, 2011, pp 190-197. *Springer Link*,
https://doi.org/10.1007/978-3-642-22365-5_23 Accessed 14 Apr. 2020.
- Callas, Jon. "The Future of Cryptography." *Information Systems Security*, vol. 16, no. 1, Jan. 2007, pp. 15–22. *Academic Search Alumni Edition*, EBSCOhost,
[doi:10.1080/10658980601051284](https://doi.org/10.1080/10658980601051284). Accessed 14 Apr. 2020.
- "Ciphers." *Practical Cryptography*, James Lyons, 2012, practicalcryptography.com/ciphers/.
Accessed 13 Apr. 2020.
- C. P. Pfleeger, S.L. Pfleeger, J. Margulies, *Security in Computing*. Fifth edition. Accessed 13 Apr. 2020.
- Cruise, Brit. "The Caesar Cipher." *Khan Academy*, Khan Academy,
www.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher.
Accessed 17 April 2020
- "Crypto." *Practical Cryptography*, practicalcryptography.com/cryptanalysis/. Accessed 13 Apr. 2020.
- "Cryptography" *Case Western Reserve University*,
<https://case.edu/artsci/math/singer/Sage/index.shtml> Accessed 15 April 2020

- Diffie, Whitfield. "Ultimate Cryptography." *Communications of the ACM*, vol. 44, no. 3, 2001, p. 84., doi:10.1145/365181.365214. Accessed 14 Apr. 2020.
- Falk, Courtney. "The Ethics of Cryptography." *The Ethics of Cryptography*, Research Gate, 21 Mar. 2016,
https://www.researchgate.net/publication/237217965_The_Ethics_of_Cryptography. Accessed 14 Apr. 2020.
- Gaines Helen Fouché. *Cryptanalysis. A Study of Ciphers and Their Solutions*. Dover, 1956.
Accessed 17 Apr. 2020.
- Geller, Eric. "Trump Officials Weigh Encryption Crackdown." *POLITICO*, 27 June 2019,
www.politico.com/story/2019/06/27/trump-officials-weigh-encryption-crackdown-1385306. Accessed 17 Apr. 2020.
- Giovanni. "Cryptanalysis and Attacks." *Experts Exchange*, Experts Exchange, 18 Apr. 2020,
www.experts-exchange.com/articles/12460/Cryptanalysis-and-Attacks.html. Accessed 17 Apr. 2020.
- Hasnat, Abul, et al. *Color Image Share Cryptography: a Novel Approach | Request PDF*.
www.researchgate.net/publication/330897035_Color_image_share_cryptography_a_novel_approach. Accessed 17 Apr. 2020.
- Jackob, Melis. "History of Encryption." *Information Security Reading Room*. SANS Institute, 8 Aug. 2001. Accessed 15 Apr. 2020.
- Jacquez, Geoffrey, et al. "Geospatial Cryptography: Enabling Researchers to Access Private, Spatially Referenced, Human Subjects Data for Cancer Control and Prevention." *Journal*

- of Geographical Systems*, vol. 19, no. 3, July 2017, pp. 197–220. *Academic Search Alumni Edition*, EBSCOhost, doi:10.1007/s10109-017-0252-3. Accessed 14 Apr. 2020.
- Karsten, Jack, and Darrell M. West. “A Brief History of U.S. Encryption Policy.” *Brookings*, Brookings, 29 July 2016, www.brookings.edu/blog/techtank/2016/04/19/a-brief-history-of-u-s-encryption-policy/. Accessed 14 Apr. 2020.
- Lyons, James. “Ciphers.” *Practical Cryptography*, 2012, practicalcryptography.com/ciphers/classical-era/Beaufort/. Accessed 14 April 2020.
- Martin, Jennifer. “Encryption Backdoors: A Discussion of Feasibility, Ethics, and the Future of Cryptography.” *Seattle Pacific Library*, 3 June 2017, <https://digitalcommons.spu.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1062&context=honorsprojects>. Accessed 14 Apr. 2020.
- Matthews, Kayla. “7 Advantages of Using Encryption Technology for Data Protection.” *SmartData Collective*, 10 Apr. 2020, www.smartdatacollective.com/5-advantages-using-encryption-technology-data-protection. Accessed 12 Apr. 2020.
- Raicu, Irina. “Ethical Questions About Encryption.” *Markkula Center for Applied Ethics*, www.scu.edu/ethics/focus-areas/technology-ethics/resources/ethical-questions-about-encryption/. Accessed 14 Apr. 2020.
- Rogaway, Phillip. *The Moral Character of Cryptographic Work*. University of California, Davis, Dec. 2015, web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf. Accessed 11 Apr. 2020.
- Rouse, Margaret. “What Is Cryptanalysis? - Definition from WhatIs.com.” *SearchSecurity*,

TechTarget, 30 Jan. 2018, searchsecurity.techtarget.com/definition/cryptanalysis.

Accessed 13 Apr. 2020.

T. W. Arnold, et al., "IBM 4765 cryptographic coprocessor," *IBM Journal of Research and Development*, vol. 56, no. 1.2, pp. 10:1-10:13, Jan.-Feb. 2012. Accessed 13 Apr. 2020.

“What Types of Encryption Are There?” *Information Commissioner's Office*, The United Kingdom,
ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/encryption/what-types-of-encryption-are-there/. Accessed 16 April 2020.