University of Louisville
ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

7-2008

Re-engineering the Enigma cipher.

Max Samuel Stoler
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Recommended Citation

Stoler, Max Samuel, "Re-engineering the Enigma cipher." (2008). *Electronic Theses and Dissertations.* Paper 1389.
https://doi.org/10.18297/etd/1389

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

Re-engineering the Enigma Cipher

By

Max Samuel Stoler
B.S., University of Louisville, 2007

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

July 2008

Re-engineering the Enigma Cipher

Submitted by: _____

Max Stoler

A Thesis Approved On

_____

(Date)

by the Following Reading and Examination Committee

_____

Dr. Ahmed Desoky, Thesis Director

_____

Dr. Adel Elmaghraby

_____

Dr. John Naber

Abstract:

The design of this thesis is to re-engineer the Enigma cipher to make it a viable, secure cipher for use on current computers. The goal is to create a cipher based on an antiquated mechanical cryptography device, the Enigma Machine, in software and improve upon it.

The basic principle that is being expounded upon here is that while the Enigma cipher's security was originally very dependent on security through obscurity, this needs to be secure on its own. Also, this must be a viable solution for the encryption of data based on modern standards.

The Enigma Phoenix, the name for this new cipher, will use Galois functions and other modern improvements to add an extra level of security to it and to make it the viable solution that is desired.

Index of Tables

Index of Figures

TABLE OF CONTENTS

APENDICIES

Appendix A: Design Diagrams
Appendix B: Source Code
Appendix C: ASCII Table of Values for Enigma Phoenix Encrypted File

1. Introduction:

1.1 History

What is being addressed in this thesis is the fact that a cipher (an algorithm for performing encryption and decryption) used in World War II was not secure, it was broken. This cipher was called the Enigma cipher. The goal of this thesis is to modify the Enigma cipher in such a way as to make it secure by using knowledge and functions gained from modern ciphers as well as modifications of the original cipher that stay true to the spirit of the original incarnation.

The field of cryptography goes as far back as ancient Greek and Roman times when simple transposition and substitution ciphers were used to encrypt sensitive messages. The process of increasing security in ciphers slowly continued until mechanical devices became prevalent in the early 20th century. Before this time there had been attempts at making secure ciphers. In the 16th century Blaise de Vigenere published his description of an Autokey cipher in 1586 which would be later used in World War I (though heavily modified) as the Vigenere Cipher. This was actually misattributed to Vigenere as it was originally Giovan Battista Bellaso who published *La cifra del. Sig. Giovan Battista Bellaso* in 1553 describing polyalphabetic substitution cipher [1][(Wolfram, 2002)].

After World War I, and the breakdown of the Vigenere Cipher, it was seen that greater security was needed. It was also determined that a person could not encrypt data as well as a mechanical device [1][(Wolfram, 2002)]. Hence, the development of the Enigma Machine was pushed forward. The Enigma Machine was a cipher machine used to encrypt and decrypt secret messages. The whole Enigma line was a family of electro-

mechanical rotor machines with a number of different models. The machine itself was used commercially from the 1920s onward and was adopted by several different military and governmental agencies, the most famous of these being Nazi Germany during and prior to World War II [1][(Wolfram, 2002)].

The most famous version of the Enigma Machine is the German military version used during World War II, the Wermacht Enigma. This version of the Enigma machine has been made famous in history, legend, and Hollywood films. The ironic fact is that the machine is so famous for its failure to be secure. The Fact that the Allies were able to break the Enigma was considered to be the best kept secret of World War II second only to the atomic bomb [1][(Wolfram, 2002)].

1.1.1 Focus

The main focus of this thesis is to re-engineer the German military model, the Wermacht Enigma, and create a derivation of the cipher that is secure on modern computers. During World War II the German forces had full confidence in the Engima cipher machine who depended on its security. This was somewhat misplaced confidence; however, it was based on the fact that the machine had a rather large (for the time) key space. The first computers were actually created to break the Enigma cipher, those being the Bombes created by British cryptologists (Alan Turing being a member of this team) [2][(Berghel, April 2008)].

1.2 What is Cryptography?

Cryptography is the process of taking a set of data and making it secure and secret by encoding and/or scrambling it. Secret keys, which will be discussed later, are generated by the cryptographic algorithm (a cipher), and these keys combined with the

strength of an algorithm and the type of network it is being used on determine the cipher's strength [3][(Algred J.Menezes, 1996)]. There are many different types of cryptographic algorithms. The most basic is a symmetric key encryption. The symmetric key cipher only requires that a single key be used to encrypt and decrypt the data. That is, the single key scrambles the data and is placed within the data in a manner allowing for the data to be unscrambled with the same key, usually using an XORed inverse in some way [4][(Schneier, 1996)].

Another type of cipher is a two key cipher, called public-key encryption. When the operations of the two-key asymmetric ciphers are performed, they are quite different from that of symmetric key encryption. They can be more secure, providing confidentiality, as well as offering integrity checking and verification of the author for non-repudiation. However, certain ciphers, like Rijndael's AES are symmetric ciphers that perform a number of separate operations to make them secure. [5][(Schneier, 1996)]

1.2.1 Basics of Cyrptography, Types of Ciphers

Cryptographic algorithms today consist of a few main structures. Most attempt to create confusion and diffusion of the input data. One such example of this type of algorithm is Feistel networks. One of the main advantages to the Feistel Networks is that their design is much like the Enigma cipher. Encryption and decryption are nearly identical, the only requirement being that the key schedule is reversed.

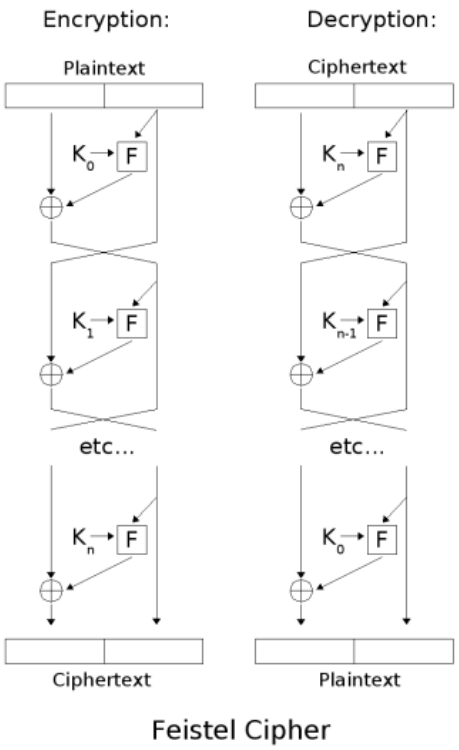


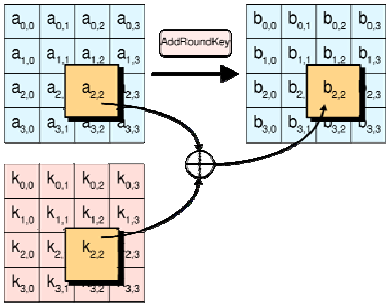
Figure 1: Feistel Network

The above Figure 1 [6][(Fiestel Cipher, 2008)] represents the process of encryption and decryption of a plaintext and ciphertext version of a document. During the encryption, the block is broken into separate halves. The first block is processed through by the function F where F performs the diffusion and confusion of the data. This is done through any method chosen by the cipher. The next half is run through as a step/part of function F or as a portion of the key K mixed into the data with an XOR. This process is thus repeated with the two halves swapping places and continuing until the full key has been utilized. When decrypting it is the same algorithm simply with the stream reversed. [7][(Schneier, 1996)]

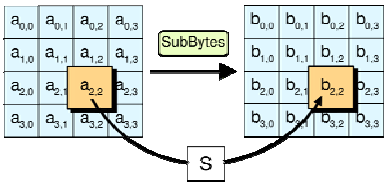
Enigma is a polyalphabetic substitution cipher. However, Enigma is more closely related to a Feistel network. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. This is not a cipher that is in common use today, however, as it is what Enigma originally was, it is important to discuss it here. The Vigenère cipher is one of the most well known examples of a simple polyalphabetic cipher. [8][(Miller, 2008)]

1.2.1.1 Basics of AES

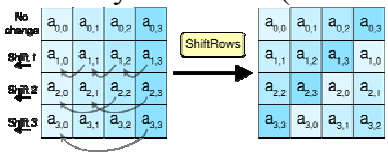
The Rijndael cipher, better known as the Advanced Encryption Standard (AES), is a Substitution Permutation Network cipher. This is a series of linked mathematical operations used in block ciphers, which is precisely what AES is. These consist of S-Boxes that transform blocks of input bits into blocks of output bits, usually these are done through hardware-efficient functions such as exclusive-or (XOR). This borrows a lot from the mixing functions of Feistel ciphers. Rijndael's cipher consists of more than simply reversing the key for encryption and decryption. [9][(Vaudenay, 2006)]



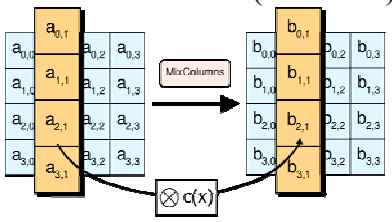
XOR Key and State (Confusion)



Substitute Bytes of State (Confusion)



Shift Rows of State (Diffusion)



MixColumns of State (Diffusion)

Figure 2: The four steps of AES

Above, in the Figure 2 [10], it can be seen that AES consists of four main steps. These four steps are: AES mixes the key by XORing with the state; AES then substitutes the bytes of the state using an S-box (Substitution box to obscure the relationship between the plaintext and the ciphertext); it then shifts the rows in the block; lastly it mixes the columns in each block. [11][(Vaudenay, 2006)]

The first step above is demonstrating how the block of plaintext, also known as the state, is XORed with part of the key. Below, in Table 1 is the truth table for XORing values. The next step in Figure 2 [10] is SubBytes which is used in several Substitution-

Permutation Network (SPN) ciphers (like AES). This uses an S-Box to create unique output for each input. ShiftRow is a diffusion step which rotates rows in the state. MixColumns is also a diffusion step. MixColumns uses matrix multiplication in the Galois Field to diffuse bits within each column among all four column entries. This all comes together when these functions of confusion and diffusion work together to provide the security for the algorithm of AES. Only the inverse functions are called in an opposite order to do decryption. S-boxes, or substitution boxes, are the core of most SPN ciphers. They are responsible for a great deal of the confusion aspect of the ciphers. How it works is that an 8-bit S-box contains 256 unique entries. That is, one entry for each of the 256 8-bit numbers. The input byte, or the starting byte is used as the index for the table and then the value at that index is then the substitution value. For the reversal of an S-box there must be an inverse S-box that must stay synced with the S-box for encryption and decryption to function properly. [11][(Vaudenay, 2006)]

Table 1: XOR Truth Table

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

1.2.2 Secret Keys

A key is a secret piece of information that is what controls a cryptographic system. In the encryption of system, the key specifies the transformation from plaintext into ciphertext or vice versa during the decryption process. Keys have already been

discussed above with block ciphers and stream ciphers and where they fit in with each. Basically a secret key transforms a file into something resembling random noise. It maps the message onto the ciphertext. One algorithm can encrypt the same thing many different ways using distinct keys. [12][(Amit Parnerkar, 2003)]

The key size itself can range from small and easily breakable to a one time pad which is, given that the key is not revealed, one hundred percent secret. The actual protection of a key, assuming it is not a poor key and easily broken, is based on people keeping the key safe. This means not writing it on one's desk and simply leaving it out in the open for all to see, or telling someone what it is, or emailing it out. These are all examples of what not to do and how keys are generally stolen. Keys are still easier to keep secret then the actual algorithms which are reverse engineered or published for all to see. [12][(Amit Parnerkar, 2003)]

1.3 Description of Enigma and its Inner Workings

This section describes the inner workings of the physical Enigma machine to get a better idea of how it actually functions in the underlying software.

The Enigma Cipher Machine consisted of five variable components:

1.3.1 The Plugboard

(1.) A plugboard which could contain from zero to thirteen dual wired cables:

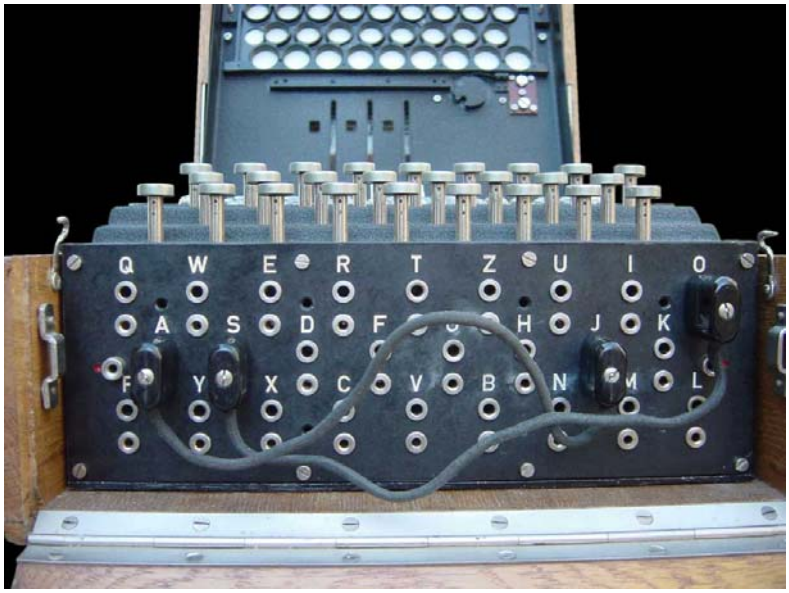


Figure 3: The plugboard of an Enigma machine, showing two pairs of letters swapped: S-O and J-A. [13]

The plugboard contributed greatly to the strength of the Enigma machine, far greater then adding an extra rotor would. Without a plugboard the Enigma ciphers created by these machines was generally easily defeated by hand. The machine operated with a cable placed onto the plugboard connecting letters up in pairs, a.l.a. S and O. These were called steckered[1] pairs and this would cause the letters to be swapped (similar to a substitution cipher) before and after the main rotor scrambling unit. In the case above, the signal sent to S would be diverted to O. Up to thirteen of these might be used at a single time. [8] [(Miller, 2008)]

Current was sent from the keyboard through the plugboard and would then proceed to the entry-rotor or Eintrittswalze. Each letter has two jacks. A plug would have a top and bottom connector that would disconnect the upper jack from the keyboard and

[1]Steckered is the German word meaning Plugged.

the lower jack to the entry-rotor of the specified letter. The other end of the plug would be connected to a separate letter which would thereby switch the connections of the two letters. [8] [(Miller, 2008)]

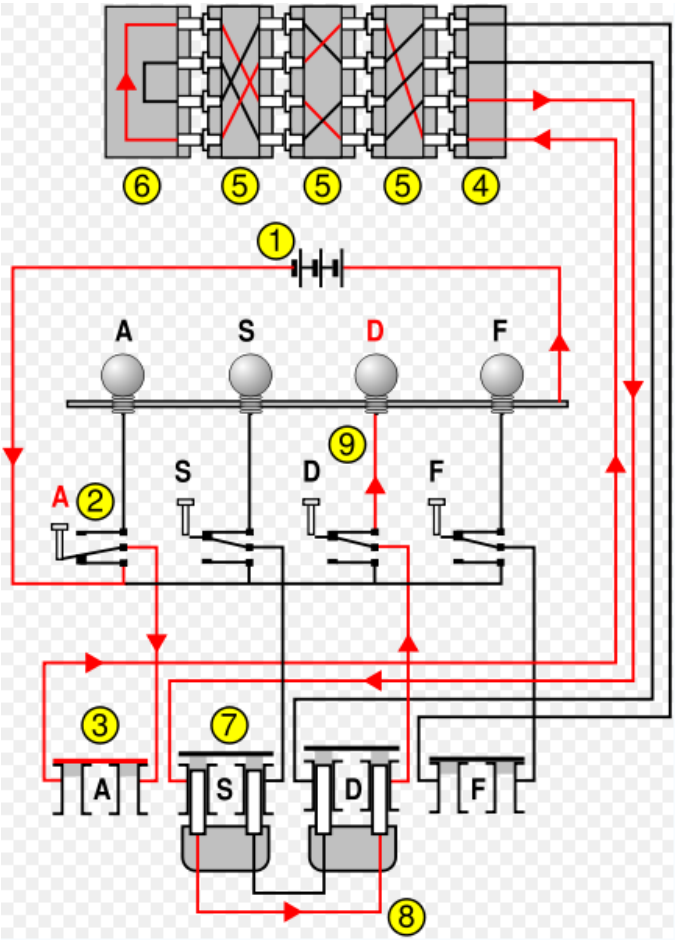


Figure 4: Schematic and Wiring Diagram of Mechanical Switches Involved in Encoding [14]

1.3.2 The Rotors, Serrations, and Ringstellung

(2.) Three left to right ordered rotors which wired 26 input to 26 output contact points on alternate faces of the disc: [8] [(Miller, 2008)]

(3.) Twenty-Six serrations around the outside of the rotors that allowed the operator to specify an initial rotational position for said rotors: [8] [(Miller, 2008)]

(4.) A ringstellung, a ring setting on each rotor controlled the rotational behavior of the rotor by means of a notch: [8] [(Miller, 2008)]

The rotors (German: walzen) were about 10cm in diameter and had a series of spring-loaded pins on one face arranged in a circle. On the opposite side of this were a corresponding number of circular electrical contacts. The pins and contacts represent the alphabet, usually the 26 letters of the English alphabet. When side-by-side the pins of one rotor rest against the contacts of a neighboring rotor forming an electrical connection. A set of 26 wires connects the contacts on one side in a complex pattern to the contacts on the other side, differing for every rotor. [8] [(Miller, 2008)]

Basically a rotor on its own does a simple substitution encryption. An example of this would be S would be wired to C from one side to the other. The strength comes from using three or more rotors in series with the regular movements of the rotors adding greater strength. [8] [(Miller, 2008)]

A rotor can be turned by the operator by hand to one of 26 positions. For the operators sake, each rotor has a letter ring attached to the outside of the disk which can be seen through a window indicating the position of the rotor to the operator. The ability to adjust the alphabet ring to the core wiring was added later to the version in which this thesis is concerned. The position of the ring is known as the Ringstellung. [8] [(Miller, 2008)]

The rotors each have a notch or multiple notches that are used to control their stepping. When the Enigma was first issued there were only three rotors available. In 1938 this changed to five, however, only three were chosen of those five to be placed in the actual enigma. These had Roman numerals, I, II, III, IV, and V to distinguish them

but this ultimately allowed for two separate attack methods to work against it. The Naval version added VI, VII, and VIII while it also allowed for a fourth rotor within the actual machine, however it did not replace the reflector. The problem with the fourth was that it did not step, it could only be manually set to one of 26 positions. [8] [(Miller, 2008)]

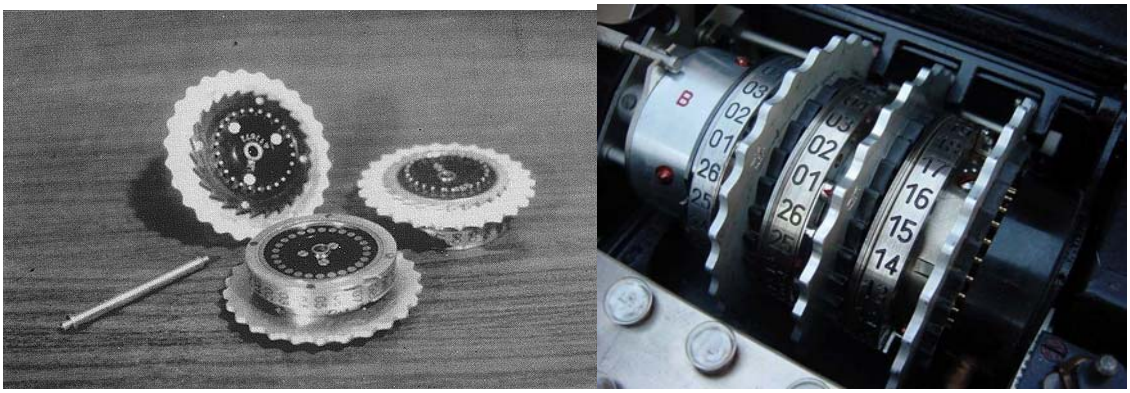


Figure 5: (left) Three rotors and the shaft on which they were placed when in use (right) Stack of rotors inside the enigma consisting of 3 rotors and the Umkehrwalze-B (the reflector

1.3.3 The Reflector

(5.) The last piece in the chain, a reflector, half-rotor to send the input back across the rotors: [8] [(Miller, 2008)]

The last "rotor" in the line did not, in fact, rotate. It was called the reflector (Umkerwhalze, reversal rotor). This was a feature distinctive solely to the Enigma in the age of the rotor encryption machines. The reflector connected up the last rotor in pairs, reflecting current back through a different route. The reflection was there to perform involution. Involution is a mathematical term for a function that is its own inverse. such that:

$f(f(x)) = x$ for all x in the domain of f

This ended up meaning that encryption was the same as decryption. A problem created by the reflector was that no letter would ever be encrypted to itself. This would later be exploited by the codebreakers at Bletchley Park. In some versions of the enigma the reflector would step as encryption was occurring, in others it did not. In any case the reflector had twenty-six positions. [8] [(Miller, 2008)]

Figure is a good visualization of just how enigma works.

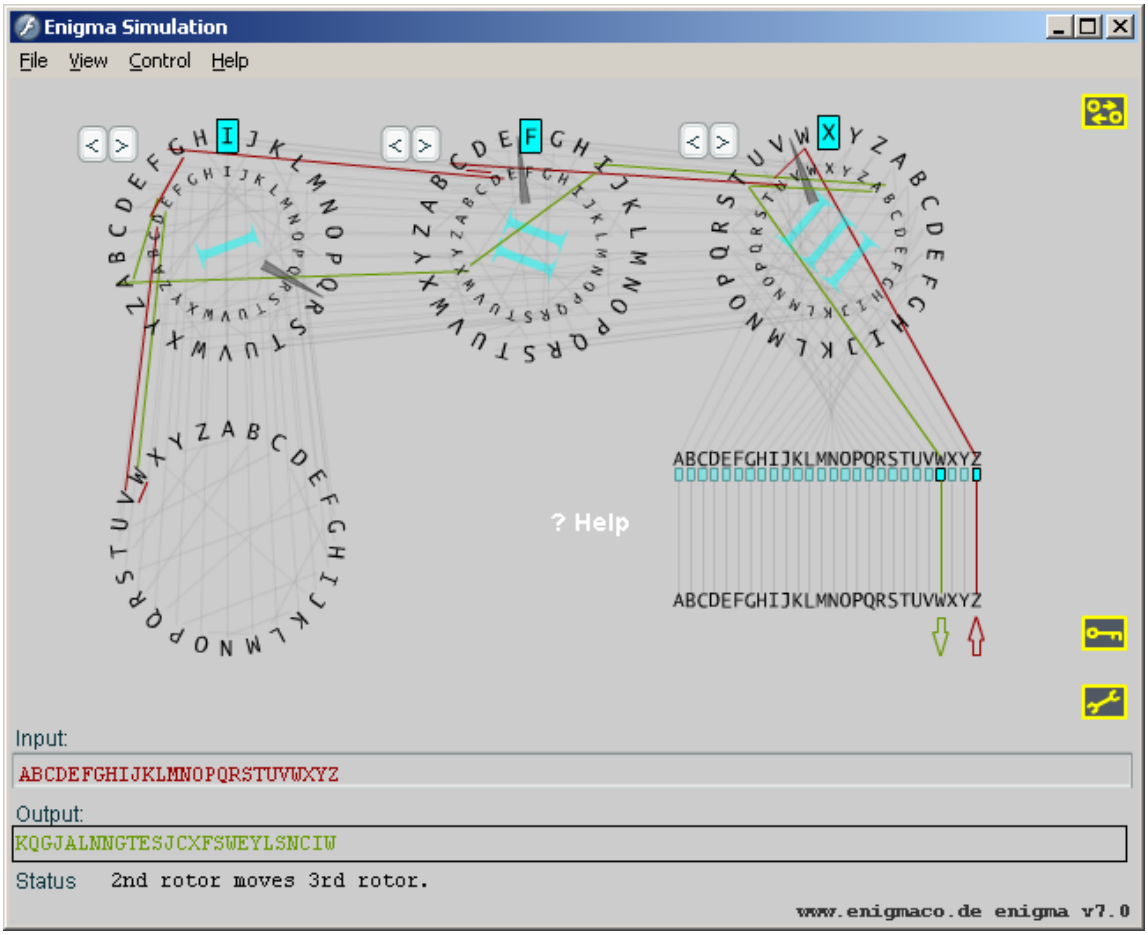


Figure 6: Example of the full workings of Enigma from Cryptool, shows the full alphabet being encoded

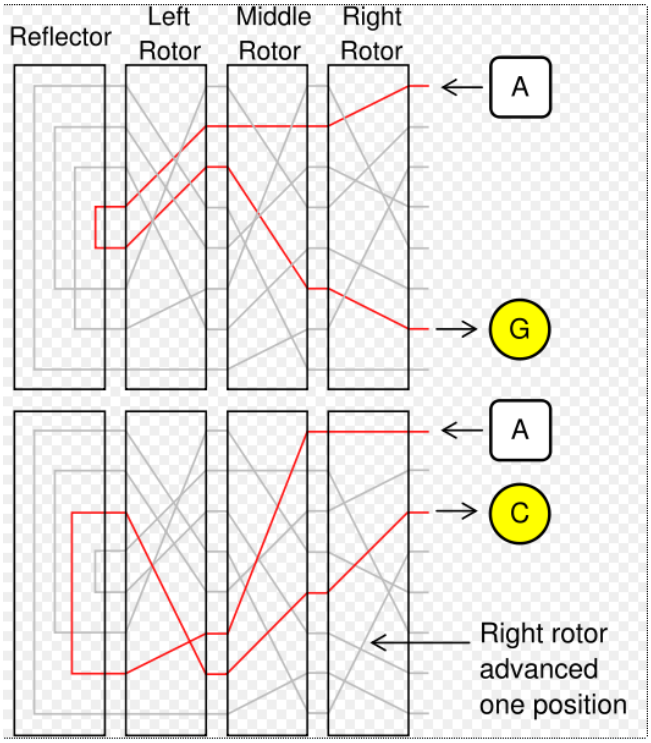


Figure 7: Scrambling action of enigma shown for two consecutive letters, the greyed out lines represent other possible circuits [15]

As can be seen, all of the lines are physically connected. The plugboard is active, however no letters are turned off. After all 26 letters have been entered, the rotor has made a full turn and thus the second rotor turns. This is a very basic example with the starting position of the third rotor beginning at A, had it begun at say M, it would have turned at Z and then gone through another 13 of the second rotor. The reflector can be seen as sending the letter next to it back through the algorithm.

1.3.4 Mathematical Description of Enigma

The transformation of each letter in Enigma can be specified as a production of permutations.

Assuming a three-rotor model:

E: Encryption
ρ: cyclic permutation
P: plugboard transformation

U: reflector
L,*M*,*R*: Left, Middle, Right Rotors

The period of a three rotor model would be this: 26 x 25 x 26 = 16,900. The reason it is not 26 x 26 x 26 is due to the double stepping of the second rotor.

[16][(Vaudenay, 2006)]

$$E = P*R*M*L*U*(L^{-1})(M^{-1})(R^{-1})(P^{-1})$$

After each key is pressed, the rotors turn. This changes the transformation. From that is derived the formula:

$$E = P(\rho^{i}R\rho^{-i})(\rho^{j}M\rho^{-j})(\rho^{k}L\rho^{-k})U(\rho^{k}L^{-1}\rho^{-k})(\rho^{j}M^{-1}\rho^{-j})(\rho^{i}R^{-1}\rho^{-i})P^{-1}$$

[16][(Vaudenay, 2006)]

2. Software

2.1 Introduction to Enigma and Changes Being Made

This algorithm is being named Enigma Phoenix. Why Enigma Phoenix, one might ask? In ancient mythology, a Phoenix is a bird that dies in flames and is reborn from the ashes. One could say that the Enigma most certainly died in flames at the hands of good and will be reborn from those ashes in the same hands.

Firstly, it is known that the Enigma cipher implemented a polyalphabetic substitution cipher. Basically a polyalphabetic substitution cipher is any substitution cipher that uses multiple substitution alphabets. That is, the letters that are being substituted as it is encrypted are changed based on a shifting set of alpha, numeric, and/or character sets. The Enigma cipher is also a stream cipher. Enigma is also a cipher whose encryption and decryption are the same. Thus through software only one function is needed to perform both.

The Enigma Cipher has been discussed above including its holes, its vulnerabilities, and all of its weaknesses. Even into the 2000s there have been attempts to break some of the original encrypted messages. There are obviously some flaws in the original model, but it was still an exceptionally secure model for the time and continues to show its strengths.

The software being developed here is set to expound upon that principle. Basically the idea is to take the original workings: the mathematics the algorithms, and the basic set of rules; and from there the basic set of rules will be modified. These modifications are intended to update the original Enigma with current technologies and knowledge gained from 75 years of progress.

From this knowledge what this thesis has set out to do is to modify the Enigma cipher in such a way that the base rules: the rotors, the reflector, the plugboards, all of this stays in use. What changes is their actual use. While the rotors still turn in a similar manner (though simulated), they turn based on a new Galois Field function. Also, the second rotor which was normally notched and thus double stepped, no longer double steps. What that function is and how it will actually work will be discussed below. The reflector works exactly the same way it did in the Enigma cipher. It simply adds another layer of diffusion to the cipher. The plugboard's actual functionality has been expanded more then it has been changed. There is still only one plugboard but the full range of the ASCII chart are its only limits.

2.2 Brief Description of AES and Comparison

The current model used as the standard is AES, the Advanced Encryption Standard. The Advanced Encryption Standard (AES), known as Rijndael, is a block cipher that was adopted fairly recently as an encryption standard used by the U.S. government. Its predecessor was the Data Encryption Standard, and much like its forbearer, it has been adopted internationally. After a five year standardization process by the National Institute of Standards and Technology (NIST) it was announced on November 25, 2001. It was initially created by two Belgian cryptographers, Joan Daemen and Vincent Rijmen and was hence submitted as Rijndael. [17][(Jamil, 2004)]

AES is not strictly Rijndael as Rijndael has a variable block length size as a possibility. Rijndaels format was chosen as the general method to be used for non-classified information by the NSA. In 2003 AES was approved for top secret information. AES itself has a fixed block size of 128 bits and key size of 128, 192, or 256 bits.

Because of this fixed block size AES operates based on a 4x4 array of bytes called the 'state'. The calculations for AES are done using Galois fields. [17][(Jamil, 2004)]

The mode of operation that is commonly used with AES is the Cipher-block Chaining (CBC). In this mode each block of plaintext is XORed with the previous ciphertext block before being encrypted. Each ciphertext block is thus dependent on all plaintext blocks processed to that point. To make each message unique, the aforementioned initialization vector is used. The main drawback of CBC is that the encryption is sequential and thus it must be padded to a multiple of the cipher block size. Even a 1 bit changed in the plaintext propagates itself throughout the ciphertext and a one bit change in the ciphertext corrupts the entire thing [18][(Vaudenay, 2006)].

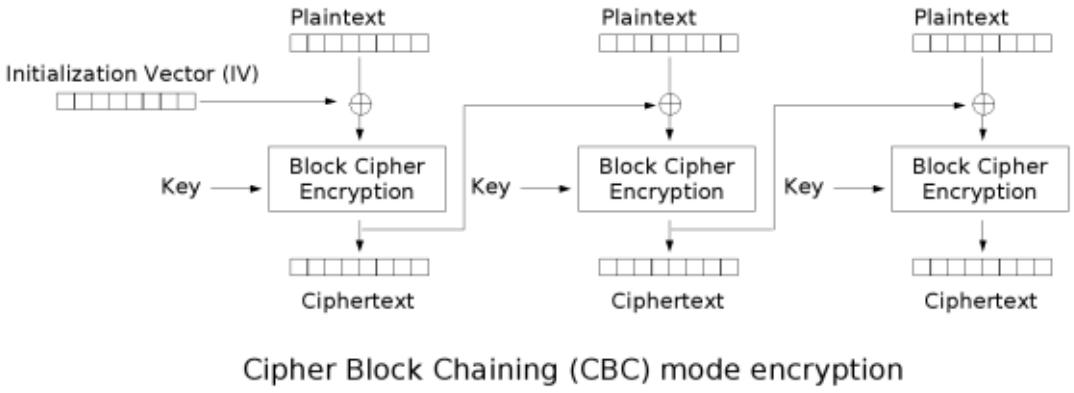


Figure 8: A view of the cipher-block chaining mode encryption.

AES uses a variable number of rounds which are fixed. At key of size 128, 10 rounds are used. At key of size 192, 12 rounds are used. At key of size 256, 14 rounds are used. During each round, four operations take place.

The first of these operations is SubBytes. In this, every byte in the state is replaced by another using 8-bit Rijndael S-boxes. These S-boxes are derived from the multiplicative over GF(2^8).

The seconds step is ShiftRow. In this every row in the 4x4 array is shifted a certain amount to the left. The first row is left unchanged in AES. In the second row, each is shifted by an offset of one, and then two for the third, etc. For 256 bit, the offset goes 1, 2, 4 etc.

The third step is the MixColumn. In this a linear transformation on the columns of the state occurs. It takes four bytes as input and outputs four bytes. This step helps provide diffusion for the cipher.

The fourth step is AddRoundKey. In this each byte of the state is combined with a round key. Each key is different based on Rijndael's key schedule. In the final round, MixColumn is not used and thus it has only three steps in the final round. Thus, AES is a fair ways different then DES but both are fairly complicated if looked at from above, however, when broken into pieces one can see exactly what is happening.

2.3 Stream Cipher Vs. Block Cipher

Stream ciphers tend to be designed to be quite efficient and exceptionally fast, much faster then a block cipher. As stated before, a stream cipher is a symmetric cipher where the plaintext bits are encrypted one at a time. A stream cipher is a fairly polar opposite approach to symmetric encryption to a block cipher (though they are both symmetric encryptions). Stream ciphers execute at a higher speed and have a lower hardware complexity than block ciphers. A stream cipher generates a 'keystream' (a sequence of bits used as a key) and the encryption is done by combining the keystream and the plaintext using a bitwise XOR. Either a synchronous (keystream is independent of the plain and ciphertexts) or a self-synchronizing (keystream depends on plain and ciphertexts) stream cipher can be used. Stream ciphers do, however, tend to be more

vulnerable to security flaws such as using the same starting state twice would make it almost completely open to successful attack [19][(Schneier, 1996)].

A block cipher is a form of symmetric encryption that transforms a fixed length block of plaintext into a block of ciphertext of the same length. Decryption is done by applying the reverse transformation to the ciphertext block using the same key. The fixed length of a block cipher is the block size. A block cipher provides a permuation of the set of all possible messages. The permutation effected is a secret since it is a function of the secret key. When a block cipher is used for a message of arbitrary length, modes of operation are used. To be useful the modes must be as efficient and as secure as the underlying cipher. Block ciphers and stream ciphers are generally fairly distinct. [20][(Schneier, 1996)]

However, the distinctions can blur. A block cipher, when used in certain modes acts almost exactly the same as a stream cipher. This takes place based on the users secret key.

2.4 Galois Field

This version of the Enigma cipher has been modified to use Galois field mathematics. The math behind the Galois field and the code behind it are somewhat disconnected. That is, an understanding of the math does not inherently translate to an understanding of the code and visa-versa. The method for the code chosen here was to create a separate file containing the Galois operations. This is as opposed to placing them within the main file or overloading the operators (albeit overloading the operators would be ever so slightly more efficient, but in the end the time spent doing so is not worth the minor increase). First it should be mentioned that the functions in galois.h are all uint8_t,

that is, an unsigned 8-bit integer. Addition and subtraction are exactly the same. They are both done with an XOR operation:

```
uint8_t gadd(uint8_t a, uint8_t b)
{
   return a ^ b;
}
uint8_t gsub(uint8_t a, uint8_t b)
{
   return a ^ b;
}
```

Multiplication is a bit more complicated. From [ ], below is a listed explanation of the mathematics behind the gmul{} function within the Galois header file using the numbers seven and three

*Take two eight-bit numbers, **a** and **b**, and an eight-bit product **p***

- *Set the product to zero.*
- *Make a copy of **a** and **b**, which we will simply call **a** and **b** in the rest of this algorithm*
- *Run the following loop eight times:*
    1. *If the low bit of **b** is set, exclusive or the product **p** by the value of **a***
    2. *Keep track of whether the high (eighth from left) bit of **a** is set to one*
    3. *Rotate **a** one bit to the left, discarding the high bit, and making the low bit have a value of zero*
    4. *If **a**'s hi bit had a value of one prior to this rotation, exclusive or **a** with the hexadecimal number `0x1b`*
    5. *Rotate **b** one bit to the right, discarding the low bit, and making the high (eighth from left) bit have a value of zero.*

*The product **p** now has the product of **a** and **b***

As for the exponentiation and logarithmic tables, they can be generated in the code included in this program. Instead however, for stability's sake and for efficiency, the tables have been hardcoded into the actual header file. This means that any logarithmic operations that are performed can be done using the tables and the exponential tables are simply the anti-logarithmic tables.

For division, a function gdiv was written that simply executes a (log(a)-log(b))mod255. The multiplicative inverse is also done using the lookup table to run a check and see what it is.

An expansion to Enigma would be to use the Galois fields in combination with s-boxes. The s-boxes would use the Galois field's multiplication and the inverse-multiplication functions. [21][(Trenholme, 2008)]

2.5 Description of Changes in Enigma

The code that is being written for this version of Enigma is somewhat extensible. That is, it has several components that can be changed and added upon. The first part has to do with the rotors. The choice that was made was to include a total of five rotors in this version. Within the realm of the ASCII table that is a total of 1099511627776 possibilities. This means that even in rather large messages the rotors will not spin back around. Of course, this leads to the problem that if one were to encrypt a document with the same key over and over again one would get essentially the same results. Should an attacker intercept the transmission of this document on multiple occasions they would more then likely be able to form an attack on the cryptographic system. The other possibility is encrypting different messages with the same key. This would yield a similar result, an attacker intercepting them would eventually find a pattern and break it.

Going back to the extensibility that was being discussed, it was mentioned that a total of five rotors are being used in this iteration. Five were selected as they would be cryptographically secure, and for academic purposes are more then enough. However, this program was designed so that with the change of a number that could be extended to as many rotors as are desired or may be deemed necessary.

The main point of this thesis was not to simply modify the original Enigma cipher. It was to make it cryptographically secure on modern computing systems. The holes in the original Enigma have been lightly touched upon already. These and more

will be touched upon later. For now it is simply needed to be understood that the original Enigma was not cryptographically secure. It had certain flaws that made it unsafe to use, especially on large messages (small messages made it difficult to break as there was not enough data to work with).

What this thesis is about is modifying of Enigma to include Galois mathematics. These are the same mathematics involved in making a current cryptographically secure algorithm, Rijndael's Advanced Encryption Standard, secure. Mostly, this is dealing with s-boxes. How this works is, a key is passed in through a function that generates the s-boxes. These s-boxes are then used to generate the rotors. The s-box generator includes the use of the galois multiplication and inverse-multiplication functions. It also includes an XOR with the generated MD5 key which makes it more secure.

The plugboard is an interesting device. It gave Enigma a great deal of its cryptographic strength. It was a fairly brilliant and simple idea on the behalf of its creators to create diffusion on a non-computing system. The plugboard in this version of Enigma adds a good deal of diffusion as well. However, simply turning off certain characters at random or at selected intervals is not really practical. What was again needed here was the use of the Galois Fields to add diffusion and allow for a certain amount of randomness while still being able to decrypt the message (if it were wholly random there would be no way to decrypt it).

The reflector has also been modified but only slightly. It now includes the full range of the ASCII table as well. This is, of course, a necessity. Without the reflector now having all 256 characters, Enigma would not work. As mentioned above, the process involves traveling through the rotors, and then the connected lines from the rotors are

then translated to the reflector (this is all after the plugboard has been applied). After this it is sent back through the rotors using the required characters.

Essentially this is a modernizing of the original Enigma. The question is, what kind of difference will this make? Can it rival today's modern block ciphers? Will it be efficient and does it even do what its supposed to? That's really what this thesis is proposing to find out. It's a simulation of a modernized Enigma.

2.6 Procedure

The purpose of this thesis is to propose a secure software version of the mechanical encryption device, the Enigma machine, that implements the following:

1. A stream cipher based on the Enigma Machine, a polyalphabetic cipher
2. Add a Galois Field function to the cipher that generates the rotors through S-box generation and using the multiplicative properties of Galois Fields.
3. Add functionality to encrypt and decrypt any binary files
4. Add the full functionality of the ASCII table to it (as opposed to the original use of the 26 English letters)
5. Use a 128 bit key to encrypt the data through an MD5 sum and use this key with the S-box function to make it more secure.
6. Make it secure in such a way that knowing the passphrase is the only way to decrypt the file

2.6.1 Description of Software

There are four main functions behind the Enigma Phoenix cipher. These four main functions are the same functions that any basic Enigma cipher would need. The first of these functions is the init_enigma(). What this does is to allocate memory for the

rotors. It also sets the default value of the rotors to zero. Beyond that it calls and initializes the rotors themselves. It also initializes the reflectors and runs through the rotor lookup.

The logic works like this:

```
Set Number of Rotors = number of rotors desired
Initialize Rotor Memory Space = array based on number of rotors
        For (0 to NumRotors)
                Initialize all 256 (ASCII) values of Each rotor

Initialize Reverse Rotors Memory Space = array based on number of rotors
        For (0 to NumRotors)
                Initialize all 256 (ASCII) values of Each rotor
Create galois object
Initialize values from 1-256
        Pass values into rotors which are generated from S-box function

Set default Values of Positions to 0

Initialize the plugboard (by default all values are 1:1)

Initialize Rotors (based on Galois function)
```

The next function is the rotor_lookup() function. This does essentially what its nomenclature suggest, it checks the rotor's arrays for the value at each of the 256 characters in each individual rotor (a.l.a. if the letter a is passed in, it checks on rotor one to see where that a would be, for example, it would be @ on the rotor).

The Logic for Rotator Lookup can be seen in Figure :

```
For the values 0-255:

        If the rotor is set to the value passed in,

        Return it
```

Char_do_enigma() does a lot of the heavy lifting, it controls the initial and continuing rotations as well as most of the other functions.

The logic looks like this:

Rotate first rotor by 1

check if any of the other rotors need to be rotated

Go through the plugboard

Go through the first rotor

Then go through the rest of the rotors

Go through the reflectors

Then go back through the plugboard

Audit_rotors() checks to see if a rotor has made a full revolution and increments the next rotor.

The logic looks like this:

Make sure current position isn't greater then 255

    If it is, set it back to 0

On the last rotor, check to make sure the current position isn't greater than the number of rotors.

    Increment position

    Call audit_rotors

FileSize() is simply a function that gets the size of the file. File_do_cipher() calls init_enigma(), reads each character from file, then proceeds to encrypt/decrypt it with char_encrypt(). Lastly it writes it back to new file.

The logic looks like this:

```
Pass the key to init_enigma()
Get the file size
Open the file for reading and writing
Check to see if there's an error
Run through to the size of the file
        Pass characters through char_do_enigma()
```

Those are the main functions behind Enigma Phoenix. Besides these the other main function would be the sbox() function. This generates the s-box for the rotors.

The logic looks like this:

```
Pass in the character from the rotor
Create a copy of the multiplicative inverse
Shift it with circular rotates to the left
XOR it with the MD5 sum key
Return the value
```

3. Practical Applications

Will this be useful? That's the main question this thesis sets out to answer. That answer exists and is listed below. What this section is set about to discuss are the practical applications assuming the usefulness of the algorithm.

Even today, Enigma plays an important role. When it was learned in the 1970s the importance that deciphering the Enigma had played in the Allies winning the war, public interest was piqued. Even today, there is a project to create a complete simulation on handheld devices called AR-ENIGMA. This will be used to demonstrate the Enigma's capabilities for Cryptography classes as well as for major museums (such as the Museum Of Natural History or the Intelligence Museum in Baltimore, Md.). It is a server-client product where the server sends out the 3D representation of the Enigma to the handheld and the interactions on the handheld are sent back to the server. [22][(Volker Paelke, 2002)]

Certainly for teaching purposes Enigma is an exceptional example to show both due to its power and due to its deficiencies.

On an application to application basis the question is not what can be done with this. The question is more whether or not this is cryptographically secure, that is, is it capable of keeping data confidential. Is it resistant to a brute force attack? If an attacker is attempting to break it, can it be made to where it is not worth the amount of time they would have to put in versus the value of the data.

Most, if not all of these questions are answered in the Results section below. Once the dust has cleared it will be shown whether or not Enigma Phoenix is simply an exercise in academic curiosity or whether it is functionally a useful cipher.

3.1 Previous Research

There hasn't been a vast amount of new research done on the Enigma Cipher. This is mostly because it is considered antiquated, something for the history books, something to be taught as a failure in classrooms around the world. Why is that? Because it was a failure, it failed at keeping the enemy's secrets, for which we are all exceptionally happy. It failed for one main reason though.

The premise behind the original Enigma cipher is security through of obscurity. The premise behind security through obscurity is that inviolability is a consequence of the enigmatic. The Enigma cipher was assumed to be inviolate due to being enigmatic, that is, due to its hidden complexity. It was assumed to be secure because it was assumed that all of the information needed to break it was hidden. This is what is called faith based security.

Dating back as far as the 1880s, Auguste Kerckhoffs proposed that no cryptographic system that claims to be secure should be predicated solely upon an assumption that people would be unable to figure out its basic functions. The emphasis should instead be predicated upon robustness of the procedure and key strength. The German war machine failed to understand the inherent weaknesses in security through obscurity. This tends to speak in favor of the robustness in open source software.

Most of the implementations of Enigma have been very similar to AR-ENIGMA, that is, they are for demonstration purposes. There are many opengl java based or C# based applications designed to emulate the Enigma machine.

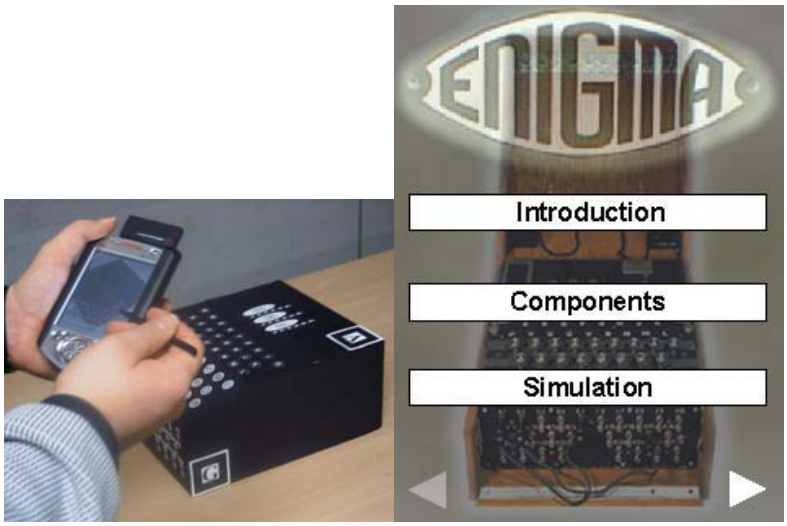


Figure 9: AR-ENIGMA running on pocket PC based handheld as well as a screenshot of the menu to a simulation [22]

Figure 10: Simulation of Enigma cipher running in openGL

4. Cryptanalysis

4.1 Cryptanalysis of the Original Enigma Cipher

The Enigma cipher was originally designed to defeat cryptanalysts by continually changing the substitution alphabet. As mentioned before it implemented a polyalphabetic substitution cipher. With single-notch rotors, the period of the machine was 16,900, or 26x25x26.

However, as mentioned above the Enigma is not a base rotor machine. Enigma added other possibilities. It added a variable starting position, a variable alphabet ring to each rotor, a reflector, and a plugboard. Despite the complexity of the machine, the key was very simple to communicate as one could send what rotors to use, what order, connections, and starting positions. [23][(Kahn, 1991)]

At the time when the machine's use was prevalent, the fact that encipherment and decipherment were the same was considered an advantage. They were the same in that, if the users had the same machine set up (same rotor choices, etc), then the decipherment process was the same as the encipherment process. The changing of the configuration of the machines changed the key. The changing of the configuration, and thus the changing of the keys was set up on a monthly, then weekly, then daily schedule [23][(Kahn, 1991)].

The different models of the Enigma machine obviously provided different levels of security. The use of a plugboard (stecker) substantially increased the level of security. Without a plugboard, Enigma was able to be broken by hand methods. With the plugboard Enigma required machines to actually be able to break it [23][(Kahn, 1991)].

Enigma had several cryptographic holes that proved exceptionally useful to cryptanalysts. One of the main flaws was that, save for models A and B which did not have a reflector, the Enigma machines could not encrypt a letter to itself. This allowed for cribs – short sections of known plaintext in the ciphertext – to be created. Another property of Enigma was that it was self-reciprocal, that is, encryption was the same as decryption. This limited the amount of scrambling it could do (at the time, software allows for a much more flexible amount of scrambling in separate components) [23][(Kahn, 1991)].

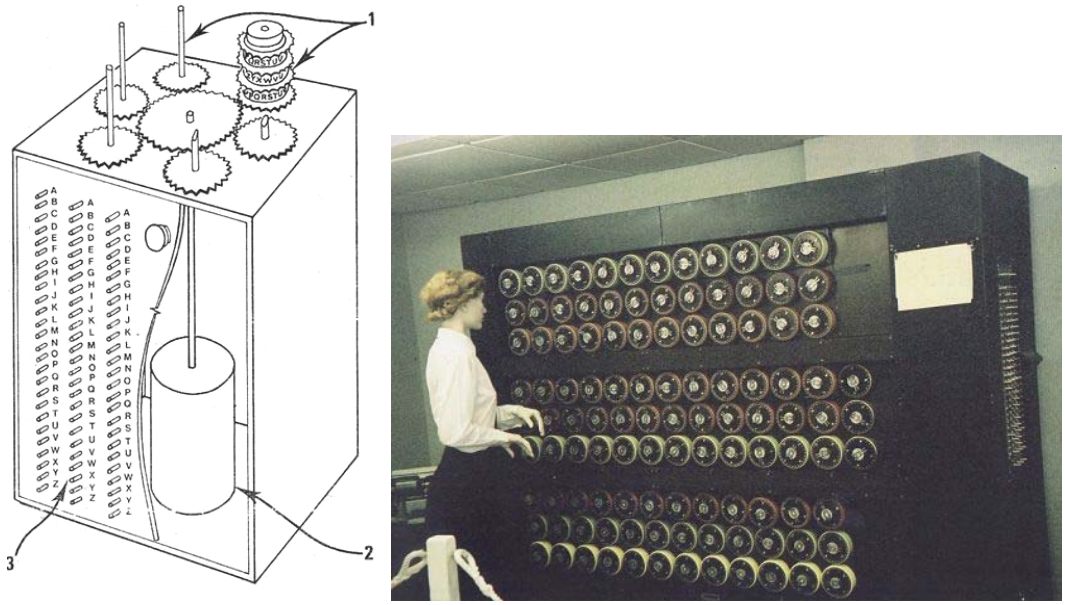


Figure 11: Bombe used for breaking the Enigma Cipher

4.2 General Cryptanalysis Techniques

There are currently and have been many attempts at analyzing the strength of a cipher. The main method that many are familiar with is ensuring an adequate key space. There are a lot of areas where a cipher can fall apart, be it from a weakness in the algorithm's design or the implementation thereof (software and hardware). The first and most basic of these tests is a data histogram. From the processing in any cipher there should be enough data to produce a uniform frequency histogram for all the bytes. This is irrespective of the length, verboseness, or other properties of the plaintext file. If this is not the case then a simple language attack can be made against it which is why the old Caesar ciphers are very easy to break (and for other reasons). Letters in the English language and their base limit of 26 make it easy to break poor ciphers. If the base graph is uniform it will be a step towards showing that the cipher is not trivial to crack. [4][(Schneier, 1996)]

There's several major properties that must be discussed, two of the more major are the Index of Coincidence (IOC) and the Critical Avalanche Effect (CAE). The CAE says that for one bit in the input byte at least fifty percent of the bits in the output byte should change [24][(Kari, 1992)]. In AES this is done through the diffusion steps that have been mentioned with ShiftRow and MixColumn. In Enigma this is done through the rotors and through the plugboard. The MixColumn in AES performs matrix multiplication which is done to ensure that every input byte affects four bytes of the output. The plugboard changes one byte over and then affects the following five as it goes through the rotors. This combined with the S-boxes for AES makes it meet the solid

criterion of CAE. The Enigma also meets this soundly as the infinite expandability of the rotors makes it very capable but not as elegant.

The IOC is the property of placing two texts next to one another and counting the number of times identical letters appear in the same position in both texts, figure 12 [25] shows the formula for calculating the IOC. Due to the randomness of both AES and Enigma Phoenix they each have very low IOC which is very good. In the original Enigma, the IOC was very high as it only used the English alphabet to encrypt, now that the full ASCII table is used it is much lower. These two statistics show that there is a high possibility, if implemented intelligently that the algorithms will be secure.

Table 2: Example of IOC calculation on a basic Substitution Cipher

Pair	Probability
AA	18.75%
BB	18.75%
AB	56.25%
BA	6.25%

One of the problems of AES is that according to some authors [4] [9] the S-boxes generated in the algorithm are static. This would allow attackers to modify the binary and discover the secret key and plaintext. Most implementations with static S-boxes are vulnerable to blanking. This would result, as mentioned, in a discovery of the key.

Since the static S-boxes are stored in a binary, with AES the attacker could blank them with zeroes and due to the XORing with an entry from the key, the key would be outputted in the cipher text. This is generally defended against by protecting the binary in the operating system or other means. Recently, with dynamic S-boxes, they are able to defend against these attacks even with direct access to the binary files.

A weakness with Enigma, and even Enigma Phoenix, is that encoding different plaintexts with the same key could result in an IOC attack that would work. It depends on the varying sizes of the plaintext but with the same rotors (based on the key) used over again it would be possible that this could be broken. The problem with this method is that it would take a large, vested amount of time and testing to do which most would not be willing to place into it. It would depend on the key and length of this and the plaintext which is unpredictable. The only way to test this for Enigma Phoenix would be to place it into practical use which isn't possible at the moment and thus cannot be tested for, this is something that would be a recommendation for further study.

$$\mathbf{IC} = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)/c}$$

Figure 12: Formula For Calculating IOC

4.3 Techniques Used For Cryptographic Analysis in Enigma

The most basic method for testing a cryptographic algorithm is the data histogram. After the processing of a file there should be adequate scrambling (through confusion and diffusion) of the data to produce a uniform frequency histogram for all byte values to produce a uniform frequency histogram for all the byte values (256).

There are a great number of tests and techniques that can be performed to analyze the performance of a cipher. Several of the main methods are used here as both tools for comparison and tools for individual testing of each cipher. As has been stated two ciphers are being compared, one being AES, and the other being this thesis version of Enigma, Enigma Phoenix.

A major measure used today is Entropy. That is, Shannon entropy also known as information entropy. This is the measure of the amount of information contained in a random variable. The amount of information in a message is the minimum number of bits needed to encode all possible meanings of that message. This is assuming all messages are equally likely. A constant pattern has an entropy of zero since it requires zero bits to transfer such a message. Formally, the amount of information in a message *M* is measured by the entropy of a message, denoted by H(*M*). [26][(Schneier, 1996)]

Generally the entropy of a message measured in bits is $\log_2 n$, where n is the number of possible meanings, this assumes that each meaning is equally likely. The entropy of any given message also measures its uncertainty, that is, the number of plaintext bits needed to be recovered when the message is scrambled in ciphertext in order to learn the plaintext.

Conditional entropy is often used as a measure of secrecy. It provides a measure of similarity between two discrete random variables. These can be used to determine whether the two variables are independent or whether they are dependent - and to what extent - on each other. If the ciphertext were compared to the plaintext and the conditional entropy were zero, this would mean that the cipher text provides all of the necessary information to break it, or undo the encryption. To have complete secrecy the conditional entropy between the plaintext and ciphertext would be equal. This would only be the case for a one-time pad [26][(Schneier, 1996)]. Figure 13 [27] and Figure 14 [28] show the formulas for entropy and conditional entropy respectively.

$$H(X) = \mathrm{E}(I(X)) = \sum_{i=1}^{n} p(x_i) \log_2 (1/p(x_i)) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Figure 13: H(x) being Entropy as well as Conditional Entropy

$$\begin{aligned} H(Y|X) &\stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} p(x)\, H(Y|X=x) \\ &= -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x)\ \log\ p(y|x) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y)\ \log\ p(y|x) \\ &= -E_{p(x,y)} \log\ p(Y|X). \end{aligned}$$

Figure 14: Conditional Entropy with H being the probability of Y given X

5. Statistical Analysis on Modified Enigma Phoenix and AES

This section is dedicated to the analysis of the Enigma Phoenix cipher as well and in comparison to the AES cipher. The techniques being used will be described in this section. The main tool being used to do the analysis is called Cryptool, it is a very useful tool for analysis data files, hex files, text files, etc. It will do its own base histogram as well as calculating entropy. This is necessary to avoid the first order language attacks that can break simple ciphers like Caesar and so forth.

5.1 Analysis of AES and Plaintext

When looking at the statistics and analysis of a cipher there are several main categories that must be taken into account. It should be noted that any cipher using s-boxes without the diffusion steps is just as cryptographically insecure as the most basic Affine shift-cipher. The version of AES that is being tested is the commonly used, and thought of as secure government AES with 128-bit strength.

Key Used to Encrypt: AABB1A1C1D2A9CD726

Entropy for unencrypted Moby Dick: 4.55

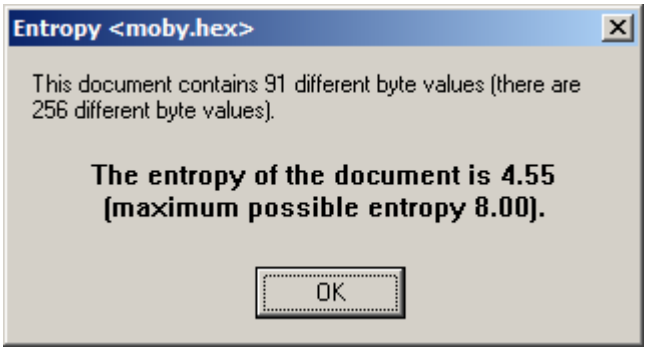


Figure 15: Entropy of Moby Dick unencrypted (converted to a hex file so that all 256 values are accounted for)

Entropy for Moby dick encrypted with true AES: 7.99

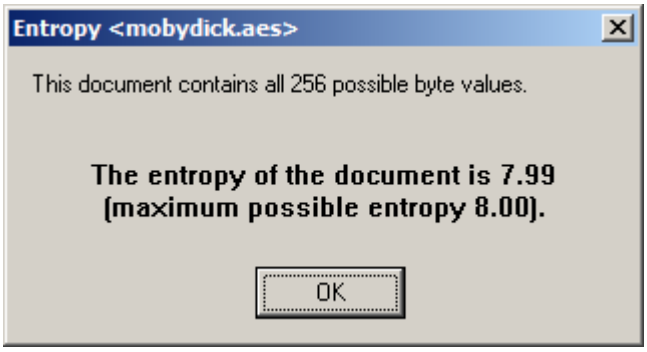


Figure 16: Entropy for Moby Dick encrypted with 128bit AES

Above, in Figure 15 and Figure 16 one can see the entropies for Moby Dick both encrypted as well as unencrypted. The unencrypted value is only as high as it is due to the exceptional length of the text being used. Despite it's somewhat higher than average entropy for a plaintext document; the encrypted ciphertext is still far higher and is about as close to eight as can be.

Related to this is the periodicity. When measuring the periodicity one wants to make sure that the file itself doesn't have repeating cycles. This is generally something that will be seen in the plaintext but should never be seen in a ciphertext that is claimed to be secret. If there are patterns then it is easier for an attacker to find some sort of attack vector. Figure 18 shows the periodicity of Moby Dick as a plaintext file. Figure 17 shows that there is no periodicity in the AES encrypted file.

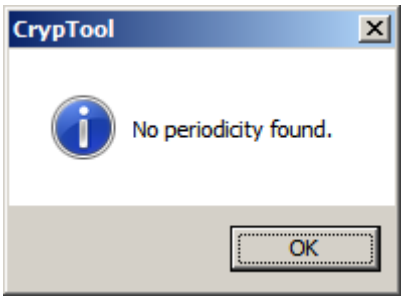


Figure 17: Analysis of the periodicity on the AES encrypted Moby Dick turns up no results

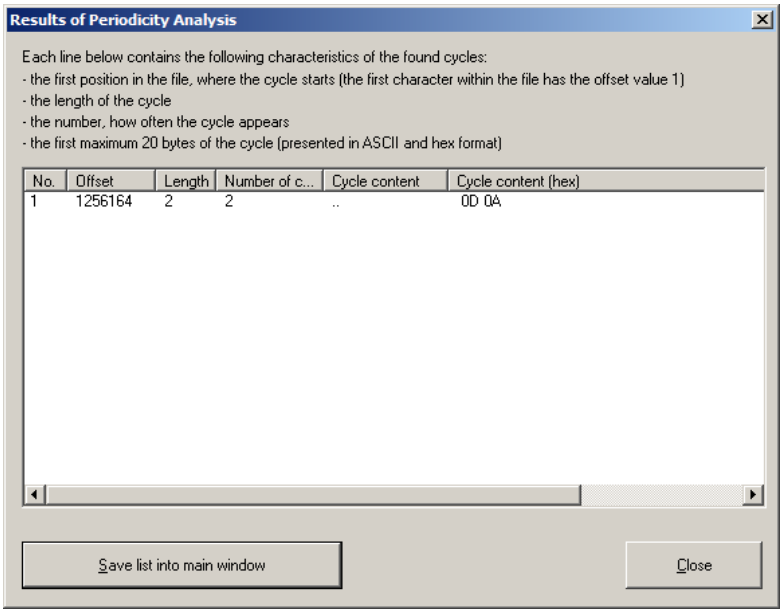


Figure 18: Periodicity of Moby Dick plaintext file

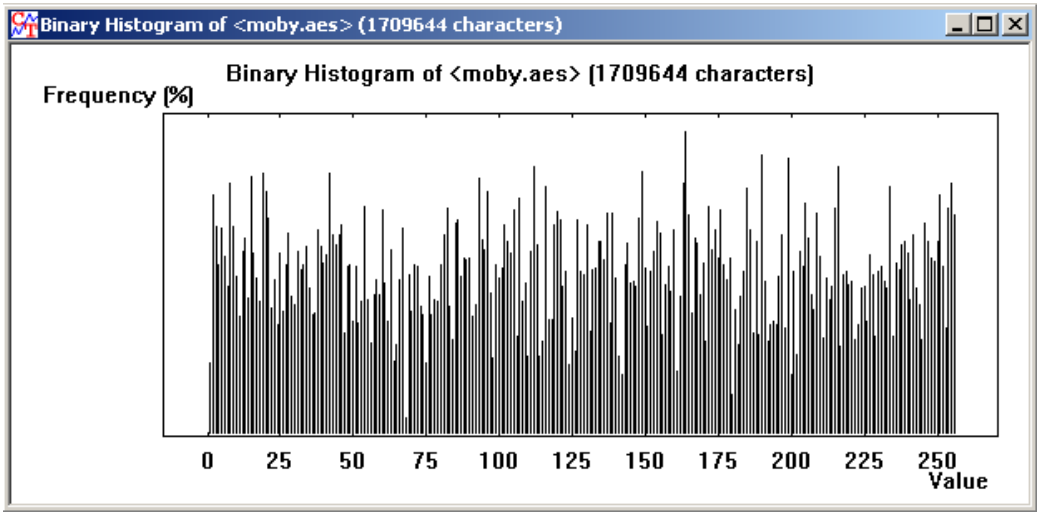


Figure 19: Binary Histogram of Moby Dick encrypted with AES

Above, in Figure 19: Binary Histogram of Moby Dick encrypted with AES is the analysis of the frequency of each ascii character in the document. As can be seen the frequency is very distributed which is what is wanted. This is a very good example of a securely encrypted document using AES. A better way to say this would be to point out that the histogram is uniform, which is what is to be expected. If a cipher is to have a chance of being better then the simplest affine cipher then the histogram needs to be

uniform. Figure 20 shows the histogram for Moby Dick when it is unencrypted and the difference is obvious.

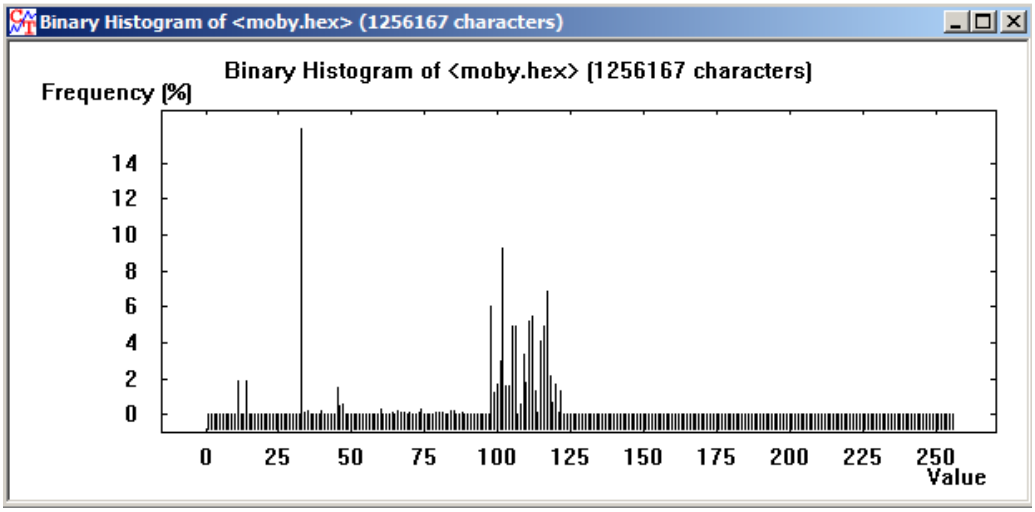


Figure 20: Binary Histogram for Moby Dick unencrypted

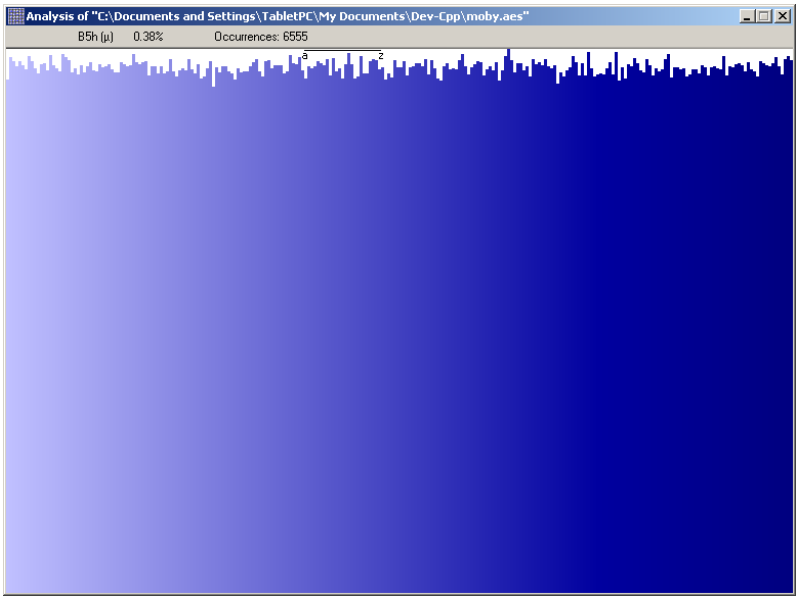


Figure 21: Base Histogram of Value Occurences as well as Total Occurences

Above is another example using the tool WinHex to determine the occurrence of each character. The y-axis represents the frequency, the x-axis represents the character in the ASCII table. This is a bit of a more complicated and closer analysis which is why it gets a better looking uniform, histogram, it doesn't have any actual numbering so it's

more difficult to interpret. Essentially though, it is another example of the uniformity of the distribution.

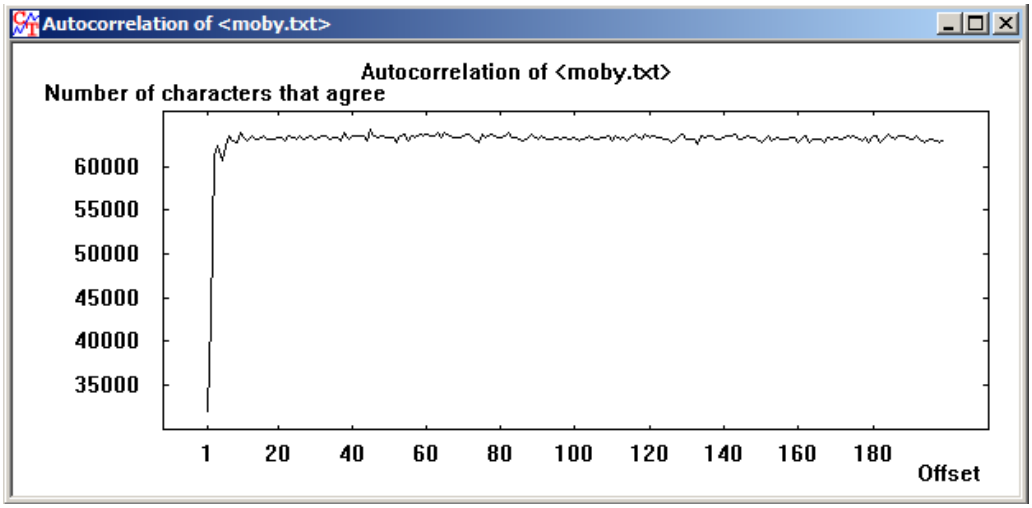


Figure 22: Autocorrelation of Moby Dick in plaintext

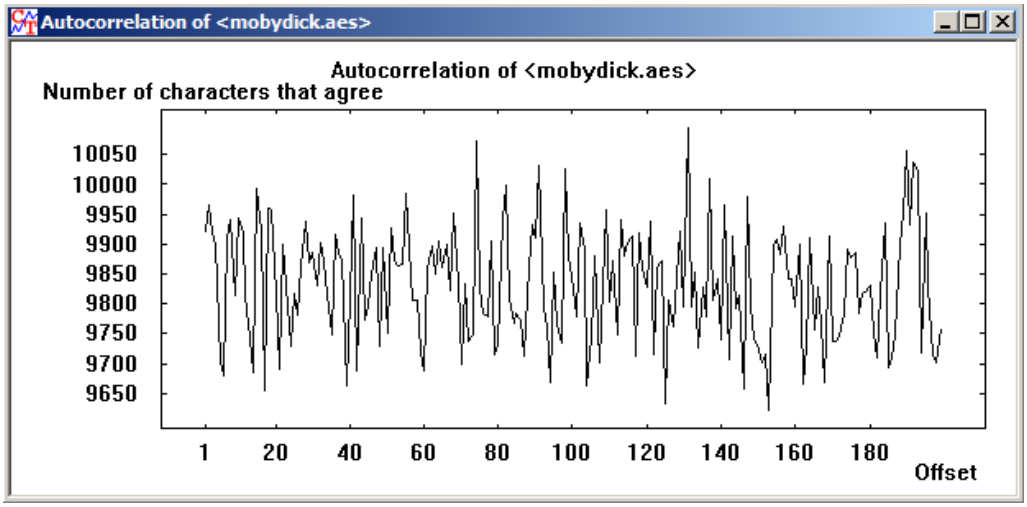


Figure 23: The autocorrelation of Moby Dick encrypted with AES

The autocorrelation is a mathematical tool for finding repeating patterns. It describes the correlation between the process at different points in time. As can be seen in Figure 22 and Figure 23 the difference between an encrypted file and the plaintext is vast. The plaintext has a large grouping of patterns which is why it shows as basically uniform

across the top. The ciphertext is all over the place, there aren't any underlying patterns that can be found.

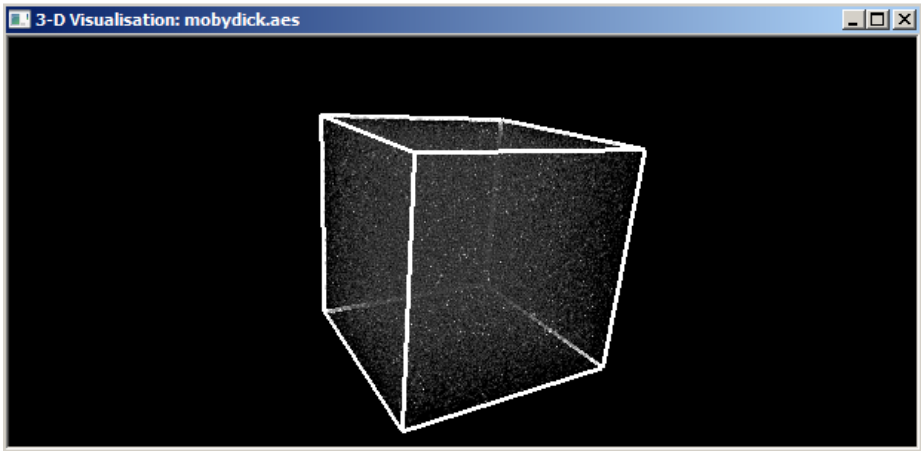


Figure 24: 3D visualization of the randomness in Moby Dick encrypted with AES

Figure 24 is a representation of the randomness of the encrypted Moby Dick. The inside of the cube represents the randomness in the file.

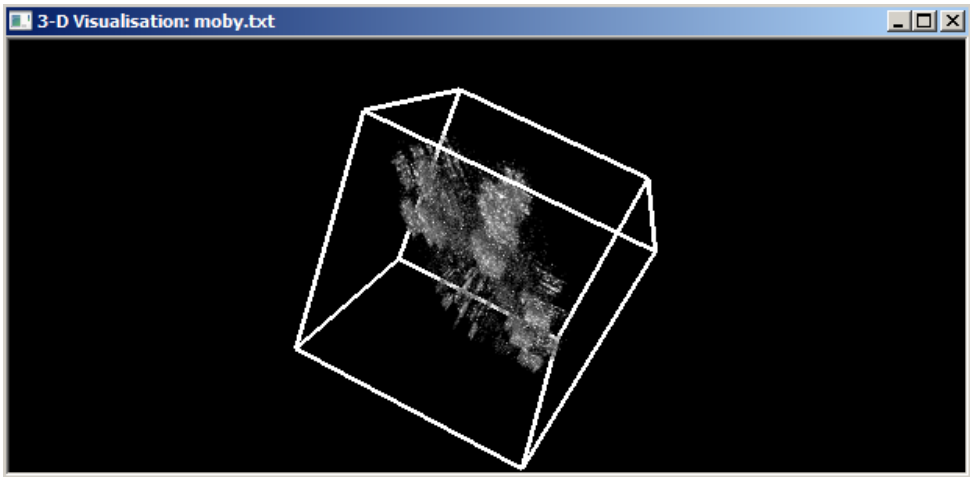


Figure 25: 3D representation of randomness in Moby Dick plaintext

Figure 25 shows the randomness of the plaintext file and the difference is instantly recognizable. The plaintext is nowhere nearly as distributed as the encrypted file.

Table 3: ASCII Chart of Moby Dick Encrypted with AES

Char			Occurence	%	Char			Occurence	%	Char			Occurence	%	Char			Occurence	%
[illegible]	0	.		0.38%	39	57	9	6663	0.39%	72	114	r	6561	0.38%	AB	171	«	6785	0.40%
01	1	.	6804	0.40%	3A	58	:	6639	0.39%	73	115	s	6819	0.40%	AC	172	¬	6713	0.39%
02	2	.	6751	0.39%	3B	59	;	6781	0.40%	74	116	t	6598	0.39%	AD	173		6746	0.39%
03	3	.	6689	0.39%	3C	60	<	6657	0.39%	75	117	u	6597	0.39%	AE	174	®	6699	0.39%
04	4	.	6749	0.39%	3D	61	=	6596	0.39%	76	118	v	6754	0.40%	AF	175	¯	6779	0.40%
05	5	.	6701	0.39%	3E	62	>	6714	0.39%	77	119	w	6777	0.40%	B0	176	°	6688	0.39%
06	6	.	6652	0.39%	3F	63	?	6529	0.38%	78	120	x	6764	0.40%	B1	177	±	6665	0.39%
07	7	.	6825	0.40%	40	64	@	6555	0.38%	79	121	y	6653	0.39%	B2	178	²	6700	0.39%
08	8	.	6753	0.39%	41	65	A	6663	0.39%	7A	122	z	6677	0.39%	B3	179	³	6473	0.38%
09	9	.	6670	0.39%	42	66	B	6749	0.39%	7B	123	{	6523	0.38%	B4	180	´	6613	0.39%
0A	10	.	6603	0.39%	43	67	C	6434	0.38%	7C	124	|	6600	0.39%	B5	181	µ	6555	0.38%
0B	11	.	6710	0.39%	44	68	D	6672	0.39%	7D	125	}	6544	0.38%	B6	182	¶	6637	0.39%
0C	12	.	6732	0.39%	45	69	E	6611	0.39%	7E	126	~	6762	0.40%	B7	183	·	6677	0.39%
0D	13	.	6634	0.39%	46	70	F	6688	0.39%	7F	127	[illegible]	6678	0.39%	B8	184	¸	6815	0.40%
0E	14	.	6835	0.40%	47	71	G	6685	0.39%	80	128	€	6673	0.39%	B9	185	¹	6745	0.39%
0F	15	.	6709	0.39%	48	72	H	6621	0.39%	81	129	[illegible]	6754	0.40%	BA	186	º	6575	0.38%
10	16	.	6668	0.39%	49	73	I	6605	0.39%	82	130	‚	6578	0.38%	BB	187	»	6726	0.39%
11	17	.	6627	0.39%	4A	74	J	6525	0.38%	83	131	ƒ	6681	0.39%	BC	188	¼	6572	0.38%
12	18	.	6840	0.40%	4B	75	K	6670	0.39%	84	132	„	6684	0.39%	BD	189	½	6872	0.40%
13	19	.	6811	0.40%	4C	76	L	6607	0.39%	85	133	…	6726	0.39%	BE	190	¾	6660	0.39%
14	20	.	6766	0.40%	4D	77	M	6630	0.39%	86	134	†	6726	0.39%	BF	191	¿	6562	0.38%
15	21	.	6618	0.39%	4E	78	N	6628	0.39%	87	135	‡	6696	0.39%	C0	192	À	6588	0.39%
16	22	.	6663	0.39%	4F	79	O	6690	0.39%	88	136	ˆ	6775	0.40%	C1	193	Á	6595	0.39%
17	23	.	6588	0.39%	50	80	P	6738	0.39%	89	137	‰	6592	0.39%	C2	194	Â	6589	0.39%
18	24	.	6708	0.39%	51	81	Q	6782	0.40%	8A	138	Š	6773	0.40%	C3	195	Ã	6669	0.39%
19	25	.	6611	0.39%	52	82	R	6619	0.39%	8B	139	‹	6666	0.39%	C4	196	Ä	6738	0.39%
1A	26	.	6689	0.39%	53	83	S	6564	0.38%	8C	140	Œ	6538	0.38%	C5	197	Å	6584	0.39%
1B	27	.	6740	0.39%	54	84	T	6758	0.40%	8D	141	[illegible]	6508	0.38%	C6	198	Æ	6865	0.40%
1C	28	.	6637	0.39%	55	85	U	6763	0.40%	8E	142	Ž	6689	0.39%	C7	199	Ç	6508	0.38%
1D	29	.	6623	0.39%	56	86	V	6669	0.39%	8F	143	[illegible]	6725	0.39%	C8	200	È	6677	0.39%
1E	30	.	6710	0.39%	57	87	W	6700	0.39%	90	144	[illegible]	6657	0.39%	C9	201	É	6539	0.38%
1F	31	.	6680	0.39%	58	88	X	6698	0.39%	91	145	‘	6660	0.39%	CA	202	Ê	6710	0.39%
20	32		6688	0.39%	59	89	Y	6699	0.39%	92	146	’	6653	0.39%	CB	203	Ë	6686	0.39%
21	33	!	6718	0.39%	5A	90	Z	6602	0.39%	93	147	“	6765	0.40%	CC	204	Ì	6791	0.40%
22	34	"	6651	0.39%	5B	91	[	6622	0.39%	94	148	”	6844	0.40%	CD	205	Í	6733	0.39%
23	35	#	6606	0.39%	5C	92	\	6833	0.40%	95	149	•	6684	0.39%	CE	206	Î	6640	0.39%
24	36	$	6608	0.39%	5D	93	]	6729	0.39%	96	150	–	6587	0.39%	CF	207	Ï	6613	0.39%
25	37	%	6747	0.39%	5E	94	^	6713	0.39%	97	151	—	6677	0.39%	D0	208	Ð	6775	0.40%
26	38	&	6720	0.39%	5F	95	_	6810	0.40%	98	152	˜	6712	0.39%	D1	209	Ñ	6701	0.39%
27	39	'	6692	0.39%	60	96	`	6641	0.39%	99	153	™	6760	0.40%	D2	210	Ò	6568	0.38%
28	40	(	6705	0.39%	61	97	a	6535	0.38%	9A	154	š	6740	0.39%	D3	211	Ó	6667	0.39%
29	41	)	6841	0.40%	62	98	b	6688	0.39%	9B	155	›	6572	0.38%	D4	212	Ô	6630	0.39%
2A	42	*	6739	0.39%	63	99	c	6666	0.39%	9C	156	œ	6654	0.39%	D5	213	Õ	6652	0.39%
2B	43	+	6721	0.39%	64	100	d	6684	0.39%	9D	157	[illegible]	6687	0.39%	D6	214	Ö	6782	0.40%
2C	44	,	6737	0.39%	65	101	e	6754	0.40%	9E	158	ž	6644	0.39%	D7	215	×	6851	0.40%
2D	45	-	6754	0.40%	66	102	f	6727	0.39%	9F	159	Ÿ	6747	0.39%	D8	216	Ø	6553	0.38%
2E	46	.	6575	0.38%	67	103	g	6709	0.39%	A0	160		6511	0.38%	D9	217	Ù	6671	0.39%
2F	47	/	6686	0.39%	68	104	h	6779	0.40%	A1	161	¡	6635	0.39%	DA	218	Ú	6676	0.39%
30	48	0	6689	0.39%	69	105	i	6569	0.38%	A2	162	¢	6824	0.40%	DB	219	Û	6656	0.39%
31	49	1	6594	0.39%	6A	106	j	6798	0.40%	A3	163	£	6910	0.40%	DC	220	Ü	6660	0.39%
32	50	2	6686	0.39%	6B	107	k	6629	0.39%	A4	164	¤	6770	0.40%	DD	221	Ý	6563	0.38%
33	51	3	6593	0.39%	6C	108	l	6659	0.39%	A5	165	¥	6608	0.39%	DE	222	Þ	6589	0.39%
34	52	4	6629	0.39%	6D	109	m	6537	0.38%	A6	166	¦	6733	0.39%	DF	223	ß	6651	0.39%
35	53	5	6786	0.40%	6E	110	n	6712	0.39%	A7	167	§	6724	0.39%	E0	224	à	6652	0.39%
36	54	6	6631	0.39%	6F	111	o	6852	0.40%	A8	168	¨	6638	0.39%	E1	225	á	6594	0.39%
37	55	7	6560	0.38%	70	112	p	6721	0.39%	A9	169	©	6691	0.39%	E2	226	â	6706	0.39%
38	56	8	6639	0.39%	71	113	q	6536	0.38%	AA	170	ª	6562	0.38%	E3	227	ã	6671	0.39%
															E4	228	ä	6570	0.38%
															E5	229	å	6677	0.39%
															E6	230	æ	6685	0.39%
															E7	231	ç	6662	0.39%
															E8	232	è	6650	0.39%
															E9	233	é	6819	0.40%
															EA	234	ê	6571	0.38%
															EB	235	ë	6691	0.39%
															EC	236	ì	6680	0.39%
															ED	237	í	6722	0.39%
															EE	238	î	6728	0.39%
															EF	239	ï	6709	0.39%
															F0	240	ð	6630	0.39%
															F1	241	ñ	6739	0.39%
															F2	242	ò	6650	0.39%
															F3	243	ó	6622	0.39%
															F4	244	ô	6565	0.38%
															F5	245	õ	6758	0.40%
															F6	246	ö	6727	0.39%
															F7	247	÷	6699	0.39%
															F8	248	ø	6694	0.39%
															F9	249	ù	6727	0.39%
															FA	250	ú	6805	0.40%
															FB	251	û	6687	0.39%
															FC	252	ü	6584	0.39%
															FD	253	ý	6783	0.40%
															FE	254	þ	6824	0.40%
															FF	255	ÿ	6771	0.40%

Table 3 shows the ASCII chart for Moby Dick encrypted in AES. This is every letter and the number and percentage that each takes up in the document. Table 4 contains the calculations based on the values above. This was done within excel and gets a closer look at the underlying calculations that are being performed by cryptool.

Table 4: Calculations on ASCII chart Including Sum, Variance, and Entropy

Total Number of Bytes:	231134
Sum of Byte Values:	4242921
Mean Byte:	18.3569747
∑(X-μ)^2 :	89746773.4
Variance (∑(X-μ)^2 / N) :	388.288929
Entropy:	7.87868995

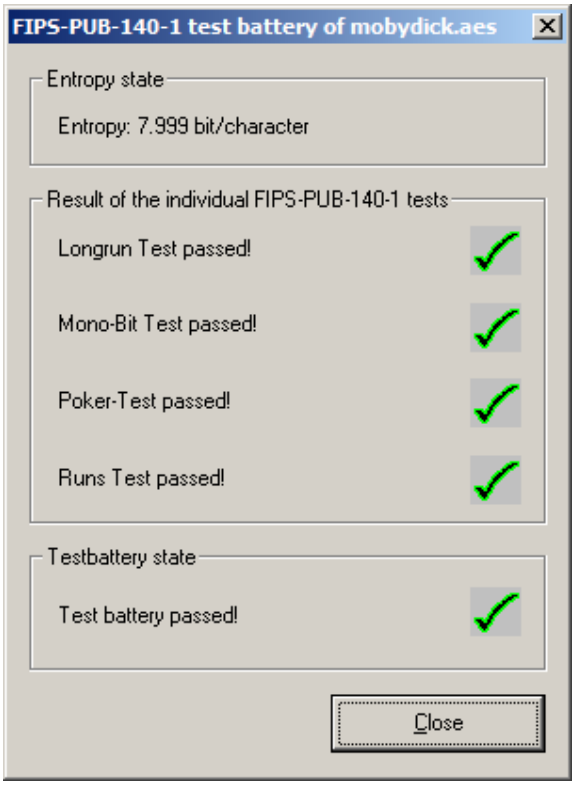


Figure 26: Battery of FIPS tests being performed on Moby Dick encrypted with AES

Figure 26 shows a battery of FIPS tests being performed using CrypTool. Basically these are a battery of statistical tests to make sure that the encrypted text doesn't reveal any weaknesses or areas which would allow it to be broken. Simple affine ciphers as well as basic substitution ciphers will only pass the long run test on a large document like Moby Dick, but the rest of the tests will generally fail (long run being a test that checks for runs of 26 or more). The mono-bit test treats each output bit of the random number generator as a coin flip test, and determines if the observed number of heads and tails are close to the expected 50% frequency. In this case it's based on the bits with 1s and 0s. The poker test tests for certain sequences of five numbers at a time and is based on hands in poker. The run test, also the Wald-Wolfowitz run test tests for the number of bit transitions between 0 bits, and 1 bits, comparing the observed frequencies with expected frequency of a random bit sequence. AES also passes a frequency test making sure that there aren't too many patterned similarities in the file. [29][(Hasegawa, Kim, 2008)]

Basically, what all of this is leading up to is that AES is a very secure algorithm. This was something that is widely known just going into this thesis. These calculations are just meant to show that simple attacks are unviable against something as strong as AES. Also, all of this information is meant to show that it's not any one part of AES that makes it secure, it is the combination of all of its parts that make it what it is, the most secure cipher publicly available today.

5.2 Cryptanalysis of Enigma Phoenix

This is the main portion of this thesis, this is the full Enigma Phoenix being tested in the same manner as AES above. The same statistical tests are used here as they were

with AES. The original plaintext of Moby Dick is being used again as well. This means that all of the analysis of the original plaintext of Moby Dick will be the same, such as the entropy being 4.55. Thus, in this section only the analysis of the Enigma Phoenix cipher will be discussed.

Key Used: ilikepancakes

Entropy: 7.99

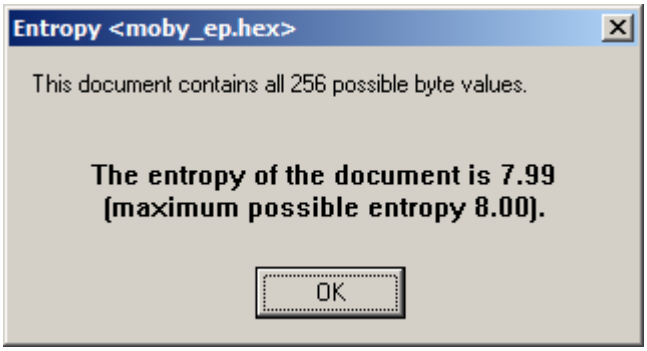


Figure 27: Entropy of Moby Dick encrypted with Enigma Phoenix

Figure 27 shows the entropy of Moby Dick encrypted with the Enigma Phoenix cipher. As can also be seen the entropy was 7.99 making it equivalent to AES. This was the first good sign that the cipher was strong. This high entropy proved that there was good randomness in the file and that no low level attacks would be successful in breaking it. This is also about as close to eight as it can be.

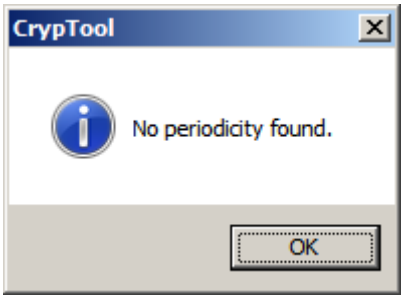


Figure 28: Checking for periodicity in Enigma Phoenix, no periodicity was found

Figure 28 is checking for the periodicity of Moby Dick encrypted with Enigma Phoenix. Just like AES no periodicity was found in the encrypted file proving that there are no easy to detect patterns that could be used to find attack vectors with which to break the cipher.

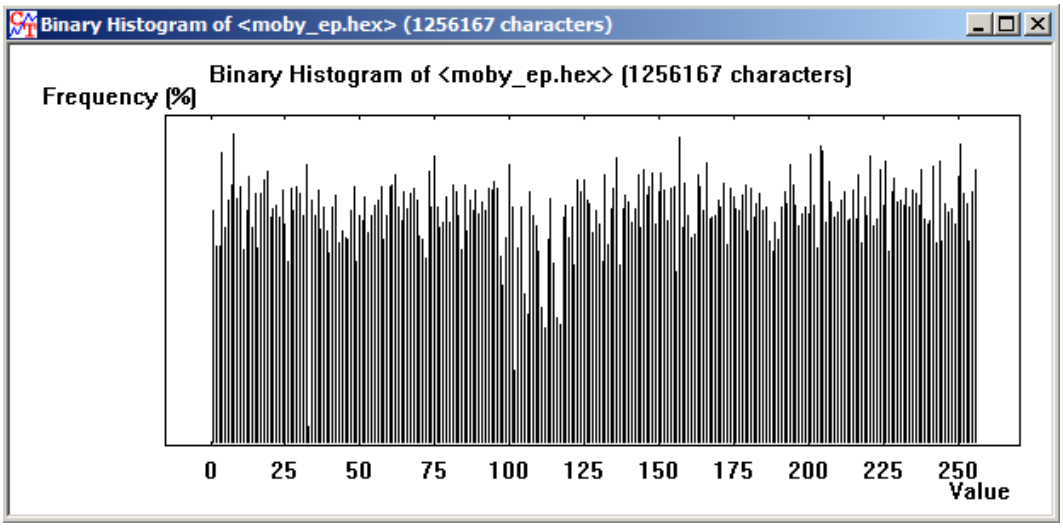


Figure 29: Binary histogram of Moby Dick encrypted with Enigma Phoenix

Figure 29 shows the binary histogram of Moby Dick encrypted with Enigma Phoenix. This is most definitely a uniform histogram with the occasional dip, it is actually better then AES. The histogram here shows that it would be less open to low level language attacks then even AES which is a feat in and of itself. It also proves that the cipher has more randomness through the confusion and diffusion.

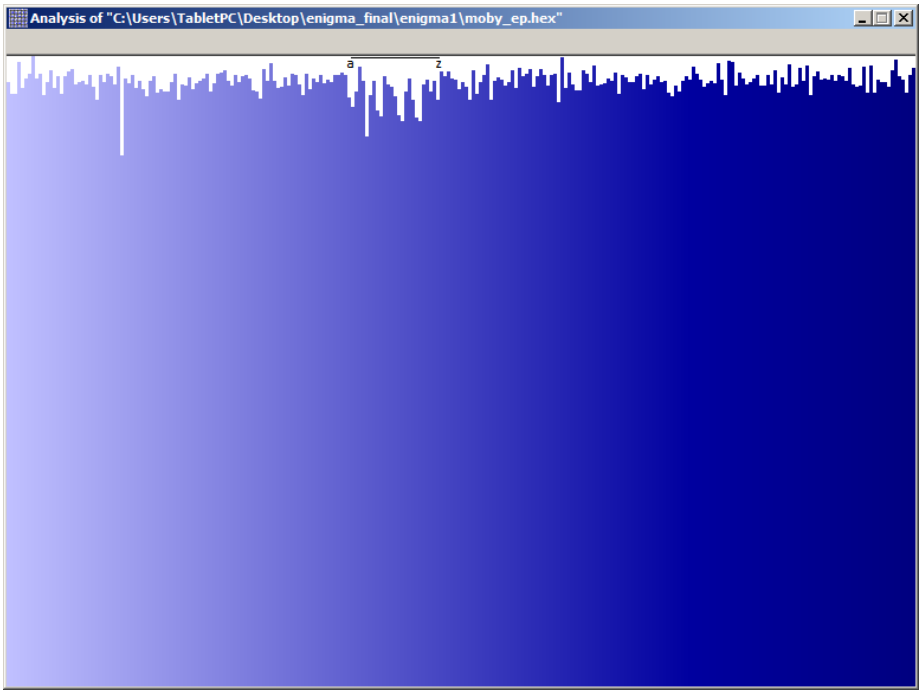


Figure 30: Binary Historgram of Moby Dick encrypted with Enigma Phoenix and analyzed in Winhex

Figure 30 is the analysis of Moby Dick encrypted with Enigma Phoenix done in Winhex. Winhex is a bit more of an advanced tool and it shows some greater dips that might prove to be a little weaker then AES. Either way it's still a very uniform binary histogram. This still shows that its quite secure by today's standards.

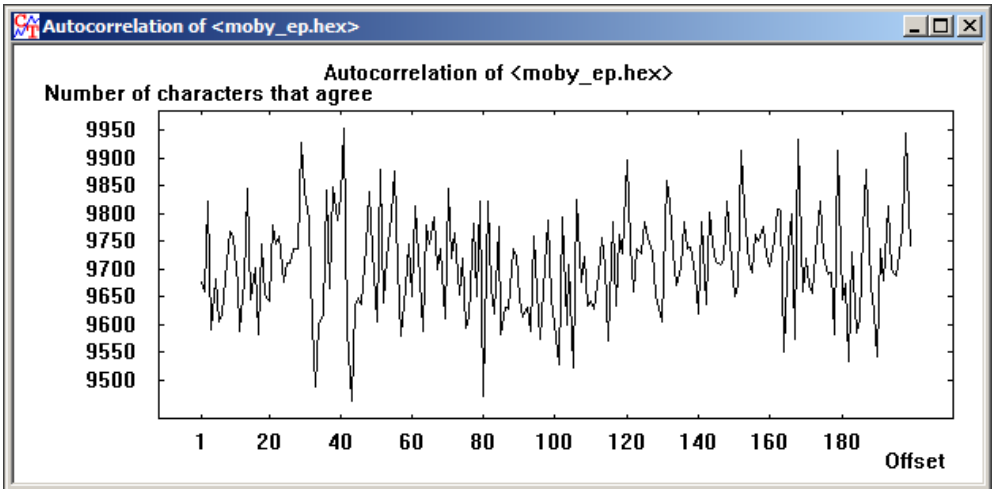


Figure 31: Autocorrelation of Moby Dick encrypted with Enigma Phoenix

Figure 31 shows the autocorrelation of Moby Dick encrypted with Enigma Phoenix. The autocorrelation here shows that there isn't a set pattern in the Enigma Phoenix. It's not quite as good as AES because it doesn't have the same range but it is very, very close to it. For a stream cipher to have this much randomness is unusual and really, for any cipher. This is a very strong algorithm and each portion of this analysis continues to prove that it is capable of holding its own with AES.

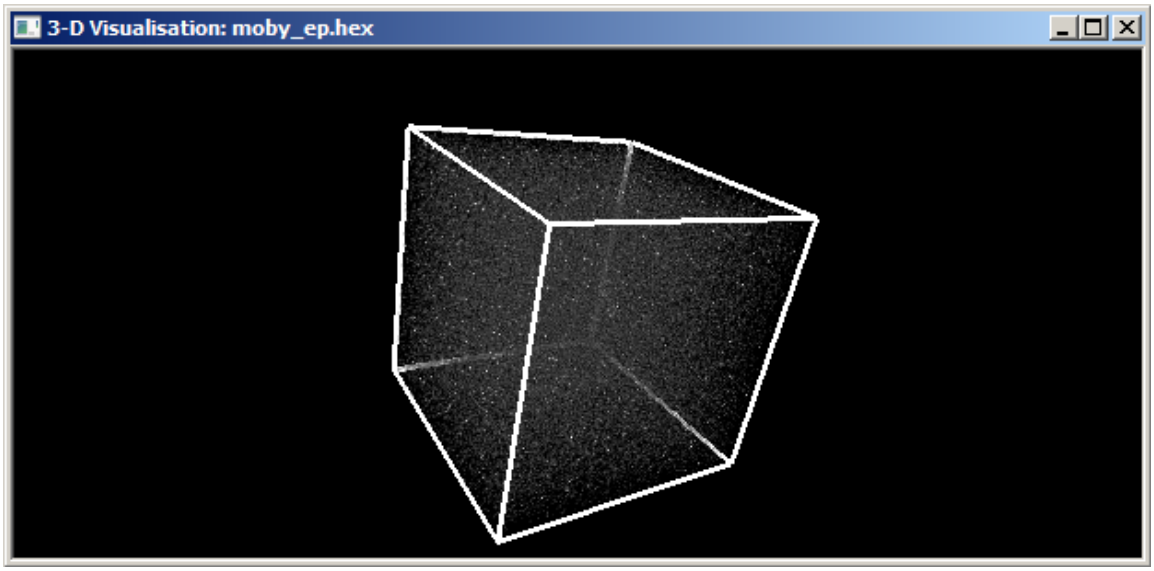


Figure 32: 3D representation of randomness in Moby Dick encrypted with Enigma Phoenix

Figure 32 is a 3D representation of the randomness in Moby Dick encrypted with the Enigma Phoenix cipher. As can be seen the randomness here is visually represented and backs up the results that are seen above. This is fantastic randomness and is well on par with AES. When comparing it to AES the distribution is fairly similar. This goes to prove that any low level attacks and even high level attacks will have great difficulty in finding a weakness solely in the ciphertext itself.

Table 5: Calculations on Moby Dick Encrypted with Enigma Phoenix

Total Number of Bytes:	170269
Sum of Byte Values:	3233709
Mean Byte:	18.991766
∑(X-μ)^2 :	83560088.5
Variance (∑(X-μ)^2 / N) :	490.753387
Entropy:	7.87665164

Table 5 contains the calculations based on an excel sheet and the data within has been included in the appendix. This just takes a closer look at the file, and, from this, it can be seen that while Enigma Phoenix is not quite on par with AES when encrypting the same file, it is exceptionally close. These are not necessarily exact calculations as they may differ from what CrypTool is outputting but they give just a better idea of how well it's actually being encrypted.

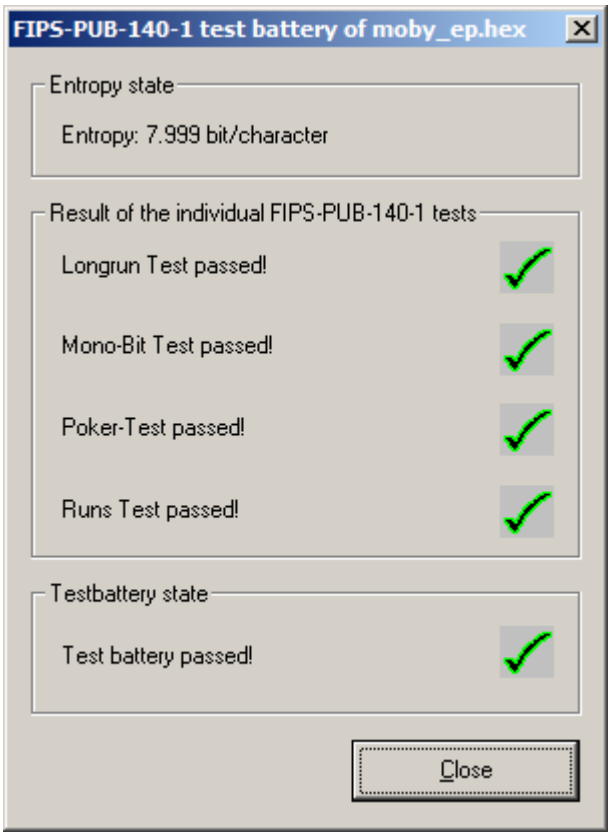


Figure 33: FIPS battery of randomness tests on Moby Dick encrypted with Enigma Phoenix

Figure 33 contains the FIPS batteries of tests that have all been explained in the AES analysis. Basically this continues to prove that the cipher is strong against all of the low level general attacks as well as any higher level brute force attacks. It shows that the ciphertext is random and that it contains no distinct patterns that can be used as a weakness against it.

6. Conclusion and Results

There are several conclusions that can be made from this. Firstly, this has proven to be a secure algorithm. Overall this is possibly one of the cryptographically strongest stream ciphers that can be made based on its comparison to AES. It is close to as cryptographically secure to AES. The addition of Galois functionality has proven to be a fruitful addition.

That's not to say that it isn't without problems. There are still several portions that could be improved upon and those are discussed in the recommendations section. However, the cipher proved to be solidly strong. It rivals, without any improvements, AES. That, in and of itself, is a feat. So what is the conclusion?

The conclusion is that re-engineering the Enigma Cipher and attempting to make it secure based on modern standards has proven possible. Not only has it proven possible, but it has happened. It is secure for today's computers. It holds up against tests built today to stress even the strongest algorithms.

7. Recommendations

As has been pointed out in this thesis, there is most definitely a main weakness in Enigma. While what has been created here is a very strong example of how secure a stream cipher can be, it still holds one main problem. When using the same key over and over again for multiple documents it could be possible, through comparison (likely using the Index of Coincidence and other methods) to determine what the key is. This is because there isn't a block chaining method that can hide patterns like a block cipher would (hence why the most popular and secure cipher today is a block cipher, AES). If there were a way to somehow convert the Enigma cipher into a block cipher it would be exceptionally secure.

Going so far as to say that as a block cipher Enigma would be equally or more secure then AES would possibly be a stretch. It is a possibility though. This is something that would need to be researched more thoroughly. It was more than likely not as apropos to compare AES to the Enigma Phoenix. It would have likely been better to use a similar stream cipher. As was discussed above, the only real weakness here though, is the fact that it's a stream cipher. Were it transformed into a block cipher then there would need to be a great deal of testing done.

There are several other recommendations that need to be made. The first of these is a general recommendation. This was more of a test than anything else, and that includes and is especially in reference to the addition of Galois operations. However, since, for the rotors, the Galois operations proved functional and secure, this should be extended to other portions. These other portions include the reflector and the plugboard. The reflector would be especially easy to generate with Galois fields since it works on a

very similar principle to the rotors. The plugboard would not necessarily work as well with any sort of Galois function, but it is foreseeable that it could somehow make use of one. The rotors occasionally run into spots where they are equal to the prior rotor just because of the way the vectors are set up. This could be modified at a later date to add even greater security to the cipher.

Just using the plugboard and reflector in some way outside of their original purpose would probably add a level of security. As far as this thesis goes, the main intention here was to modify the rotors to add security and test to see if it was viable. One last note is that adding more rotors would make it more secure, at some point it slows the algorithm down too much. Setting it to ten or so would probably make it completely pointless to even attempt a brute force on the most powerful computer in the world.

8. References

[1] Wolfram, S. (2002). A New Kind of Science. In S. Wolfram, *A New Kind of Science* (pp. 1085-1086). Champaign, Il: Library of Congress Cataloging.

[2] Berghel, H. (April 2008). Faith-Based Security. *Communications of The ACM* , 14.

[3] Algred J.Menezes, P. C. (1996). *Introduction to Cryptography - Symmetric Key Encryption, Handbook of Applied Cryptography* (5th ed.). Boca Raton, Florida: CRC-Press.

[4] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 173-174). New York, New York: John Wiley & Sons, Inc.

[5] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 31-32). New York, New York: John Wiley & Sons, Inc.

[6] Public Domain Images, "Wikipedia - Fiestel Cipher," in http://en.wikipedia.org/wiki/Image:Feistel.png 2008.

[7] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 347). New York, New York: John Wiley & Sons, Inc.

[8] Miller, D. A. (2008). *The Cryptographic Mathematics of Enigma.* Retrieved 2008, from NSA Publications: http://www.nsa.gov/publications/publi00004.cfm

[9] Vaudenay, S. (2006). *A Classical Introduction to Cryptogrpahy.* (pp. 42-43) New York, New York: Springer.

[10] Public Domain Images, "Wikipedia - AES," in http://en.wikipedia.org/wiki/Advanced_Encryption_Standard 2008.

[11] Vaudenay, S. (2006). *A Classical Introduction to Cryptogrpahy.* (pp. 44-46) New York, New York: Springer.

[12] Amit Parnerkar, D. G. (2003). SECRET KEY DISTRIBUTION PROTOCOL USING PUBLIC KEY CRYPTOGRAPHY. *CCSC: Rocky Mountain Conference.* CCSC.

[13] Public Domain Images, “Wikipedia – Enigma Chiper,” In http://en.wikipedia.org/wiki/Image:Enigma-plugboard.jpg 2008.

[14] Public Domain Images, “Wikipedia – Enigma Chiper,” In http://en.wikipedia.org/wiki/Image:Enigma_wiring_kleur.svg 2008.

[15] Public Domain Images, “Wikipedia – Enigma Chiper,” In http://en.wikipedia.org/wiki/Image:Enigma-action.svg 2008.

[16] Vaudenay, S. (2006). *A Classical Introduction to Cryptogrpahy.* (pp. 8-11) New York, New York: Springer.

[17] T. Jamil, "The Rijndael algorithm," *Potentials, IEEE*, vol. 23, no. 2, pp. 30-32, 2004.

[18] Vaudenay, S. (2006). *A Classical Introduction to Cryptogrpahy.* (pp. 25-27) New York, New York: Springer.

[19] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 197-199). New York, New York: John Wiley & Sons, Inc.

[20] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 349-351). New York, New York: John Wiley & Sons, Inc.

[21] Sam Trenholme, "AES’ Galois Field," in http://samiam.org/galois.html 2008.

[22] Volker Paelke, J. S. (2002). The AR-ENIGMA – A PDA based Interactive Illustration. *SIGGRAPH*, (p. 260).

[23] Kahn, D. (1991). *Seizing the Enigma the Race to Break the German U-Boat Codes 1939-1943* (pp. 39-46). New York, New York: Houghton Mifflin.

[24] Jarkko Kari, "Cryptosystems Based on Reversible Cellular Automata, University of Turku, Finland," 1992.

[25] Public Domain Image, "Information Entropy," in http://en.wikipedia.org/wiki/Index_of_coincidence 2008.

[26] Schneier, B. (1996). Applied Cryptography. In B. Schneier, *Applied Cryptography* (Second ed., pp. 233-234). New York, New York: John Wiley & Sons, Inc.

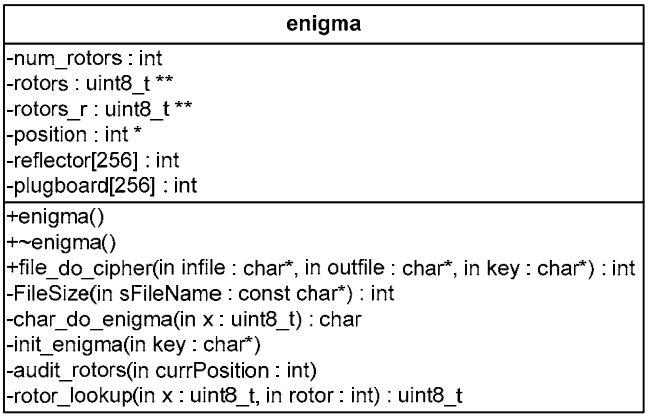
[27] Public Domain Image, "Information Entropy," in http://en.wikipedia.org/wiki/Information_entropy 2008

[28] Public Domain Images, "Conditional Entropy," in http://en.wikipedia.org/wiki/Conditional_entropy 2008.

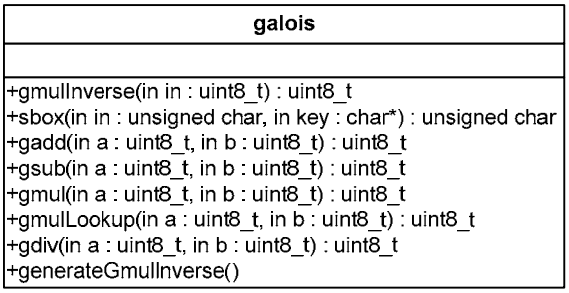
[29] Akio Hasegawa, S.-J. K. (2008). *IP Core of Statistical Test Suite of FIPS 140-2.* Retrieved from Design Reuse: http://www.design-reuse.com/articles/7946/ip-core-of-statistical-test-suite-of-fips-140-2.html

Appendix A: Design Diagrams

A1. Enigma Class

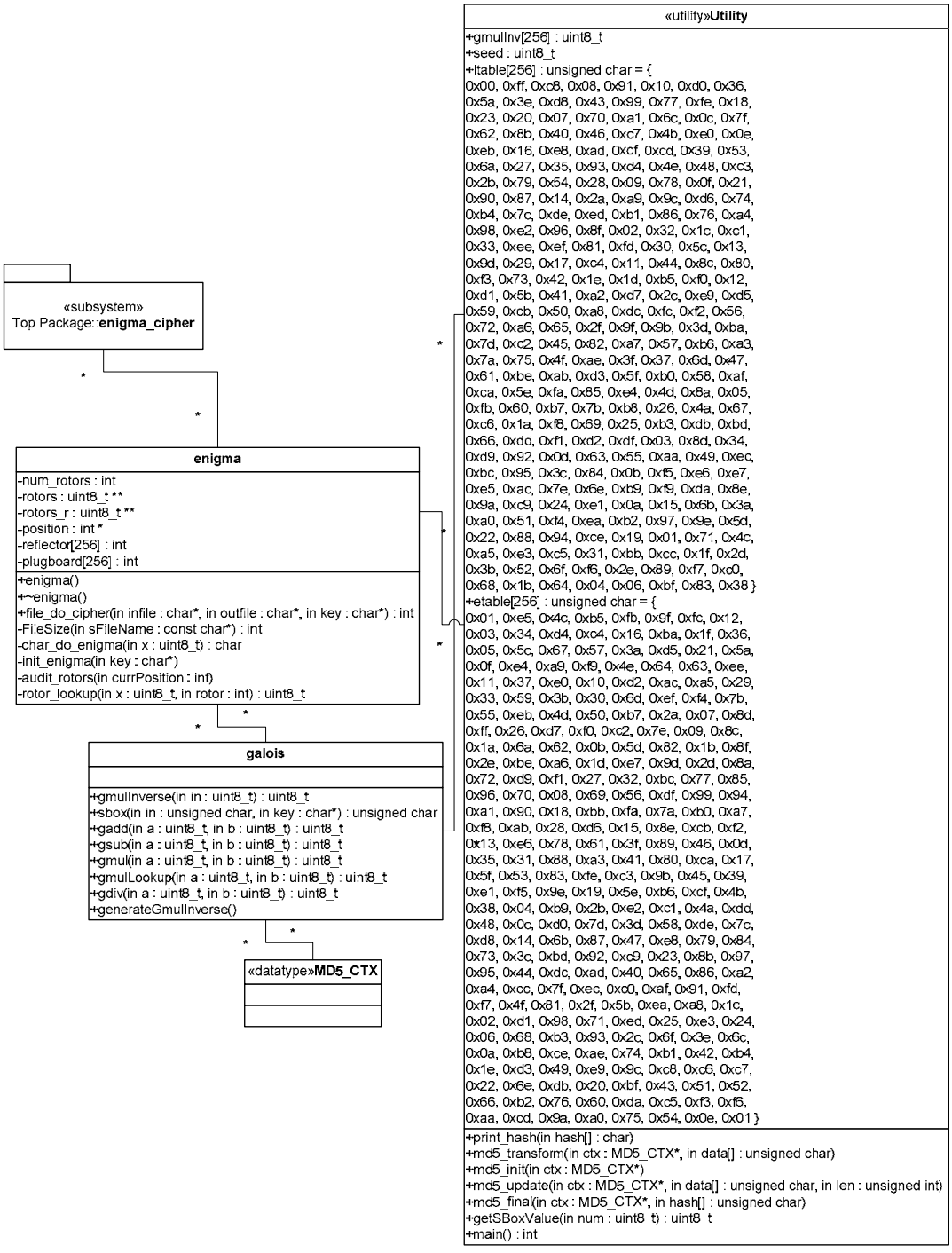


A2. Galois Class



A3. MD5 and Galois Utility Functionality

A4. Code Overview



Appendix B: Code

B1. MD5 Header File

```
#ifndef MD5_H
#define MD5_H
#include <stdio.h>
#include <string.h>

typedef struct {
  unsigned char data[64];
  unsigned int datalen;
  unsigned int bitlen[2];
  unsigned int state[4];
} MD5_CTX;

void print_hash(char hash[]);

void md5_transform(MD5_CTX *ctx, unsigned char data[]);

void md5_init(MD5_CTX *ctx);

void md5_update(MD5_CTX *ctx, unsigned char data[], unsigned int len);

void md5_final(MD5_CTX *ctx, unsigned char hash[]);

#endif
```

B2. MD5 Implementation File

```
// MD5 Hash Digest implementation (little endian byte order)
#include "md5.h"

// Bah, signed variables are for wimps
#define uchar unsigned char
#define uint unsigned int

// DBL_INT_ADD treats two unsigned ints a and b as one 64-bit integer and adds c to it
#define DBL_INT_ADD(a,b,c) if (a > 0xffffffff - c) ++b; a += c;
#define ROTLEFT(a,b) ((a << b) | (a >> (32-b)))

#define F(x,y,z) ((x & y) | (~x & z))
#define G(x,y,z) ((x & z) | (y & ~z))
#define H(x,y,z) (x ^ y ^ z)
#define I(x,y,z) (y ^ (x | ~z))

#define FF(a,b,c,d,m,s,t) { a += F(b,c,d) + m + t; \
                            a = b + ROTLEFT(a,s); }
```

```
#define GG(a,b,c,d,m,s,t) { a += G(b,c,d) + m + t; \
                        a = b + ROTLEFT(a,s); }
#define HH(a,b,c,d,m,s,t) { a += H(b,c,d) + m + t; \
                        a = b + ROTLEFT(a,s); }
#define II(a,b,c,d,m,s,t) { a += I(b,c,d) + m + t; \
                        a = b + ROTLEFT(a,s); }

void print_hash(char hash[])
{
   int idx;
   int x;

   for (idx=0; idx < 16; idx++) {
         printf("%02x",hash[idx]&0x000000ff);
   }

   printf("\n");
}

void md5_transform(MD5_CTX *ctx, uchar data[])
{
  uint a,b,c,d,m[16],i,j;

  // MD5 specifies big endian byte order, but this implementation assumes a little
  // endian byte order CPU. Reverse all the bytes upon input, and re-reverse them
  // on output (in md5_final()).
  for (i=0,j=0; i < 16; ++i, j += 4)
    m[i] = (data[j]) + (data[j+1] << 8) + (data[j+2] << 16) + (data[j+3] << 24);

  a = ctx->state[0];
  b = ctx->state[1];
  c = ctx->state[2];
  d = ctx->state[3];

  FF(a,b,c,d,m[0],  7,0xd76aa478);
  FF(d,a,b,c,m[1], 12,0xe8c7b756);
  FF(c,d,a,b,m[2], 17,0x242070db);
  FF(b,c,d,a,m[3], 22,0xc1bdceee);
  FF(a,b,c,d,m[4],  7,0xf57c0faf);
  FF(d,a,b,c,m[5], 12,0x4787c62a);
  FF(c,d,a,b,m[6], 17,0xa8304613);
  FF(b,c,d,a,m[7], 22,0xfd469501);
  FF(a,b,c,d,m[8],  7,0x698098d8);
  FF(d,a,b,c,m[9], 12,0x8b44f7af);
  FF(c,d,a,b,m[10],17,0xffff5bb1);
  FF(b,c,d,a,m[11],22,0x895cd7be);
```

```
FF(a,b,c,d,m[12], 7,0x6b901122);
FF(d,a,b,c,m[13],12,0xfd987193);
FF(c,d,a,b,m[14],17,0xa679438e);
FF(b,c,d,a,m[15],22,0x49b40821);

GG(a,b,c,d,m[1],  5,0xf61e2562);
GG(d,a,b,c,m[6],  9,0xc040b340);
GG(c,d,a,b,m[11],14,0x265e5a51);
GG(b,c,d,a,m[0], 20,0xe9b6c7aa);
GG(a,b,c,d,m[5],  5,0xd62f105d);
GG(d,a,b,c,m[10], 9,0x02441453);
GG(c,d,a,b,m[15],14,0xd8a1e681);
GG(b,c,d,a,m[4], 20,0xe7d3fbc8);
GG(a,b,c,d,m[9],  5,0x21e1cde6);
GG(d,a,b,c,m[14], 9,0xc33707d6);
GG(c,d,a,b,m[3], 14,0xf4d50d87);
GG(b,c,d,a,m[8], 20,0x455a14ed);
GG(a,b,c,d,m[13], 5,0xa9e3e905);
GG(d,a,b,c,m[2],  9,0xfcefa3f8);
GG(c,d,a,b,m[7], 14,0x676f02d9);
GG(b,c,d,a,m[12],20,0x8d2a4c8a);

HH(a,b,c,d,m[5],  4,0xfffa3942);
HH(d,a,b,c,m[8], 11,0x8771f681);
HH(c,d,a,b,m[11],16,0x6d9d6122);
HH(b,c,d,a,m[14],23,0xfde5380c);
HH(a,b,c,d,m[1],  4,0xa4beea44);
HH(d,a,b,c,m[4], 11,0x4bdecfa9);
HH(c,d,a,b,m[7], 16,0xf6bb4b60);
HH(b,c,d,a,m[10],23,0xbebfbc70);
HH(a,b,c,d,m[13], 4,0x289b7ec6);
HH(d,a,b,c,m[0], 11,0xeaa127fa);
HH(c,d,a,b,m[3], 16,0xd4ef3085);
HH(b,c,d,a,m[6], 23,0x04881d05);
HH(a,b,c,d,m[9],  4,0xd9d4d039);
HH(d,a,b,c,m[12],11,0xe6db99e5);
HH(c,d,a,b,m[15],16,0x1fa27cf8);
HH(b,c,d,a,m[2], 23,0xc4ac5665);

II(a,b,c,d,m[0],  6,0xf4292244);
II(d,a,b,c,m[7], 10,0x432aff97);
II(c,d,a,b,m[14],15,0xab9423a7);
II(b,c,d,a,m[5], 21,0xfc93a039);
II(a,b,c,d,m[12], 6,0x655b59c3);
II(d,a,b,c,m[3], 10,0x8f0ccc92);
II(c,d,a,b,m[10],15,0xffeff47d);
```

```
  II(b,c,d,a,m[1], 21,0x85845dd1);
  II(a,b,c,d,m[8],  6,0x6fa87e4f);
  II(d,a,b,c,m[15],10,0xfe2ce6e0);
  II(c,d,a,b,m[6], 15,0xa3014314);
  II(b,c,d,a,m[13],21,0x4e0811a1);
  II(a,b,c,d,m[4],  6,0xf7537e82);
  II(d,a,b,c,m[11],10,0xbd3af235);
  II(c,d,a,b,m[2], 15,0x2ad7d2bb);
  II(b,c,d,a,m[9], 21,0xeb86d391);

  ctx->state[0] += a;
  ctx->state[1] += b;
  ctx->state[2] += c;
  ctx->state[3] += d;
}

void md5_init(MD5_CTX *ctx)
{
  ctx->datalen = 0;
  ctx->bitlen[0] = 0;
  ctx->bitlen[1] = 0;
  ctx->state[0] = 0x67452301;
  ctx->state[1] = 0xEFCDAB89;
  ctx->state[2] = 0x98BADCFE;
  ctx->state[3] = 0x10325476;
}

void md5_update(MD5_CTX *ctx, uchar data[], uint len)
{
  uint t,i;

  for (i=0; i < len; ++i) {
    ctx->data[ctx->datalen] = data[i];
    ctx->datalen++;
    if (ctx->datalen == 64) {
      md5_transform(ctx,ctx->data);
      DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],512);
      ctx->datalen = 0;
    }
  }
}

void md5_final(MD5_CTX *ctx, uchar hash[])
{
  uint i;
```

```
  i = ctx->datalen;

  // Pad whatever data is left in the buffer.
  if (ctx->datalen < 56) {
    ctx->data[i++] = 0x80;
    while (i < 56)
      ctx->data[i++] = 0x00;
  }
  else if (ctx->datalen >= 56) {
    ctx->data[i++] = 0x80;
    while (i < 64)
      ctx->data[i++] = 0x00;
    md5_transform(ctx,ctx->data);
    memset(ctx->data,0,56);
  }

  // Append to the padding the total message's length in bits and transform.
  DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],8 * ctx->datalen);
  ctx->data[56] = ctx->bitlen[0];
  ctx->data[57] = ctx->bitlen[0] >> 8;
  ctx->data[58] = ctx->bitlen[0] >> 16;
  ctx->data[59] = ctx->bitlen[0] >> 24;
  ctx->data[60] = ctx->bitlen[1];
  ctx->data[61] = ctx->bitlen[1] >> 8;
  ctx->data[62] = ctx->bitlen[1] >> 16;
  ctx->data[63] = ctx->bitlen[1] >> 24;
  md5_transform(ctx,ctx->data);

  // Since this implementation uses little endian byte ordering and MD uses big endian,
  // reverse all the bytes when copying the final state to the output hash.
  for (i=0; i < 4; ++i) {
    hash[i]    = (ctx->state[0] >> (i*8)) & 0x000000ff;
    hash[i+4]  = (ctx->state[1] >> (i*8)) & 0x000000ff;
    hash[i+8]  = (ctx->state[2] >> (i*8)) & 0x000000ff;
    hash[i+12] = (ctx->state[3] >> (i*8)) & 0x000000ff;
  }

}
```

B3. Galois Header File

```
#ifndef GALOIS_H
#define GALOIS_H
#include <iostream>
#include <fstream>
#include <stdint.h>
```

```
#include <stdlib.h>
#include <math.h>

/*uint8_t key[16] =
{0x00,0x08,0x10,0x18,0x20,0x28,0x30,0x38,0x40,0x48,0x50,0x58,0x60,0x68,0x70,0x7
8};
uint8_t state[16]; */

using namespace std;
class galois
{

public:
   uint8_t gmulInverse(uint8_t in);
   unsigned char sbox(unsigned char in, char * key);
//    void sub_bytes(char streamLetter, unsigned char *key)

   uint8_t gadd(uint8_t a, uint8_t b);
   uint8_t gsub(uint8_t a, uint8_t b);
   uint8_t gmul(uint8_t a, uint8_t b);
   uint8_t gmulLookup(uint8_t a, uint8_t b);
   uint8_t gdiv(uint8_t a, uint8_t b);
   void generateGmulInverse();
};

#endif
```

B4. Galois Class File

```
#include "galois.h"

uint8_t gmulInv[256];
uint8_t seed;

const uint8_t generators[128] = {
                        0x03, 0x05, 0x06, 0x09, 0x0b, 0x0e, 0x11, 0x12, 0x13, 0x14,
0x17, 0x18, 0x19, 0x1a, 0x1c, 0x1e,
                        0x1f, 0x21, 0x22, 0x23, 0x27, 0x28, 0x2a, 0x2c, 0x30, 0x31, 0x3c,
0x3e, 0x3f, 0x41, 0x45, 0x46,
                        0x47, 0x48, 0x49, 0x4b, 0x4c, 0x4e, 0x4f, 0x52, 0x54, 0x56, 0x57,
0x58, 0x59, 0x5a, 0x5b, 0x5f,
                        0x64, 0x65, 0x68, 0x69, 0x6d, 0x6e, 0x70, 0x71, 0x76, 0x77,
0x79, 0x7a, 0x7b, 0x7e, 0x81, 0x84,
                        0x86, 0x87, 0x88, 0x8a, 0x8e, 0x8f, 0x90, 0x93, 0x95, 0x96, 0x98,
0x99, 0x9b, 0x9d, 0xa0, 0xa4,
```

```
                        0xa5, 0xa6, 0xa7, 0xa9, 0xaa, 0xac, 0xad, 0xb2, 0xb4, 0xb7, 0xb8,
0xb9, 0xba, 0xbe, 0xbf, 0xc0,
                        0xc1, 0xc4, 0xc8, 0xc9, 0xce, 0xcf, 0xd0, 0xd6, 0xd7, 0xda, 0xdc,
0xdd, 0xde, 0xe2, 0xe3, 0xe5,
                        0xe6, 0xe7, 0xe9, 0xea, 0xeb, 0xee, 0xf0, 0xf1, 0xf4, 0xf5, 0xf6,
0xf8, 0xfb, 0xfd, 0xfe, 0xff
                    };

unsigned char ltable[256] = {
0x00, 0xff, 0xc8, 0x08, 0x91, 0x10, 0xd0, 0x36,
0x5a, 0x3e, 0xd8, 0x43, 0x99, 0x77, 0xfe, 0x18,
0x23, 0x20, 0x07, 0x70, 0xa1, 0x6c, 0x0c, 0x7f,
0x62, 0x8b, 0x40, 0x46, 0xc7, 0x4b, 0xe0, 0x0e,
0xeb, 0x16, 0xe8, 0xad, 0xcf, 0xcd, 0x39, 0x53,
0x6a, 0x27, 0x35, 0x93, 0xd4, 0x4e, 0x48, 0xc3,
0x2b, 0x79, 0x54, 0x28, 0x09, 0x78, 0x0f, 0x21,
0x90, 0x87, 0x14, 0x2a, 0xa9, 0x9c, 0xd6, 0x74,
0xb4, 0x7c, 0xde, 0xed, 0xb1, 0x86, 0x76, 0xa4,
0x98, 0xe2, 0x96, 0x8f, 0x02, 0x32, 0x1c, 0xc1,
0x33, 0xee, 0xef, 0x81, 0xfd, 0x30, 0x5c, 0x13,
0x9d, 0x29, 0x17, 0xc4, 0x11, 0x44, 0x8c, 0x80,
0xf3, 0x73, 0x42, 0x1e, 0x1d, 0xb5, 0xf0, 0x12,
0xd1, 0x5b, 0x41, 0xa2, 0xd7, 0x2c, 0xe9, 0xd5,
0x59, 0xcb, 0x50, 0xa8, 0xdc, 0xfc, 0xf2, 0x56,
0x72, 0xa6, 0x65, 0x2f, 0x9f, 0x9b, 0x3d, 0xba,
0x7d, 0xc2, 0x45, 0x82, 0xa7, 0x57, 0xb6, 0xa3,
0x7a, 0x75, 0x4f, 0xae, 0x3f, 0x37, 0x6d, 0x47,
0x61, 0xbe, 0xab, 0xd3, 0x5f, 0xb0, 0x58, 0xaf,
0xca, 0x5e, 0xfa, 0x85, 0xe4, 0x4d, 0x8a, 0x05,
0xfb, 0x60, 0xb7, 0x7b, 0xb8, 0x26, 0x4a, 0x67,
0xc6, 0x1a, 0xf8, 0x69, 0x25, 0xb3, 0xdb, 0xbd,
0x66, 0xdd, 0xf1, 0xd2, 0xdf, 0x03, 0x8d, 0x34,
0xd9, 0x92, 0x0d, 0x63, 0x55, 0xaa, 0x49, 0xec,
0xbc, 0x95, 0x3c, 0x84, 0x0b, 0xf5, 0xe6, 0xe7,
0xe5, 0xac, 0x7e, 0x6e, 0xb9, 0xf9, 0xda, 0x8e,
0x9a, 0xc9, 0x24, 0xe1, 0x0a, 0x15, 0x6b, 0x3a,
0xa0, 0x51, 0xf4, 0xea, 0xb2, 0x97, 0x9e, 0x5d,
0x22, 0x88, 0x94, 0xce, 0x19, 0x01, 0x71, 0x4c,
0xa5, 0xe3, 0xc5, 0x31, 0xbb, 0xcc, 0x1f, 0x2d,
0x3b, 0x52, 0x6f, 0xf6, 0x2e, 0x89, 0xf7, 0xc0,
0x68, 0x1b, 0x64, 0x04, 0x06, 0xbf, 0x83, 0x38 };

unsigned char etable[256] = {
0x01, 0xe5, 0x4c, 0xb5, 0xfb, 0x9f, 0xfc, 0x12,
0x03, 0x34, 0xd4, 0xc4, 0x16, 0xba, 0x1f, 0x36,
0x05, 0x5c, 0x67, 0x57, 0x3a, 0xd5, 0x21, 0x5a,
```

```
0x0f, 0xe4, 0xa9, 0xf9, 0x4e, 0x64, 0x63, 0xee,
0x11, 0x37, 0xe0, 0x10, 0xd2, 0xac, 0xa5, 0x29,
0x33, 0x59, 0x3b, 0x30, 0x6d, 0xef, 0xf4, 0x7b,
0x55, 0xeb, 0x4d, 0x50, 0xb7, 0x2a, 0x07, 0x8d,
0xff, 0x26, 0xd7, 0xf0, 0xc2, 0x7e, 0x09, 0x8c,
0x1a, 0x6a, 0x62, 0x0b, 0x5d, 0x82, 0x1b, 0x8f,
0x2e, 0xbe, 0xa6, 0x1d, 0xe7, 0x9d, 0x2d, 0x8a,
0x72, 0xd9, 0xf1, 0x27, 0x32, 0xbc, 0x77, 0x85,
0x96, 0x70, 0x08, 0x69, 0x56, 0xdf, 0x99, 0x94,
0xa1, 0x90, 0x18, 0xbb, 0xfa, 0x7a, 0xb0, 0xa7,
0xf8, 0xab, 0x28, 0xd6, 0x15, 0x8e, 0xcb, 0xf2,
0x13, 0xe6, 0x78, 0x61, 0x3f, 0x89, 0x46, 0x0d,
0x35, 0x31, 0x88, 0xa3, 0x41, 0x80, 0xca, 0x17,
0x5f, 0x53, 0x83, 0xfe, 0xc3, 0x9b, 0x45, 0x39,
0xe1, 0xf5, 0x9e, 0x19, 0x5e, 0xb6, 0xcf, 0x4b,
0x38, 0x04, 0xb9, 0x2b, 0xe2, 0xc1, 0x4a, 0xdd,
0x48, 0x0c, 0xd0, 0x7d, 0x3d, 0x58, 0xde, 0x7c,
0xd8, 0x14, 0x6b, 0x87, 0x47, 0xe8, 0x79, 0x84,
0x73, 0x3c, 0xbd, 0x92, 0xc9, 0x23, 0x8b, 0x97,
0x95, 0x44, 0xdc, 0xad, 0x40, 0x65, 0x86, 0xa2,
0xa4, 0xcc, 0x7f, 0xec, 0xc0, 0xaf, 0x91, 0xfd,
0xf7, 0x4f, 0x81, 0x2f, 0x5b, 0xea, 0xa8, 0x1c,
0x02, 0xd1, 0x98, 0x71, 0xed, 0x25, 0xe3, 0x24,
0x06, 0x68, 0xb3, 0x93, 0x2c, 0x6f, 0x3e, 0x6c,
0x0a, 0xb8, 0xce, 0xae, 0x74, 0xb1, 0x42, 0xb4,
0x1e, 0xd3, 0x49, 0xe9, 0x9c, 0xc8, 0xc6, 0xc7,
0x22, 0x6e, 0xdb, 0x20, 0xbf, 0x43, 0x51, 0x52,
0x66, 0xb2, 0x76, 0x60, 0xda, 0xc5, 0xf3, 0xf6,
0xaa, 0xcd, 0x9a, 0xa0, 0x75, 0x54, 0x0e, 0x01 };

uint8_t getSBoxValue(uint8_t num)
{
uint8_t sbox[256] =   {
//0     1    2      3     4    5     6     7      8    9     A      B C     D     E     F
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76, //0
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15, //2
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75, //3
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84, //4
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf, //5
```

```
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2, //7
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73, //8
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb, //9
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79, //A
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08, //B
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf, //E
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F
return sbox[num];
}


uint8_t galois::gmulInverse(uint8_t in)
{
   if (in == 0)
      return 0;
   else
      return etable[(255 - ltable[in])];
}

unsigned char galois::sbox(unsigned char in, char * key) {
      unsigned char c, s, x;
      s = x = gmulInverse(in);
      for(c = 0; c < 4; c++) {
            /* One bit circular rotate to the left */
            s = (s << 1) | (s >> 7);
            /* xor with x */
            x ^= s;
      }
      x ^= (int)key;//99; /* 0x63 */
//       cout << x;
      return x;
}
```

```
uint8_t galois::gadd(uint8_t a, uint8_t b)
{
   return a ^ b;
}

uint8_t galois::gsub(uint8_t a, uint8_t b)
{
   return a ^ b;
}

uint8_t galois::gmul(uint8_t a, uint8_t b)
{
   uint8_t p = 0;
   uint8_t counter;
   uint8_t hi_bit_set;
   for(counter = 0; counter < 8; counter++)
   {
      if ((b & 1) == 1)
         p ^= a;
      hi_bit_set = (a & 0x80);
      a <<= 1;
      if (hi_bit_set == 0x80)
         a ^= 0x1b;
      b >>= 1;
   }
   return p;
}

uint8_t galois::gmulLookup(uint8_t a, uint8_t b)
{
   int s;
   int q;
   int z = 0;
   s = ltable[a] + ltable[b];
   s %= 255;
   s = etable[s];
   q = s;
   if (a == 0)
   {
      s = z;
   }
   else
   {
      s = q;
   }
   if (b == 0)
```

```
    {
        s = z;
    }
    else
    {
        q = z;
    }
    return s;
}

uint8_t galois::gdiv(uint8_t a, uint8_t b)
{
    uint8_t c = ltable[a]-ltable[b];
    return c%255;
}

void galois::generateGmulInverse()
{
    gmulInv[0] = 0;
    for (int c=1;c<256;c++)
    {
        gmulInv[c] = gmulInverse(c);
    }
}
```

B5. Enigma Header File

```
#ifndef ENIGMA_H
#define ENIGMA_H

//#include "galois.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
//#include <iostream.h>
#include <string.h>
#include <math.h>
#include <fstream>
#include <cstdio>
#include <stdint.h>
#include <sys/stat.h>
#include <iostream>
#include <vector>
#include "md5.h"
#include "galois.h"
```

```
//uses 128 bit key
using namespace std;

/*typedef struct ListStruct {
   int val;
   struct ListStruct * ptr;
} List;*/

//int delete_list_entry(List * list, int entry);

class enigma
{
   public:
      enigma();
      ~enigma();
      int file_do_cipher(char * infile, char * outfile, char * key);

   private:
      int FileSize(const char* sFileName);
      char char_do_enigma(uint8_t x);
      void init_enigma(char * key);
      void audit_rotors(int currPosition);
      uint8_t rotor_lookup(uint8_t x, int rotor);

      int num_rotors;
      uint8_t ** rotors;
      uint8_t ** rotors_r;
      int * position;

      int reflector[256];
      int plugboard[256];
};

#endif
```

B6. Enigma Class File

```
#include "enigma.h"
//stl vectors


enigma::enigma() {
```

```
}

/* free up memory for  char ** rotors, and  int * positions */
enigma::~enigma() {
   //free up memory!
   for(int i=0; i<num_rotors; i++)
      delete []rotors[i];

   delete []rotors;
   delete []position;
}

int enigma::FileSize(const char* sFileName)
{
   ifstream f;
   f.open(sFileName, ios_base::binary | ios_base::in);
   if (!f.good() || f.eof() || !f.is_open()) { return 0; }
   f.seekg(0, ios_base::beg);
   ifstream::pos_type begin_pos = f.tellg();
   f.seekg(0, ios_base::end);
   return static_cast<int>(f.tellg() - begin_pos);
}

/* call init_enigma, read each character from file, encrypt/decrypt it with
char_encrypt, then write it back to new file */
int enigma::file_do_cipher(char * infile, char * outfile, char * key)
{
   init_enigma(key);
   //FILE *inf, *of;
  // struct stat results;

   int fsize = FileSize(infile);
   /*if (stat(infile, &results) != 0) {
      printf("Cannot open file, quitting \n");
      exit(1);
   }*/
   //inf = fopen(infile, "rb");
   //of = fopen(outfile, "wb+");

   /*int filesize = results.st_size;*/

   uint8_t x;
   int p=0;

   ifstream inf;
   inf.open(infile, ifstream::in|ifstream::binary);
```

```
    ofstream outf;
    outf.open(outfile, ofstream::out|ofstream::binary);

    if(!inf) {
        printf("An error occured opening the file!\n");
        return 1;
    }

    char y;

/*    while(p<fsize)
    {*/
    for(int i = 0; i<fsize; i++)
    {
        while(p<fsize)
        {
            inf.read(&y, 1);
            y = char_do_enigma((uint8_t) y);
            outf.write(&y, 1);
            p++;
        }
    }
    //}
    //printf("encrypted/decrypted characters: %d\n", p);
}

/* checks to see if a rotor has made a full revolution
and increments the next rotor */
void enigma::audit_rotors(int currPosition) {
    if(position[currPosition] > 255) {
        position[currPosition] = 0;
        if(currPosition < num_rotors) { //on the last rotor, so we don't increment other rotors
            position[currPosition+1]++;
            audit_rotors(currPosition+1);
        }
    }
    return;
}

uint8_t enigma::rotor_lookup(uint8_t x, int rotor) {
   // printf("rotor lookup, x: %d, rotor: %d\n", x, rotor);
    for(int i=0; i<256; i++) {
        if(rotors[rotor][i] == x)
            return (uint8_t)i;
    }
}
```

```
char enigma::char_do_enigma(uint8_t x) {

   //rotate first rotor by one
     position[0]++;
   //check if any of the other rotors need to be rotated
     audit_rotors(0);

  // printf("initial: %d\n", x);

   uint8_t temp;
   //go through the plug board
   temp = (uint8_t)plugboard[x];

   // printf("plugboard: %d\n", temp);
   //go through the first rotor
   if(rotors[0][(temp+position[0])%256] - position[0] < 0)
       temp = rotors[0][(temp+position[0])%256] - position[0] + 256;
   else
       temp = rotors[0][(temp+position[0])%256] - position[0];

  // printf("rotor 0: %d\n", temp);
   //go through the rest of the rotors
   for(int i=1; i<num_rotors; i++) {
       if(rotors[i][(temp+position[i])%256] - position[i] < 0 )
           temp = rotors[i][(temp+position[i])%256] - position[i] + 256;
       else
           temp = rotors[i][(temp+position[i])%256] - position[i];

   //    printf("rotor %d: %d\n", i, temp);
   }

   //go through relflector
  // printf("before reflector: %d\n", temp);
   temp = reflector[temp];
  // printf("after reflector: %d\n", temp);

   uint8_t temp1;

   for(int i=(num_rotors-1); i>=0; i--) {
       if (rotors_r[i][(temp+position[i])%256] - position[i] < 0)
           temp = rotors_r[i][(temp+position[i])%256] - position[i] + 256;
       else
           temp = rotors_r[i][(temp+position[i])%256] - position[i];
     //  printf("position: %d\n", position[i]);
     //  printf("rotor %d: %d\n", i, temp);
```

```
    }

    //back through plugboard here
    for(int i=0; i<256; i++) {
        if(plugboard[i] == (int)temp)  {
         //   printf("plugboard: %d\n", i);
            return (char) i;
        }
    }
}

/* set up char ** rotors, int * positions, refelector, plugboard, and numrotors
in future, will use galois field class                                    */
void enigma::init_enigma(char * key) {
    num_rotors = 5;

    // allocate memory for rotors
    rotors = new uint8_t*[num_rotors];
        for (int i = 0; i < num_rotors; ++i)
            rotors[i] = new uint8_t[256];

    rotors_r = new uint8_t*[num_rotors];
        for (int i = 0; i < num_rotors; ++i)
            rotors_r[i] = new uint8_t[256];

    position = new int[num_rotors];
    // set the default position to zero
    for(int i=0; i < num_rotors; i++) {
        position[i] = 0;
    }

    // initialize plug board
    for(int i=0; i < 256; i++) {
        plugboard[i] = i;
    }

/*    List * temp_list = NULL;
    List * temp_head;

    temp_list = (List *)malloc(sizeof(List));
    temp_head = temp_list;

    for(int i=0; i<256; i++) {
        temp_list->val = i;
        temp_list->ptr = (List *) malloc(sizeof(List));
        temp_list = temp_list->ptr;
```

```
    }
    int upper_range = 255;
    for(int i=0; i<256; i++){
       upper_range -= i;
       delete_list_entry(temp_head, randvar);
    }*/

    galois g;
    /*srand((unsigned)time(0));
    int random_integer;
    int low=0, high=255;
    int range=(high-low);
    for(int ind=0; ind<256; ind++){
       for(int i=0; i < num_rotors; i++){
          random_integer = low+int(range*rand()/(RAND_MAX + 1.0));
          rotors[i][ind] = (uint8_t)g.sbox(random_integer);
       }
    } */

    vector<int> valueVector;
    for(int i=0; i<256; i++) {
       valueVector.push_back(i);
    }
    sort(valueVector.begin(), valueVector.end());
    for(int i=0; i<num_rotors; i++){
       for(int j=0; j<256; j++) {
          rotors[i][j] = (uint8_t)g.sbox(valueVector[j], key);
          sort(valueVector.begin(), valueVector.end());
       }
    }

/*    rotors[0][0] = 210;
    rotors[0][1] = 128;
    rotors[0][2] = 234;
    rotors[0][3] = 158;
    rotors[0][4] = 63;
    rotors[0][5] = 49;
    rotors[0][6] = 108;
    rotors[0][7] = 135;
    rotors[0][8] = 37;
    rotors[0][9] = 1;
    rotors[0][10] = 189;
    rotors[0][11] = 14;
    rotors[0][12] = 133;
    rotors[0][13] = 141;
    rotors[0][14] = 24;
```

```
rotors[0][15] = 196;
rotors[0][16] = 228;
rotors[0][17] = 64;
rotors[0][18] = 57;
rotors[0][19] = 89;
rotors[0][20] = 243;
rotors[0][21] = 109;
rotors[0][22] = 104;
rotors[0][23] = 0;
rotors[0][24] = 169;
rotors[0][25] = 240;
rotors[0][26] = 72;
rotors[0][27] = 47;
rotors[0][28] = 30;
rotors[0][29] = 6;
rotors[0][30] = 33;
rotors[0][31] = 252;
rotors[0][32] = 54;
rotors[0][33] = 82;
rotors[0][34] = 55;
rotors[0][35] = 183;
rotors[0][36] = 138;
rotors[0][37] = 197;
rotors[0][38] = 81;
rotors[0][39] = 192;
rotors[0][40] = 69;
rotors[0][41] = 245;
rotors[0][42] = 168;
rotors[0][43] = 136;
rotors[0][44] = 23;
rotors[0][45] = 179;
rotors[0][46] = 208;
rotors[0][47] = 140;
rotors[0][48] = 10;
rotors[0][49] = 32;
rotors[0][50] = 78;
rotors[0][51] = 155;
rotors[0][52] = 237;
rotors[0][53] = 130;
rotors[0][54] = 76;
rotors[0][55] = 253;
rotors[0][56] = 94;
rotors[0][57] = 176;
rotors[0][58] = 172;
rotors[0][59] = 62;
rotors[0][60] = 117;
```

```
rotors[0][61] = 44;
rotors[0][62] = 129;
rotors[0][63] = 97;
rotors[0][64] = 255;
rotors[0][65] = 251;
rotors[0][66] = 3;
rotors[0][67] = 166;
rotors[0][68] = 68;
rotors[0][69] = 200;
rotors[0][70] = 191;
rotors[0][71] = 11;
rotors[0][72] = 198;
rotors[0][73] = 71;
rotors[0][74] = 190;
rotors[0][75] = 214;
rotors[0][76] = 48;
rotors[0][77] = 27;
rotors[0][78] = 60;
rotors[0][79] = 142;
rotors[0][80] = 95;
rotors[0][81] = 162;
rotors[0][82] = 131;
rotors[0][83] = 182;
rotors[0][84] = 150;
rotors[0][85] = 38;
rotors[0][86] = 34;
rotors[0][87] = 21;
rotors[0][88] = 120;
rotors[0][89] = 193;
rotors[0][90] = 118;
rotors[0][91] = 149;
rotors[0][92] = 224;
rotors[0][93] = 74;
rotors[0][94] = 107;
rotors[0][95] = 22;
rotors[0][96] = 122;
rotors[0][97] = 80;
rotors[0][98] = 152;
rotors[0][99] = 248;
rotors[0][100] = 20;
rotors[0][101] = 195;
rotors[0][102] = 213;
rotors[0][103] = 5;
rotors[0][104] = 212;
rotors[0][105] = 52;
rotors[0][106] = 205;
```

```
rotors[0][107] = 91;
rotors[0][108] = 8;
rotors[0][109] = 105;
rotors[0][110] = 125;
rotors[0][111] = 25;
rotors[0][112] = 29;
rotors[0][113] = 239;
rotors[0][114] = 15;
rotors[0][115] = 199;
rotors[0][116] = 246;
rotors[0][117] = 99;
rotors[0][118] = 201;
rotors[0][119] = 249;
rotors[0][120] = 222;
rotors[0][121] = 45;
rotors[0][122] = 59;
rotors[0][123] = 229;
rotors[0][124] = 145;
rotors[0][125] = 9;
rotors[0][126] = 84;
rotors[0][127] = 174;
rotors[0][128] = 218;
rotors[0][129] = 180;
rotors[0][130] = 123;
rotors[0][131] = 244;
rotors[0][132] = 170;
rotors[0][133] = 92;
rotors[0][134] = 207;
rotors[0][135] = 236;
rotors[0][136] = 167;
rotors[0][137] = 219;
rotors[0][138] = 75;
rotors[0][139] = 231;
rotors[0][140] = 203;
rotors[0][141] = 230;
rotors[0][142] = 161;
rotors[0][143] = 194;
rotors[0][144] = 42;
rotors[0][145] = 254;
rotors[0][146] = 124;
rotors[0][147] = 16;
rotors[0][148] = 184;
rotors[0][149] = 77;
rotors[0][150] = 223;
rotors[0][151] = 116;
rotors[0][152] = 115;
```

```
rotors[0][153] = 137;
rotors[0][154] = 12;
rotors[0][155] = 148;
rotors[0][156] = 93;
rotors[0][157] = 221;
rotors[0][158] = 50;
rotors[0][159] = 73;
rotors[0][160] = 227;
rotors[0][161] = 41;
rotors[0][162] = 88;
rotors[0][163] = 146;
rotors[0][164] = 70;
rotors[0][165] = 250;
rotors[0][166] = 177;
rotors[0][167] = 119;
rotors[0][168] = 18;
rotors[0][169] = 226;
rotors[0][170] = 160;
rotors[0][171] = 43;
rotors[0][172] = 235;
rotors[0][173] = 206;
rotors[0][174] = 217;
rotors[0][175] = 65;
rotors[0][176] = 36;
rotors[0][177] = 132;
rotors[0][178] = 83;
rotors[0][179] = 225;
rotors[0][180] = 154;
rotors[0][181] = 143;
rotors[0][182] = 4;
rotors[0][183] = 151;
rotors[0][184] = 58;
rotors[0][185] = 238;
rotors[0][186] = 90;
rotors[0][187] = 35;
rotors[0][188] = 178;
rotors[0][189] = 216;
rotors[0][190] = 126;
rotors[0][191] = 106;
rotors[0][192] = 87;
rotors[0][193] = 98;
rotors[0][194] = 53;
rotors[0][195] = 86;
rotors[0][196] = 7;
rotors[0][197] = 220;
rotors[0][198] = 85;
```

```
rotors[0][199] = 102;
rotors[0][200] = 157;
rotors[0][201] = 147;
rotors[0][202] = 139;
rotors[0][203] = 163;
rotors[0][204] = 31;
rotors[0][205] = 67;
rotors[0][206] = 233;
rotors[0][207] = 181;
rotors[0][208] = 153;
rotors[0][209] = 19;
rotors[0][210] = 121;
rotors[0][211] = 127;
rotors[0][212] = 241;
rotors[0][213] = 100;
rotors[0][214] = 46;
rotors[0][215] = 247;
rotors[0][216] = 144;
rotors[0][217] = 110;
rotors[0][218] = 66;
rotors[0][219] = 156;
rotors[0][220] = 209;
rotors[0][221] = 232;
rotors[0][222] = 171;
rotors[0][223] = 13;
rotors[0][224] = 204;
rotors[0][225] = 134;
rotors[0][226] = 79;
rotors[0][227] = 51;
rotors[0][228] = 113;
rotors[0][229] = 114;
rotors[0][230] = 26;
rotors[0][231] = 61;
rotors[0][232] = 40;
rotors[0][233] = 101;
rotors[0][234] = 39;
rotors[0][235] = 202;
rotors[0][236] = 96;
rotors[0][237] = 28;
rotors[0][238] = 215;
rotors[0][239] = 2;
rotors[0][240] = 173;
rotors[0][241] = 111;
rotors[0][242] = 187;
rotors[0][243] = 17;
rotors[0][244] = 103;
```

```
rotors[0][245] = 56;
rotors[0][246] = 159;
rotors[0][247] = 175;
rotors[0][248] = 188;
rotors[0][249] = 185;
rotors[0][250] = 211;
rotors[0][251] = 112;
rotors[0][252] = 242;
rotors[0][253] = 186;
rotors[0][254] = 165;
rotors[0][255] = 164;

rotors[1][0] = 105;
rotors[1][1] = 91;
rotors[1][2] = 6;
rotors[1][3] = 47;
rotors[1][4] = 225;
rotors[1][5] = 121;
rotors[1][6] = 130;
rotors[1][7] = 132;
rotors[1][8] = 20;
rotors[1][9] = 217;
rotors[1][10] = 39;
rotors[1][11] = 50;
rotors[1][12] = 18;
rotors[1][13] = 82;
rotors[1][14] = 252;
rotors[1][15] = 106;
rotors[1][16] = 96;
rotors[1][17] = 139;
rotors[1][18] = 114;
rotors[1][19] = 231;
rotors[1][20] = 138;
rotors[1][21] = 45;
rotors[1][22] = 194;
rotors[1][23] = 84;
rotors[1][24] = 254;
rotors[1][25] = 163;
rotors[1][26] = 206;
rotors[1][27] = 111;
rotors[1][28] = 193;
rotors[1][29] = 171;
rotors[1][30] = 230;
rotors[1][31] = 54;
rotors[1][32] = 58;
rotors[1][33] = 0;
```

```
rotors[1][34] = 61;
rotors[1][35] = 63;
rotors[1][36] = 233;
rotors[1][37] = 123;
rotors[1][38] = 154;
rotors[1][39] = 158;
rotors[1][40] = 191;
rotors[1][41] = 151;
rotors[1][42] = 44;
rotors[1][43] = 211;
rotors[1][44] = 178;
rotors[1][45] = 208;
rotors[1][46] = 15;
rotors[1][47] = 107;
rotors[1][48] = 55;
rotors[1][49] = 8;
rotors[1][50] = 34;
rotors[1][51] = 156;
rotors[1][52] = 159;
rotors[1][53] = 180;
rotors[1][54] = 155;
rotors[1][55] = 95;
rotors[1][56] = 67;
rotors[1][57] = 125;
rotors[1][58] = 69;
rotors[1][59] = 161;
rotors[1][60] = 160;
rotors[1][61] = 182;
rotors[1][62] = 77;
rotors[1][63] = 24;
rotors[1][64] = 246;
rotors[1][65] = 94;
rotors[1][66] = 209;
rotors[1][67] = 229;
rotors[1][68] = 108;
rotors[1][69] = 166;
rotors[1][70] = 57;
rotors[1][71] = 253;
rotors[1][72] = 174;
rotors[1][73] = 175;
rotors[1][74] = 97;
rotors[1][75] = 157;
rotors[1][76] = 212;
rotors[1][77] = 185;
rotors[1][78] = 127;
rotors[1][79] = 135;
```

```
rotors[1][80] = 237;
rotors[1][81] = 117;
rotors[1][82] = 238;
rotors[1][83] = 64;
rotors[1][84] = 140;
rotors[1][85] = 71;
rotors[1][86] = 234;
rotors[1][87] = 112;
rotors[1][88] = 137;
rotors[1][89] = 207;
rotors[1][90] = 216;
rotors[1][91] = 72;
rotors[1][92] = 199;
rotors[1][93] = 176;
rotors[1][94] = 33;
rotors[1][95] = 103;
rotors[1][96] = 23;
rotors[1][97] = 235;
rotors[1][98] = 188;
rotors[1][99] = 201;
rotors[1][100] = 131;
rotors[1][101] = 46;
rotors[1][102] = 43;
rotors[1][103] = 28;
rotors[1][104] = 198;
rotors[1][105] = 10;
rotors[1][106] = 192;
rotors[1][107] = 60;
rotors[1][108] = 150;
rotors[1][109] = 204;
rotors[1][110] = 169;
rotors[1][111] = 215;
rotors[1][112] = 243;
rotors[1][113] = 153;
rotors[1][114] = 165;
rotors[1][115] = 239;
rotors[1][116] = 25;
rotors[1][117] = 152;
rotors[1][118] = 5;
rotors[1][119] = 214;
rotors[1][120] = 42;
rotors[1][121] = 29;
rotors[1][122] = 136;
rotors[1][123] = 134;
rotors[1][124] = 21;
rotors[1][125] = 74;
```

```
rotors[1][126] = 242;
rotors[1][127] = 40;
rotors[1][128] = 41;
rotors[1][129] = 70;
rotors[1][130] = 86;
rotors[1][131] = 200;
rotors[1][132] = 126;
rotors[1][133] = 116;
rotors[1][134] = 27;
rotors[1][135] = 31;
rotors[1][136] = 218;
rotors[1][137] = 35;
rotors[1][138] = 14;
rotors[1][139] = 32;
rotors[1][140] = 12;
rotors[1][141] = 62;
rotors[1][142] = 220;
rotors[1][143] = 1;
rotors[1][144] = 251;
rotors[1][145] = 120;
rotors[1][146] = 22;
rotors[1][147] = 232;
rotors[1][148] = 245;
rotors[1][149] = 109;
rotors[1][150] = 142;
rotors[1][151] = 37;
rotors[1][152] = 236;
rotors[1][153] = 66;
rotors[1][154] = 202;
rotors[1][155] = 247;
rotors[1][156] = 196;
rotors[1][157] = 187;
rotors[1][158] = 181;
rotors[1][159] = 3;
rotors[1][160] = 19;
rotors[1][161] = 53;
rotors[1][162] = 190;
rotors[1][163] = 168;
rotors[1][164] = 226;
rotors[1][165] = 222;
rotors[1][166] = 224;
rotors[1][167] = 255;
rotors[1][168] = 48;
rotors[1][169] = 90;
rotors[1][170] = 30;
rotors[1][171] = 219;
```

```
rotors[1][172] = 227;
rotors[1][173] = 241;
rotors[1][174] = 75;
rotors[1][175] = 38;
rotors[1][176] = 7;
rotors[1][177] = 13;
rotors[1][178] = 11;
rotors[1][179] = 210;
rotors[1][180] = 98;
rotors[1][181] = 79;
rotors[1][182] = 141;
rotors[1][183] = 183;
rotors[1][184] = 143;
rotors[1][185] = 80;
rotors[1][186] = 223;
rotors[1][187] = 197;
rotors[1][188] = 244;
rotors[1][189] = 203;
rotors[1][190] = 83;
rotors[1][191] = 170;
rotors[1][192] = 133;
rotors[1][193] = 177;
rotors[1][194] = 167;
rotors[1][195] = 93;
rotors[1][196] = 240;
rotors[1][197] = 9;
rotors[1][198] = 36;
rotors[1][199] = 164;
rotors[1][200] = 189;
rotors[1][201] = 248;
rotors[1][202] = 102;
rotors[1][203] = 81;
rotors[1][204] = 124;
rotors[1][205] = 149;
rotors[1][206] = 184;
rotors[1][207] = 104;
rotors[1][208] = 122;
rotors[1][209] = 92;
rotors[1][210] = 110;
rotors[1][211] = 99;
rotors[1][212] = 147;
rotors[1][213] = 186;
rotors[1][214] = 26;
rotors[1][215] = 89;
rotors[1][216] = 100;
rotors[1][217] = 146;
```

```
rotors[1][218] = 205;
rotors[1][219] = 51;
rotors[1][220] = 195;
rotors[1][221] = 87;
rotors[1][222] = 179;
rotors[1][223] = 59;
rotors[1][224] = 221;
rotors[1][225] = 249;
rotors[1][226] = 162;
rotors[1][227] = 73;
rotors[1][228] = 56;
rotors[1][229] = 52;
rotors[1][230] = 16;
rotors[1][231] = 49;
rotors[1][232] = 88;
rotors[1][233] = 76;
rotors[1][234] = 85;
rotors[1][235] = 173;
rotors[1][236] = 172;
rotors[1][237] = 118;
rotors[1][238] = 128;
rotors[1][239] = 65;
rotors[1][240] = 113;
rotors[1][241] = 228;
rotors[1][242] = 148;
rotors[1][243] = 144;
rotors[1][244] = 17;
rotors[1][245] = 2;
rotors[1][246] = 78;
rotors[1][247] = 129;
rotors[1][248] = 68;
rotors[1][249] = 250;
rotors[1][250] = 4;
rotors[1][251] = 213;
rotors[1][252] = 145;
rotors[1][253] = 101;
rotors[1][254] = 115;
rotors[1][255] = 119;


rotors[2][0] = 110;
rotors[2][1] = 156;
rotors[2][2] = 28;
rotors[2][3] = 76;
rotors[2][4] = 219;
rotors[2][5] = 64;
```

```
rotors[2][6] = 68;
rotors[2][7] = 45;
rotors[2][8] = 52;
rotors[2][9] = 16;
rotors[2][10] = 65;
rotors[2][11] = 20;
rotors[2][12] = 224;
rotors[2][13] = 69;
rotors[2][14] = 139;
rotors[2][15] = 220;
rotors[2][16] = 50;
rotors[2][17] = 61;
rotors[2][18] = 172;
rotors[2][19] = 164;
rotors[2][20] = 107;
rotors[2][21] = 83;
rotors[2][22] = 41;
rotors[2][23] = 96;
rotors[2][24] = 222;
rotors[2][25] = 209;
rotors[2][26] = 6;
rotors[2][27] = 147;
rotors[2][28] = 128;
rotors[2][29] = 66;
rotors[2][30] = 163;
rotors[2][31] = 194;
rotors[2][32] = 141;
rotors[2][33] = 175;
rotors[2][34] = 217;
rotors[2][35] = 158;
rotors[2][36] = 11;
rotors[2][37] = 202;
rotors[2][38] = 98;
rotors[2][39] = 30;
rotors[2][40] = 82;
rotors[2][41] = 255;
rotors[2][42] = 113;
rotors[2][43] = 47;
rotors[2][44] = 176;
rotors[2][45] = 106;
rotors[2][46] = 9;
rotors[2][47] = 170;
rotors[2][48] = 36;
rotors[2][49] = 78;
rotors[2][50] = 225;
rotors[2][51] = 135;
```

```
rotors[2][52] = 143;
rotors[2][53] = 149;
rotors[2][54] = 241;
rotors[2][55] = 226;
rotors[2][56] = 168;
rotors[2][57] = 117;
rotors[2][58] = 253;
rotors[2][59] = 125;
rotors[2][60] = 13;
rotors[2][61] = 191;
rotors[2][62] = 137;
rotors[2][63] = 185;
rotors[2][64] = 54;
rotors[2][65] = 88;
rotors[2][66] = 212;
rotors[2][67] = 5;
rotors[2][68] = 229;
rotors[2][69] = 236;
rotors[2][70] = 24;
rotors[2][71] = 92;
rotors[2][72] = 144;
rotors[2][73] = 109;
rotors[2][74] = 70;
rotors[2][75] = 245;
rotors[2][76] = 160;
rotors[2][77] = 10;
rotors[2][78] = 130;
rotors[2][79] = 35;
rotors[2][80] = 166;
rotors[2][81] = 33;
rotors[2][82] = 204;
rotors[2][83] = 213;
rotors[2][84] = 171;
rotors[2][85] = 215;
rotors[2][86] = 40;
rotors[2][87] = 197;
rotors[2][88] = 243;
rotors[2][89] = 60;
rotors[2][90] = 75;
rotors[2][91] = 238;
rotors[2][92] = 132;
rotors[2][93] = 37;
rotors[2][94] = 232;
rotors[2][95] = 3;
rotors[2][96] = 173;
rotors[2][97] = 169;
```

```
rotors[2][98] = 42;
rotors[2][99] = 104;
rotors[2][100] = 208;
rotors[2][101] = 93;
rotors[2][102] = 57;
rotors[2][103] = 205;
rotors[2][104] = 239;
rotors[2][105] = 99;
rotors[2][106] = 91;
rotors[2][107] = 1;
rotors[2][108] = 89;
rotors[2][109] = 116;
rotors[2][110] = 223;
rotors[2][111] = 127;
rotors[2][112] = 85;
rotors[2][113] = 72;
rotors[2][114] = 58;
rotors[2][115] = 56;
rotors[2][116] = 2;
rotors[2][117] = 12;
rotors[2][118] = 221;
rotors[2][119] = 235;
rotors[2][120] = 31;
rotors[2][121] = 26;
rotors[2][122] = 162;
rotors[2][123] = 115;
rotors[2][124] = 201;
rotors[2][125] = 247;
rotors[2][126] = 123;
rotors[2][127] = 246;
rotors[2][128] = 155;
rotors[2][129] = 7;
rotors[2][130] = 59;
rotors[2][131] = 34;
rotors[2][132] = 177;
rotors[2][133] = 183;
rotors[2][134] = 112;
rotors[2][135] = 227;
rotors[2][136] = 25;
rotors[2][137] = 18;
rotors[2][138] = 74;
rotors[2][139] = 62;
rotors[2][140] = 187;
rotors[2][141] = 79;
rotors[2][142] = 174;
rotors[2][143] = 242;
```

```
rotors[2][144] = 27;
rotors[2][145] = 190;
rotors[2][146] = 120;
rotors[2][147] = 86;
rotors[2][148] = 95;
rotors[2][149] = 134;
rotors[2][150] = 152;
rotors[2][151] = 138;
rotors[2][152] = 203;
rotors[2][153] = 251;
rotors[2][154] = 14;
rotors[2][155] = 153;
rotors[2][156] = 73;
rotors[2][157] = 8;
rotors[2][158] = 121;
rotors[2][159] = 184;
rotors[2][160] = 87;
rotors[2][161] = 233;
rotors[2][162] = 179;
rotors[2][163] = 55;
rotors[2][164] = 38;
rotors[2][165] = 122;
rotors[2][166] = 136;
rotors[2][167] = 19;
rotors[2][168] = 81;
rotors[2][169] = 230;
rotors[2][170] = 186;
rotors[2][171] = 111;
rotors[2][172] = 165;
rotors[2][173] = 100;
rotors[2][174] = 193;
rotors[2][175] = 198;
rotors[2][176] = 228;
rotors[2][177] = 211;
rotors[2][178] = 102;
rotors[2][179] = 108;
rotors[2][180] = 157;
rotors[2][181] = 118;
rotors[2][182] = 44;
rotors[2][183] = 161;
rotors[2][184] = 200;
rotors[2][185] = 240;
rotors[2][186] = 67;
rotors[2][187] = 4;
rotors[2][188] = 129;
rotors[2][189] = 167;
```

```
rotors[2][190] = 124;
rotors[2][191] = 80;
rotors[2][192] = 206;
rotors[2][193] = 237;
rotors[2][194] = 32;
rotors[2][195] = 105;
rotors[2][196] = 97;
rotors[2][197] = 133;
rotors[2][198] = 182;
rotors[2][199] = 140;
rotors[2][200] = 94;
rotors[2][201] = 63;
rotors[2][202] = 90;
rotors[2][203] = 151;
rotors[2][204] = 71;
rotors[2][205] = 43;
rotors[2][206] = 196;
rotors[2][207] = 154;
rotors[2][208] = 0;
rotors[2][209] = 195;
rotors[2][210] = 29;
rotors[2][211] = 103;
rotors[2][212] = 84;
rotors[2][213] = 114;
rotors[2][214] = 51;
rotors[2][215] = 146;
rotors[2][216] = 189;
rotors[2][217] = 148;
rotors[2][218] = 249;
rotors[2][219] = 192;
rotors[2][220] = 188;
rotors[2][221] = 39;
rotors[2][222] = 216;
rotors[2][223] = 250;
rotors[2][224] = 46;
rotors[2][225] = 178;
rotors[2][226] = 159;
rotors[2][227] = 15;
rotors[2][228] = 150;
rotors[2][229] = 48;
rotors[2][230] = 207;
rotors[2][231] = 244;
rotors[2][232] = 254;
rotors[2][233] = 23;
rotors[2][234] = 180;
rotors[2][235] = 53;
```

```
rotors[2][236] = 119;
rotors[2][237] = 218;
rotors[2][238] = 234;
rotors[2][239] = 231;
rotors[2][240] = 126;
rotors[2][241] = 101;
rotors[2][242] = 252;
rotors[2][243] = 248;
rotors[2][244] = 77;
rotors[2][245] = 145;
rotors[2][246] = 142;
rotors[2][247] = 214;
rotors[2][248] = 199;
rotors[2][249] = 17;
rotors[2][250] = 49;
rotors[2][251] = 131;
rotors[2][252] = 22;
rotors[2][253] = 181;
rotors[2][254] = 21;
rotors[2][255] = 210;

rotors[3][0] = 42;
rotors[3][1] = 157;
rotors[3][2] = 81;
rotors[3][3] = 34;
rotors[3][4] = 151;
rotors[3][5] = 255;
rotors[3][6] = 13;
rotors[3][7] = 146;
rotors[3][8] = 4;
rotors[3][9] = 118;
rotors[3][10] = 49;
rotors[3][11] = 138;
rotors[3][12] = 193;
rotors[3][13] = 218;
rotors[3][14] = 182;
rotors[3][15] = 205;
rotors[3][16] = 24;
rotors[3][17] = 227;
rotors[3][18] = 0;
rotors[3][19] = 38;
rotors[3][20] = 124;
rotors[3][21] = 140;
rotors[3][22] = 112;
rotors[3][23] = 2;
rotors[3][24] = 128;
```

```
rotors[3][25] = 211;
rotors[3][26] = 44;
rotors[3][27] = 115;
rotors[3][28] = 125;
rotors[3][29] = 224;
rotors[3][30] = 212;
rotors[3][31] = 131;
rotors[3][32] = 98;
rotors[3][33] = 221;
rotors[3][34] = 119;
rotors[3][35] = 36;
rotors[3][36] = 78;
rotors[3][37] = 248;
rotors[3][38] = 135;
rotors[3][39] = 199;
rotors[3][40] = 175;
rotors[3][41] = 84;
rotors[3][42] = 110;
rotors[3][43] = 23;
rotors[3][44] = 53;
rotors[3][45] = 147;
rotors[3][46] = 139;
rotors[3][47] = 165;
rotors[3][48] = 8;
rotors[3][49] = 89;
rotors[3][50] = 83;
rotors[3][51] = 102;
rotors[3][52] = 169;
rotors[3][53] = 172;
rotors[3][54] = 141;
rotors[3][55] = 203;
rotors[3][56] = 51;
rotors[3][57] = 148;
rotors[3][58] = 149;
rotors[3][59] = 69;
rotors[3][60] = 136;
rotors[3][61] = 113;
rotors[3][62] = 22;
rotors[3][63] = 194;
rotors[3][64] = 26;
rotors[3][65] = 170;
rotors[3][66] = 95;
rotors[3][67] = 241;
rotors[3][68] = 233;
rotors[3][69] = 226;
rotors[3][70] = 214;
```

```
rotors[3][71] = 127;
rotors[3][72] = 80;
rotors[3][73] = 196;
rotors[3][74] = 20;
rotors[3][75] = 191;
rotors[3][76] = 77;
rotors[3][77] = 52;
rotors[3][78] = 10;
rotors[3][79] = 1;
rotors[3][80] = 30;
rotors[3][81] = 55;
rotors[3][82] = 70;
rotors[3][83] = 15;
rotors[3][84] = 14;
rotors[3][85] = 156;
rotors[3][86] = 39;
rotors[3][87] = 5;
rotors[3][88] = 143;
rotors[3][89] = 152;
rotors[3][90] = 121;
rotors[3][91] = 60;
rotors[3][92] = 230;
rotors[3][93] = 82;
rotors[3][94] = 177;
rotors[3][95] = 243;
rotors[3][96] = 3;
rotors[3][97] = 126;
rotors[3][98] = 249;
rotors[3][99] = 176;
rotors[3][100] = 90;
rotors[3][101] = 183;
rotors[3][102] = 64;
rotors[3][103] = 29;
rotors[3][104] = 47;
rotors[3][105] = 56;
rotors[3][106] = 216;
rotors[3][107] = 239;
rotors[3][108] = 185;
rotors[3][109] = 65;
rotors[3][110] = 123;
rotors[3][111] = 134;
rotors[3][112] = 76;
rotors[3][113] = 236;
rotors[3][114] = 21;
rotors[3][115] = 137;
rotors[3][116] = 144;
```

```
rotors[3][117] = 96;
rotors[3][118] = 88;
rotors[3][119] = 242;
rotors[3][120] = 195;
rotors[3][121] = 79;
rotors[3][122] = 86;
rotors[3][123] = 179;
rotors[3][124] = 130;
rotors[3][125] = 237;
rotors[3][126] = 247;
rotors[3][127] = 186;
rotors[3][128] = 108;
rotors[3][129] = 57;
rotors[3][130] = 43;
rotors[3][131] = 197;
rotors[3][132] = 66;
rotors[3][133] = 253;
rotors[3][134] = 181;
rotors[3][135] = 215;
rotors[3][136] = 168;
rotors[3][137] = 229;
rotors[3][138] = 209;
rotors[3][139] = 198;
rotors[3][140] = 37;
rotors[3][141] = 178;
rotors[3][142] = 101;
rotors[3][143] = 220;
rotors[3][144] = 234;
rotors[3][145] = 63;
rotors[3][146] = 32;
rotors[3][147] = 97;
rotors[3][148] = 189;
rotors[3][149] = 50;
rotors[3][150] = 120;
rotors[3][151] = 129;
rotors[3][152] = 153;
rotors[3][153] = 155;
rotors[3][154] = 35;
rotors[3][155] = 244;
rotors[3][156] = 18;
rotors[3][157] = 114;
rotors[3][158] = 100;
rotors[3][159] = 93;
rotors[3][160] = 206;
rotors[3][161] = 61;
rotors[3][162] = 200;
```

```
rotors[3][163] = 75;
rotors[3][164] = 62;
rotors[3][165] = 9;
rotors[3][166] = 59;
rotors[3][167] = 11;
rotors[3][168] = 254;
rotors[3][169] = 240;
rotors[3][170] = 25;
rotors[3][171] = 161;
rotors[3][172] = 91;
rotors[3][173] = 87;
rotors[3][174] = 12;
rotors[3][175] = 246;
rotors[3][176] = 173;
rotors[3][177] = 158;
rotors[3][178] = 150;
rotors[3][179] = 204;
rotors[3][180] = 67;
rotors[3][181] = 188;
rotors[3][182] = 111;
rotors[3][183] = 238;
rotors[3][184] = 99;
rotors[3][185] = 154;
rotors[3][186] = 251;
rotors[3][187] = 164;
rotors[3][188] = 104;
rotors[3][189] = 162;
rotors[3][190] = 223;
rotors[3][191] = 207;
rotors[3][192] = 73;
rotors[3][193] = 28;
rotors[3][194] = 17;
rotors[3][195] = 201;
rotors[3][196] = 68;
rotors[3][197] = 159;
rotors[3][198] = 31;
rotors[3][199] = 192;
rotors[3][200] = 122;
rotors[3][201] = 94;
rotors[3][202] = 167;
rotors[3][203] = 19;
rotors[3][204] = 54;
rotors[3][205] = 225;
rotors[3][206] = 231;
rotors[3][207] = 107;
rotors[3][208] = 202;
```

```
rotors[3][209] = 213;
rotors[3][210] = 92;
rotors[3][211] = 235;
rotors[3][212] = 103;
rotors[3][213] = 116;
rotors[3][214] = 16;
rotors[3][215] = 142;
rotors[3][216] = 27;
rotors[3][217] = 160;
rotors[3][218] = 45;
rotors[3][219] = 145;
rotors[3][220] = 228;
rotors[3][221] = 109;
rotors[3][222] = 166;
rotors[3][223] = 48;
rotors[3][224] = 250;
rotors[3][225] = 58;
rotors[3][226] = 6;
rotors[3][227] = 219;
rotors[3][228] = 85;
rotors[3][229] = 222;
rotors[3][230] = 133;
rotors[3][231] = 232;
rotors[3][232] = 33;
rotors[3][233] = 252;
rotors[3][234] = 210;
rotors[3][235] = 72;
rotors[3][236] = 208;
rotors[3][237] = 41;
rotors[3][238] = 117;
rotors[3][239] = 245;
rotors[3][240] = 46;
rotors[3][241] = 7;
rotors[3][242] = 105;
rotors[3][243] = 187;
rotors[3][244] = 190;
rotors[3][245] = 180;
rotors[3][246] = 217;
rotors[3][247] = 132;
rotors[3][248] = 174;
rotors[3][249] = 71;
rotors[3][250] = 40;
rotors[3][251] = 163;
rotors[3][252] = 74;
rotors[3][253] = 106;
rotors[3][254] = 184;
```

```
rotors[3][255] = 171;

rotors[4][0] = 35;
rotors[4][1] = 175;
rotors[4][2] = 205;
rotors[4][3] = 178;
rotors[4][4] = 28;
rotors[4][5] = 164;
rotors[4][6] = 191;
rotors[4][7] = 159;
rotors[4][8] = 171;
rotors[4][9] = 9;
rotors[4][10] = 26;
rotors[4][11] = 233;
rotors[4][12] = 210;
rotors[4][13] = 160;
rotors[4][14] = 0;
rotors[4][15] = 95;
rotors[4][16] = 114;
rotors[4][17] = 249;
rotors[4][18] = 150;
rotors[4][19] = 85;
rotors[4][20] = 199;
rotors[4][21] = 223;
rotors[4][22] = 173;
rotors[4][23] = 182;
rotors[4][24] = 23;
rotors[4][25] = 1;
rotors[4][26] = 64;
rotors[4][27] = 84;
rotors[4][28] = 112;
rotors[4][29] = 128;
rotors[4][30] = 153;
rotors[4][31] = 20;
rotors[4][32] = 200;
rotors[4][33] = 251;
rotors[4][34] = 49;
rotors[4][35] = 82;
rotors[4][36] = 148;
rotors[4][37] = 103;
rotors[4][38] = 69;
rotors[4][39] = 13;
rotors[4][40] = 192;
rotors[4][41] = 67;
rotors[4][42] = 16;
rotors[4][43] = 96;
```

```
rotors[4][44] = 252;
rotors[4][45] = 176;
rotors[4][46] = 177;
rotors[4][47] = 131;
rotors[4][48] = 89;
rotors[4][49] = 187;
rotors[4][50] = 107;
rotors[4][51] = 196;
rotors[4][52] = 108;
rotors[4][53] = 38;
rotors[4][54] = 40;
rotors[4][55] = 237;
rotors[4][56] = 117;
rotors[4][57] = 79;
rotors[4][58] = 124;
rotors[4][59] = 231;
rotors[4][60] = 149;
rotors[4][61] = 116;
rotors[4][62] = 215;
rotors[4][63] = 47;
rotors[4][64] = 140;
rotors[4][65] = 46;
rotors[4][66] = 180;
rotors[4][67] = 105;
rotors[4][68] = 110;
rotors[4][69] = 94;
rotors[4][70] = 184;
rotors[4][71] = 80;
rotors[4][72] = 104;
rotors[4][73] = 156;
rotors[4][74] = 122;
rotors[4][75] = 90;
rotors[4][76] = 98;
rotors[4][77] = 3;
rotors[4][78] = 87;
rotors[4][79] = 119;
rotors[4][80] = 211;
rotors[4][81] = 224;
rotors[4][82] = 181;
rotors[4][83] = 168;
rotors[4][84] = 142;
rotors[4][85] = 75;
rotors[4][86] = 92;
rotors[4][87] = 220;
rotors[4][88] = 5;
rotors[4][89] = 100;
```

```
rotors[4][90] = 226;
rotors[4][91] = 7;
rotors[4][92] = 240;
rotors[4][93] = 14;
rotors[4][94] = 158;
rotors[4][95] = 166;
rotors[4][96] = 74;
rotors[4][97] = 130;
rotors[4][98] = 137;
rotors[4][99] = 154;
rotors[4][100] = 81;
rotors[4][101] = 44;
rotors[4][102] = 51;
rotors[4][103] = 58;
rotors[4][104] = 101;
rotors[4][105] = 11;
rotors[4][106] = 254;
rotors[4][107] = 144;
rotors[4][108] = 121;
rotors[4][109] = 246;
rotors[4][110] = 195;
rotors[4][111] = 188;
rotors[4][112] = 118;
rotors[4][113] = 27;
rotors[4][114] = 17;
rotors[4][115] = 97;
rotors[4][116] = 247;
rotors[4][117] = 228;
rotors[4][118] = 190;
rotors[4][119] = 127;
rotors[4][120] = 193;
rotors[4][121] = 155;
rotors[4][122] = 163;
rotors[4][123] = 55;
rotors[4][124] = 151;
rotors[4][125] = 113;
rotors[4][126] = 32;
rotors[4][127] = 18;
rotors[4][128] = 31;
rotors[4][129] = 229;
rotors[4][130] = 37;
rotors[4][131] = 8;
rotors[4][132] = 172;
rotors[4][133] = 133;
rotors[4][134] = 93;
rotors[4][135] = 62;
```

```
rotors[4][136] = 136;
rotors[4][137] = 77;
rotors[4][138] = 244;
rotors[4][139] = 73;
rotors[4][140] = 66;
rotors[4][141] = 147;
rotors[4][142] = 143;
rotors[4][143] = 208;
rotors[4][144] = 33;
rotors[4][145] = 42;
rotors[4][146] = 102;
rotors[4][147] = 53;
rotors[4][148] = 204;
rotors[4][149] = 167;
rotors[4][150] = 236;
rotors[4][151] = 59;
rotors[4][152] = 145;
rotors[4][153] = 162;
rotors[4][154] = 125;
rotors[4][155] = 91;
rotors[4][156] = 76;
rotors[4][157] = 198;
rotors[4][158] = 52;
rotors[4][159] = 120;
rotors[4][160] = 65;
rotors[4][161] = 161;
rotors[4][162] = 174;
rotors[4][163] = 48;
rotors[4][164] = 25;
rotors[4][165] = 45;
rotors[4][166] = 183;
rotors[4][167] = 30;
rotors[4][168] = 115;
rotors[4][169] = 135;
rotors[4][170] = 141;
rotors[4][171] = 255;
rotors[4][172] = 68;
rotors[4][173] = 213;
rotors[4][174] = 203;
rotors[4][175] = 238;
rotors[4][176] = 6;
rotors[4][177] = 56;
rotors[4][178] = 43;
rotors[4][179] = 235;
rotors[4][180] = 21;
rotors[4][181] = 88;
```

```
rotors[4][182] = 241;
rotors[4][183] = 22;
rotors[4][184] = 197;
rotors[4][185] = 218;
rotors[4][186] = 24;
rotors[4][187] = 222;
rotors[4][188] = 36;
rotors[4][189] = 57;
rotors[4][190] = 209;
rotors[4][191] = 248;
rotors[4][192] = 86;
rotors[4][193] = 170;
rotors[4][194] = 169;
rotors[4][195] = 253;
rotors[4][196] = 217;
rotors[4][197] = 132;
rotors[4][198] = 216;
rotors[4][199] = 70;
rotors[4][200] = 219;
rotors[4][201] = 19;
rotors[4][202] = 12;
rotors[4][203] = 63;
rotors[4][204] = 179;
rotors[4][205] = 243;
rotors[4][206] = 10;
rotors[4][207] = 123;
rotors[4][208] = 225;
rotors[4][209] = 129;
rotors[4][210] = 15;
rotors[4][211] = 202;
rotors[4][212] = 138;
rotors[4][213] = 152;
rotors[4][214] = 111;
rotors[4][215] = 139;
rotors[4][216] = 194;
rotors[4][217] = 234;
rotors[4][218] = 232;
rotors[4][219] = 157;
rotors[4][220] = 71;
rotors[4][221] = 206;
rotors[4][222] = 212;
rotors[4][223] = 165;
rotors[4][224] = 186;
rotors[4][225] = 185;
rotors[4][226] = 99;
rotors[4][227] = 60;
```

```
rotors[4][228] = 239;
rotors[4][229] = 201;
rotors[4][230] = 41;
rotors[4][231] = 78;
rotors[4][232] = 126;
rotors[4][233] = 146;
rotors[4][234] = 207;
rotors[4][235] = 109;
rotors[4][236] = 29;
rotors[4][237] = 250;
rotors[4][238] = 34;
rotors[4][239] = 214;
rotors[4][240] = 227;
rotors[4][241] = 39;
rotors[4][242] = 4;
rotors[4][243] = 61;
rotors[4][244] = 134;
rotors[4][245] = 72;
rotors[4][246] = 221;
rotors[4][247] = 245;
rotors[4][248] = 83;
rotors[4][249] = 2;
rotors[4][250] = 54;
rotors[4][251] = 106;
rotors[4][252] = 50;
rotors[4][253] = 242;
rotors[4][254] = 189;
rotors[4][255] = 230;*/

reflector[244] = 197;
reflector[197] = 244;
reflector[8] = 44;
reflector[44] = 8;
reflector[110] = 143;
reflector[143] = 110;
reflector[14] = 28;
reflector[28] = 14;
reflector[73] = 249;
reflector[249] = 73;
reflector[10] = 33;
reflector[33] = 10;
reflector[208] = 108;
reflector[108] = 208;
reflector[58] = 60;
reflector[60] = 58;
reflector[49] = 196;
```

```
reflector[196] = 49;
reflector[81] = 112;
reflector[112] = 81;
reflector[69] = 203;
reflector[203] = 69;
reflector[166] = 107;
reflector[107] = 166;
reflector[39] = 210;
reflector[210] = 39;
reflector[158] = 47;
reflector[47] = 158;
reflector[141] = 195;
reflector[195] = 141;
reflector[230] = 4;
reflector[4] = 230;
reflector[226] = 202;
reflector[202] = 226;
reflector[79] = 101;
reflector[101] = 79;
reflector[214] = 183;
reflector[183] = 214;
reflector[144] = 78;
reflector[78] = 144;
reflector[153] = 128;
reflector[128] = 153;
reflector[67] = 218;
reflector[218] = 67;
reflector[125] = 38;
reflector[38] = 125;
reflector[240] = 206;
reflector[206] = 240;
reflector[66] = 199;
reflector[199] = 66;
reflector[64] = 19;
reflector[19] = 64;
reflector[190] = 181;
reflector[181] = 190;
reflector[94] = 160;
reflector[160] = 94;
reflector[135] = 104;
reflector[104] = 135;
reflector[146] = 17;
reflector[17] = 146;
reflector[194] = 98;
reflector[98] = 194;
reflector[2] = 75;
```

```
reflector[75] = 2;
reflector[31] = 11;
reflector[11] = 31;
reflector[85] = 173;
reflector[173] = 85;
reflector[61] = 165;
reflector[165] = 61;
reflector[56] = 191;
reflector[191] = 56;
reflector[111] = 155;
reflector[155] = 111;
reflector[204] = 20;
reflector[20] = 204;
reflector[157] = 142;
reflector[142] = 157;
reflector[253] = 167;
reflector[167] = 253;
reflector[95] = 48;
reflector[48] = 95;
reflector[216] = 187;
reflector[187] = 216;
reflector[201] = 26;
reflector[26] = 201;
reflector[90] = 5;
reflector[5] = 90;
reflector[96] = 0;
reflector[0] = 96;
reflector[46] = 232;
reflector[232] = 46;
reflector[136] = 186;
reflector[186] = 136;
reflector[178] = 145;
reflector[145] = 178;
reflector[1] = 37;
reflector[37] = 1;
reflector[6] = 164;
reflector[164] = 6;
reflector[102] = 93;
reflector[93] = 102;
reflector[92] = 42;
reflector[42] = 92;
reflector[15] = 35;
reflector[35] = 15;
reflector[13] = 252;
reflector[252] = 13;
reflector[156] = 62;
```

```
reflector[62] = 156;
reflector[236] = 71;
reflector[71] = 236;
reflector[222] = 137;
reflector[137] = 222;
reflector[255] = 53;
reflector[53] = 255;
reflector[239] = 116;
reflector[116] = 239;
reflector[113] = 171;
reflector[171] = 113;
reflector[233] = 224;
reflector[224] = 233;
reflector[228] = 77;
reflector[77] = 228;
reflector[227] = 68;
reflector[68] = 227;
reflector[88] = 51;
reflector[51] = 88;
reflector[25] = 86;
reflector[86] = 25;
reflector[127] = 231;
reflector[231] = 127;
reflector[161] = 65;
reflector[65] = 161;
reflector[105] = 243;
reflector[243] = 105;
reflector[205] = 76;
reflector[76] = 205;
reflector[251] = 242;
reflector[242] = 251;
reflector[114] = 134;
reflector[134] = 114;
reflector[177] = 22;
reflector[22] = 177;
reflector[12] = 219;
reflector[219] = 12;
reflector[225] = 246;
reflector[246] = 225;
reflector[234] = 54;
reflector[54] = 234;
reflector[151] = 211;
reflector[211] = 151;
reflector[121] = 217;
reflector[217] = 121;
reflector[106] = 82;
```

```
reflector[82] = 106;
reflector[192] = 83;
reflector[83] = 192;
reflector[229] = 180;
reflector[180] = 229;
reflector[45] = 198;
reflector[198] = 45;
reflector[148] = 34;
reflector[34] = 148;
reflector[29] = 152;
reflector[152] = 29;
reflector[179] = 213;
reflector[213] = 179;
reflector[103] = 89;
reflector[89] = 103;
reflector[184] = 175;
reflector[175] = 184;
reflector[18] = 122;
reflector[122] = 18;
reflector[237] = 223;
reflector[223] = 237;
reflector[80] = 55;
reflector[55] = 80;
reflector[30] = 139;
reflector[139] = 30;
reflector[131] = 118;
reflector[118] = 131;
reflector[215] = 147;
reflector[147] = 215;
reflector[193] = 24;
reflector[24] = 193;
reflector[154] = 162;
reflector[162] = 154;
reflector[129] = 32;
reflector[32] = 129;
reflector[91] = 140;
reflector[140] = 91;
reflector[36] = 74;
reflector[74] = 36;
reflector[57] = 9;
reflector[9] = 57;
reflector[120] = 87;
reflector[87] = 120;
reflector[212] = 130;
reflector[130] = 212;
reflector[150] = 43;
```

```
reflector[43] = 150;
reflector[63] = 23;
reflector[23] = 63;
reflector[248] = 221;
reflector[221] = 248;
reflector[172] = 185;
reflector[185] = 172;
reflector[169] = 138;
reflector[138] = 169;
reflector[209] = 84;
reflector[84] = 209;
reflector[72] = 50;
reflector[50] = 72;
reflector[247] = 70;
reflector[70] = 247;
reflector[16] = 168;
reflector[168] = 16;
reflector[170] = 235;
reflector[235] = 170;
reflector[200] = 133;
reflector[133] = 200;
reflector[149] = 7;
reflector[7] = 149;
reflector[176] = 123;
reflector[123] = 176;
reflector[59] = 159;
reflector[159] = 59;
reflector[3] = 124;
reflector[124] = 3;
reflector[97] = 189;
reflector[189] = 97;
reflector[238] = 109;
reflector[109] = 238;
reflector[182] = 188;
reflector[188] = 182;
reflector[52] = 163;
reflector[163] = 52;
reflector[117] = 174;
reflector[174] = 117;
reflector[27] = 132;
reflector[132] = 27;
reflector[207] = 254;
reflector[254] = 207;
reflector[241] = 245;
reflector[245] = 241;
reflector[21] = 100;
```

```
    reflector[100] = 21;
    reflector[99] = 126;
    reflector[126] = 99;
    reflector[220] = 40;
    reflector[40] = 220;
    reflector[41] = 250;
    reflector[250] = 41;
    reflector[115] = 119;
    reflector[119] = 115;

     for(int i=0; i<num_rotors; i++) {
        for(int j=0; j<256; j++) {
            rotors_r[i][j] = rotor_lookup(j, i);
        }
    }
}
```

Appendix C: ASCII Table of Values for Enigma Phoenix Encrypted File

Hex	Dec	Chr	No	%	Sum	(X-μ)^2	Entropy

0	0	.	4908	0.39%	0	1770252.65	-0.03120901
1	1	.	4810	0.38%	4810	1557014.52	-0.03055118
2	2	.	4811	0.38%	9622	1389032.45	-0.03055118
3	3	.	5068	0.40%	15204	1296072.98	-0.03186314
4	4	.	4861	0.39%	19444	1092524.56	-0.03120901
5	5	.	4933	0.39%	24665	965731.017	-0.03120901
6	6	.	4975	0.40%	29850	839710.266	-0.03186314
7	7	.	5118	0.41%	35826	735980.945	-0.03251366
8	8	.	4940	0.39%	39520	596845.461	-0.03120901
9	9	.	4970	0.40%	44730	496181.874	-0.03186314
0A	10	.	4798	0.38%	47980	387927.202	-0.03055118
0B	11	.	4906	0.39%	53966	313337.994	-0.03120901
0C	12	.	5001	0.40%	60012	244472.842	-0.03186314
0D	13	.	4863	0.39%	63219	174587.825	-0.03120901
0E	14	.	4952	0.39%	69328	123392.587	-0.03120901
0F	15	.	4806	0.38%	72090	76579.7439	-0.03055118
10	16	.	4955	0.39%	79280	44350.5382	-0.03120901
11	17	.	4990	0.40%	84830	19795.9871	-0.03186314
12	18	.	5017	0.40%	90306	4934.71989	-0.03186314
13	19	.	4889	0.39%	92891	0.33147051	-0.03120901
14	20	.	4911	0.39%	98220	4992.2076	-0.03120901
15	21	.	4921	0.39%	103341	19846.4123	-0.03120901
16	22	.	4887	0.39%	107514	44224.7696	-0.03120901
17	23	.	4963	0.40%	114149	79735.2604	-0.03186314
18	24	.	4871	0.39%	116904	122176.41	-0.03120901
19	25	.	4767	0.38%	119175	172083.343	-0.03055118
1A	26	.	4968	0.40%	129168	244005.03	-0.03186314
1B	27	.	4907	0.39%	132489	314694.803	-0.03120901

1C	28	.	4972	0.40%	139216	403469.25	-0.03186314
1D	29	.	4955	0.39%	143695	496316.328	-0.03120901
1E	30	.	4892	0.39%	146760	592818.511	-0.03120901
1F	31	.	5033	0.40%	156023	725746.946	-0.03186314
20	32		4313	0.34%	138016	729820.64	-0.02788085
21	33	!	4933	0.39%	162789	968005.652	-0.03120901
22	34	4894 0.39%			0	0	0
		23 35 # 4962 0.40%					
		24 36 $ 4854 0.39%					
		25 37 % 4917 0.39%					
		26 38 & 4853 0.39%					
		27 39 ' 4788 0.38%					
		28 40 (4915 0.39%					
		29 41) 4951 0.39%					
		2A 42 * 4819 0.38%					
		2B 43 + 4853 0.39%					
		2C 44 , 4832 0.38%					
		2D 45 -					

4830
0.38%

2E 46 .
4909
0.39%

2F 47 /
4972
0.40%

30 48 0
4767
0.38%

31 49 1
4892
0.39%

32 50 2
4880
0.39%

33 51 3
4945
0.39%

34 52 4
4848
0.39%

35 53 5
4895
0.39%

36 54 6
4919
0.39%

37 55 7
4933
0.39%

38 56 8
4972
0.40%

39 57 9
4827
0.38%

3A 58 :
4894
0.39%

3B 59 ;

4971
0.40%

3C 60 <
4979
0.40%

3D 61 =
5007
0.40%

3E 62 >
4918
0.39%

3F 63 ?
4880
0.39%

40 64 @
4960
0.39%

41 65 A
4911
0.39%

42 66 B
4952
0.39%

43 67 C
4969
0.40%

44 68 D
4937
0.39%

45 69 E
4836
0.38%

46 70 F
4829
0.38%

47 71 G
4777
0.38%

48 72 H
5013
0.40%

49 73 I

4917
0.39%

4A 74 J
5058
0.40%

4B 75 K
4916
0.39%

4C 76 L
4862
0.39%

4D 77 M
4873
0.39%

4E 78 N
4942
0.39%

4F 79 O
4875
0.39%

50 80 P
4977
0.40%

51 81 Q
4960
0.39%

52 82 R
4891
0.39%

53 83 S
4799
0.38%

54 84 T
4976
0.40%

55 85 U
4850
0.39%

56 86 V
4937
0.39%

57 87 W

4907
0.39%

58 88 X
4963
0.40%

59 89 Y
4897
0.39%

5A 90 Z
4929
0.39%

5B 91 [
4907
0.39%

5C 92 \
4966
0.40%

5D 93]
4961
0.39%

5E 94 ^
4986
0.40%

5F 95 _
4967
0.40%

60 96 `
4781
0.38%

61 97 a
4703
0.37%

62 98 b
4832
0.38%

63 99 c
5032
0.40%

64 100 d
4916
0.39%

65 101 e

4466
0.36%

66 102 f
4804
0.38%

67 103 g
4915
0.39%

68 104 h
4681
0.37%

69 105 i
4624
0.37%

6A 106 j
4959
0.39%

6B 107 k
4891
0.39%

6C 108 l
4865
0.39%

6D 109 m
4795
0.38%

6E 110 n
4642
0.37%

6F 111 o
4587
0.37%

70 112 p
4828
0.38%

71 113 q
4938
0.39%

72 114 r
4764
0.38%

73 115 s

4615
0.37%

74 116 t
4593
0.37%

75 117 u
4889
0.39%

76 118 v
4922
0.39%

77 119 w
4832
0.38%

78 120 x
4915
0.39%

79 121 y
4760
0.38%

7A 122 z
4990
0.40%

7B 123 {
4959
0.39%

7C 124 |
4991
0.40%

7D 125 }
4937
0.39%

7E 126 ~
4924
0.39%

7F 127
4845
0.39%

80 128 €
4905
0.39%

81 129

4869
0.39%

82 130 ‚
4765
0.38%

83 131 *ƒ*
5005
0.40%

84 132 „
4813
0.38%

85 133 …
4911
0.39%

86 134 †
4969
0.40%

87 135 ‡
5053
0.40%

88 136 ˆ
4758
0.38%

89 137 ‰
4913
0.39%

8A 138 Š
4948
0.39%

8B 139 ‹
4930
0.39%

8C 140 Œ
4877
0.39%

8D 141
4911
0.39%

8E 142 Ž
5006
0.40%

8F 143

4859
0.39%

90 144
5020
0.40%

91 145 ‘
4950
0.39%

92 146 ’
4973
0.40%

93 147 “
5012
0.40%

94 148 ”
4869
0.39%

95 149 •
4958
0.39%

96 150 –
5011
0.40%

97 151 —
4964
0.40%

98 152 ˜
4880
0.39%

99 153 ™
4969
0.40%

9A 154 š
4970
0.40%

9B 155 ›
4740
0.38%

9C 156 œ
5107
0.41%

9D 157

4859
0.39%

9E 158 ž
4981
0.40%

9F 159 Ÿ
4891
0.39%

A0 160
4835
0.38%

A1 161 ¡
4844
0.39%

A2 162 ¢
5005
0.40%

A3 163 £
4974
0.40%

A4 164 ¤
4906
0.39%

A5 165 ¥
5040
0.40%

A6 166 ¦
4882
0.39%

A7 167 §
4889
0.39%

A8 168 ¨
4895
0.39%

A9 169 ©
4933
0.39%

AA 170 ª
4915
0.39%

AB 171 «

4983
0.40%

AC 172 ¬
4815
0.38%

AD 173
4968
0.40%

AE 174 ®
4945
0.39%

AF 175 ¯
4910
0.39%

B0 176 °
4906
0.39%

B1 177 ±
4951
0.39%

B2 178 ²
4976
0.40%

B3 179 ³
4851
0.39%

B4 180 ´
4966
0.40%

B5 181 µ
4887
0.39%

B6 182 ¶
4924
0.39%

B7 183 ·
4953
0.39%

B8 184 ¸
4908
0.39%

B9 185 ¹

4918
0.39%

BA 186 º
4824
0.38%

BB 187 »
4795
0.38%

BC 188 ¼
4876
0.39%

BD 189 ½
4830
0.38%

BE 190 ¾
4916
0.39%

BF 191 ¿
4956
0.39%

C0 192 À
4925
0.39%

C1 193 Á
5033
0.40%

C2 194 Â
4976
0.40%

C3 195 Ã
4923
0.39%

C4 196 Ä
4865
0.39%

C5 197 Å
4896
0.39%

C6 198 Æ
4918
0.39%

C7 199 Ç

4896
0.39%

C8 200 È
5059
0.40%

C9 201 É
4923
0.39%

CA 202 Ê
4803
0.38%

CB 203 Ë
5084
0.40%

CC 204 Ì
5069
0.40%

CD 205 Í
4877
0.39%

CE 206 Î
4987
0.40%

CF 207 Ï
4932
0.39%

D0 208 Đ
4890
0.39%

D1 209 Ñ
4902
0.39%

D2 210 Ò
4933
0.39%

D3 211 Ó
4960
0.39%

D4 212 Ô
4878
0.39%

D5 213 Õ

4882
0.39%

D6 214 Ö
4965
0.40%

D7 215 ×
4886
0.39%

D8 216 Ø
5004
0.40%

D9 217 Ù
4816
0.38%

DA 218 Ú
4942
0.39%

DB 219 Û
4891
0.39%

DC 220 Ü
5055
0.40%

DD 221 Ý
4866
0.39%

DE 222 Þ
4886
0.39%

DF 223 ß
5019
0.40%

E0 224 à
4921
0.39%

E1 225 á
5044
0.40%

E2 226 â
4797
0.38%

E3 227 ã

4959
0.39%

E4 228 ä
4996
0.40%

E5 229 å
4929
0.39%

E6 230 æ
4936
0.39%

E7 231 ç
4922
0.39%

E8 232 è
4968
0.40%

E9 233 é
4918
0.39%

EA 234 ê
4963
0.40%

EB 235 ë
4955
0.39%

EC 236 ì
4920
0.39%

ED 237 í
5020
0.40%

EE 238 î
4883
0.39%

EF 239 ï
4872
0.39%

F0 240 ð
4878
0.39%

F1 241 ñ

5031
0.40%

F2 242 ò
4818
0.38%

F3 243 ó
5041
0.40%

F4 244 ô
4823
0.38%

F5 245 õ
4925
0.39%

F6 246 ö
4903
0.39%

F7 247 ÷
4911
0.39%

F8 248 ø
4869
0.39%

F9 249 ù
5000
0.40%

FA 250 ú
5087
0.40%

FB 251 û
4952
0.39%

FC 252 ü
4926
0.39%

FD 253 ý
4824
0.38%

FE 254 þ
4960
0.39%

FF 255 ÿ

5018
0.40%

24	36	$	16	0.40%	576	4628.4804	-0.03186314
25	37	%	17	0.43%	629	5513.04038	-0.03380422
26	38	&	14	0.35%	532	5058.38145	-0.0285545
27	39	'	15	0.38%	585	6004.94143	-0.03055118
28	40	(	13	0.33%	520	5737.49666	-0.02720295
29	41	)	13	0.33%	533	6296.71075	-0.02720295
2A	42	*	19	0.48%	798	10058.1978	-0.0369732
2B	43	+	20	0.50%	860	11527.906	-0.03821928
2C	44	,	14	0.35%	616	8755.76477	-0.0285545
2D	45	-	14	0.35%	630	9469.99532	-0.0285545
2E	46	.	18	0.45%	828	13130.0047	-0.03508137
2F	47	/	17	0.43%	799	13335.8399	-0.03380422

30	48	0	10	0.25%	480	8414.77642	-0.02160964
31	49	1	13	0.33%	637	11706.4234	-0.02720295
32	50	2	22	0.55%	1100	21153.2327	-0.04128494
33	51	3	21	0.53%	1071	21515.068	-0.0400669
34	52	4	11	0.28%	572	11984.9787	-0.023745
35	53	5	14	0.35%	742	16191.8397	-0.0285545
36	54	6	19	0.48%	1026	23285.9525	-0.0369732
37	55	7	24	0.60%	1320	31118.23	-0.04428493
38	56	8	22	0.55%	1232	30131.4065	-0.04128494
39	57	9	20	0.50%	1140	28892.5171	-0.03821928
3A	58	:	14	0.35%	812	21302.9925	-0.0285545
3B	59	;	16	0.40%	944	25610.5406	-0.03186314
3C	60	<	13	0.33%	780	21861.7784	-0.02720295
3D	61	=	15	0.38%	915	26470.3759	-0.03055118
3E	62	>	13	0.33%	806	24046.2065	-0.02720295
3F	63	?	17	0.43%	1071	32924.3193	-0.03380422
40	64	@	24	0.60%	1536	48617.7871	-0.04428493
41	65	A	14	0.35%	910	29634.6064	-0.0285545
42	66	B	22	0.55%	1452	48615.0295	-0.04128494
43	67	C	18	0.45%	1206	41486.2296	-0.03508137
44	68	D	18	0.45%	1224	43232.526	-0.03508137
45	69	E	6	0.15%	414	15004.9408	-0.01407123
46	70	F	14	0.35%	980	36425.7591	-0.0285545
47	71	G	15	0.38%	1065	40572.8461	-0.03055118
48	72	H	15	0.38%	1080	42148.0931	-0.03055118
49	73	I	22	0.55%	1606	64171.5655	-0.04128494
4A	74	J	14	0.35%	1036	42362.6814	-0.0285545
4B	75	K	18	0.45%	1350	56464.601	-0.03508137
4C	76	L	13	0.33%	988	42249.2037	-0.02720295
4D	77	M	13	0.33%	1001	43744.4178	-0.02720295
4E	78	N	11	0.28%	858	38301.6885	-0.023745
4F	79	O	14	0.35%	1106	50413.8341	-0.0285545
50	80	P	16	0.40%	1280	59552.0739	-

51	81	Q	13	0.33%	1053	49985.2741	-0.03186314
52	82	R	17	0.43%	1394	67490.6384	-0.02720295
53	83	S	16	0.40%	1328	65552.8644	-0.03380422
54	84	T	12	0.30%	1008	50712.8459	-0.03186314
55	85	U	10	0.25%	850	43570.8696	-0.02514247
56	86	V	12	0.30%	1032	53881.2411	-0.02160964
57	87	W	15	0.38%	1305	69376.7984	-0.02514247
58	88	X	16	0.40%	1408	76194.1818	-0.03055118
59	89	Y	18	0.45%	1602	88220.751	-0.03186314
5A	90	Z	18	0.45%	1620	90759.0474	-0.03508137
5B	91	[	14	0.35%	1274	72592.6008	-0.03508137
5C	92	\	11	0.28%	1012	58632.2246	-0.023745
5D	93	]	21	0.53%	1953	115021.593	-0.0400669
5E	94	^	15	0.38%	1410	84393.5276	-0.03055118
5F	95	_	13	0.33%	1235	75104.2713	-0.02720295
60	96	`	16	0.40%	1536	94884.2897	-0.03186314
61	97	a	27	0.68%	2619	164302.684	-0.0489617
62	98	b	18	0.45%	1764	112361.419	-0.03508137
63	99	c	15	0.38%	1485	96019.7627	-0.03055118
64	100	d	17	0.43%	1700	111559.678	-0.03380422
65	101	e	24	0.60%	2424	161408.411	-0.04428493
66	102	f	22	0.55%	2244	151588.072	-0.04128494
67	103	g	14	0.35%	1442	98803.3674	-0.0285545
68	104	h	15	0.38%	1560	108395.998	-0.03055118
69	105	i	18	0.45%	1890	133153.494	-0.03508137
6A	106	j	16	0.40%	1696	121126.925	-0.03186314
6B	107	k	19	0.48%	2033	147163.536	-0.0369732
6C	108	l	13	0.33%	1404	102992.054	-0.02720295
6D	109	m	12	0.30%	1308	97217.7863	-0.02514247
6E	110	n	29	0.73%	3190	240192.461	-0.05181458

6F	111	o	15	0.38%	1665	126982.727	-0.03055118
70	112	p	10	0.25%	1120	86505.316	-0.02160964
71	113	q	18	0.45%	2034	159075.865	-0.03508137
72	114	r	13	0.33%	1482	117345.339	-0.02720295
73	115	s	14	0.35%	1610	129046.134	-0.0285545
74	116	t	19	0.48%	2204	178801.352	-0.0369732
75	117	u	7	0.18%	819	67239.2976	-0.01641202
76	118	v	18	0.45%	2124	176447.347	-0.03508137
77	119	w	14	0.35%	1666	140023.056	-0.0285545
78	120	x	17	0.43%	2040	173445.277	-0.03380422
79	121	y	21	0.53%	2541	218519.276	-0.0400669
7A	122	z	13	0.33%	1586	137939.052	-0.02720295
7B	123	{	12	0.30%	1476	129812.553	-0.02514247
7C	124	|	18	0.45%	2232	198481.126	-0.03508137
7D	125	}	16	0.40%	2000	179803.931	-0.03186314
7E	126	~	18	0.45%	2268	206113.719	-0.03508137
7F	127		17	0.43%	2159	198318.237	-0.03380422
80	128	€	10	0.25%	1280	118827.951	-0.02160964
81	129		18	0.45%	2322	217832.608	-0.03508137
82	130	‚	9	0.23%	1170	110905.452	-0.02015755
83	131	*ƒ*	13	0.33%	1703	163095.978	-0.02720295
84	132	„	18	0.45%	2376	229875.497	-0.03508137
85	133	…	16	0.40%	2128	207966.039	-0.03186314
86	134	†	18	0.45%	2412	238084.09	-0.03508137
87	135	‡	13	0.33%	1755	174952.835	-0.02720295
88	136	ˆ	19	0.48%	2584	260127.61	-0.0369732
89	137	‰	18	0.45%	2466	250666.979	-0.03508137
8A	138	Š	20	0.50%	2760	283259.195	-0.03821928
8B	139	‹	17	0.43%	2363	244833.596	-0.03380422
8C	140	Œ	12	0.30%	1680	175715.912	-0.02514247

8D	141		20	0.50%	2820	297720.183	-0.03821928
8E	142	Ž	21	0.53%	2982	317751.538	-0.0400669
8F	143		12	0.30%	1716	184536.505	-0.02514247
90	144		17	0.43%	2448	265659.996	-0.03380422
91	145	‘	13	0.33%	1885	206414.976	-0.02720295
92	146	’	15	0.38%	2190	241966.373	-0.03055118
93	147	“	15	0.38%	2205	245791.62	-0.03055118
94	148	”	9	0.23%	1332	149788.12	-0.02015755
95	149	•	17	0.43%	2533	287336.396	-0.03380422
96	150	–	15	0.38%	2250	257447.361	-0.03055118
97	151	—	21	0.53%	3171	365949.651	-0.0400669
98	152	˜	15	0.38%	2280	265367.855	-0.03055118
99	153	™	20	0.50%	3060	359164.136	-0.03821928
9A	154	š	18	0.45%	2772	328090.019	-0.03508137
9B	155	›	10	0.25%	1550	184982.397	-0.02160964
9C	156	œ	12	0.30%	1872	225255.074	-0.02514247
9D	157		15	0.38%	2355	285694.09	-0.03055118
9E	158	ž	11	0.28%	1738	212556.18	-0.023745
9F	159	Ÿ	13	0.33%	2067	254829.973	-0.02720295
A0	160		11	0.28%	1760	218716.543	-0.023745
A1	161	¡	15	0.38%	2415	302495.078	-0.03055118
A2	162	¢	11	0.28%	1782	224964.905	-0.023745
A3	163	£	24	0.60%	3912	497720.915	-0.04428493
A4	164	¤	6	0.15%	984	126164.328	-0.01407123
A5	165	¥	21	0.53%	3465	447686.492	-0.0400669
A6	166	¦	19	0.48%	3154	410616.997	-0.0369732
A7	167	§	16	0.40%	2672	350502.997	-0.03186314
A8	168	¨	15	0.38%	2520	333051.807	-0.03055118
A9	169	©	13	0.33%	2197	292532.114	-0.02720295
AA	170	ª	12	0.30%	2040	273641.841	-0.02514247
AB	171	«	12	0.30%	2052	277278.039	-0.02514247

AC	172	¬	18	0.45%	3096	421407.354	-0.03508137
AD	173		18	0.45%	3114	426933.651	-0.03508137
AE	174	®	17	0.43%	2958	408468.394	-0.03380422
AF	175	¯	9	0.23%	1575	219047.122	-0.02015755
B0	176	°	20	0.50%	3520	493031.711	-0.03821928
B1	177	±	10	0.25%	1770	249666.02	-0.02160964
B2	178	²	15	0.38%	2670	379254.277	-0.03055118
B3	179	³	19	0.48%	3401	486450.064	-0.0369732
B4	180	´	16	0.40%	2880	414778.423	-0.03186314
B5	181	µ	14	0.35%	2534	367453.351	-0.0285545
B6	182	¶	17	0.43%	3094	451718.634	-0.03380422
B7	183	·	16	0.40%	2928	430379.213	-0.03186314
B8	184	¸	13	0.33%	2392	353960.325	-0.02720295
B9	185	¹	18	0.45%	3330	496057.208	-0.03508137
BA	186	º	16	0.40%	2976	446268.004	-0.03186314
BB	187	»	19	0.48%	3553	536308.567	-0.0369732
BC	188	¼	19	0.48%	3572	542711.88	-0.0369732
BD	189	½	17	0.43%	3213	491347.594	-0.03380422
BE	190	¾	7	0.18%	1330	204706.713	-0.01641202
BF	191	¿	20	0.50%	3820	591736.651	-0.03821928
C0	192	À	16	0.40%	3072	478909.585	-0.03186314
C1	193	Á	17	0.43%	3281	514740.714	-0.03380422
C2	194	Â	12	0.30%	2328	367534.584	-0.02514247
C3	195	Ã	14	0.35%	2730	433704.578	-0.0285545
C4	196	Ä	18	0.45%	3528	563974.468	-0.03508137
C5	197	Å	16	0.40%	3152	506990.902	-0.03186314
C6	198	Æ	12	0.30%	2376	384527.374	-0.02514247
C7	199	Ç	14	0.35%	2786	453641.5	-0.0285545
C8	200	È	12	0.30%	2400	393167.769	-0.02514247
C9	201	É	12	0.30%	2412	397523.967	-0.02514247
CA	202	Ê	15	0.38%	3030	502380.206	-

CB	203	Ë	13	0.33%	2639	440167.392	-0.03055118
CC	204	Ì	18	0.45%	3672	616104.84	-0.02720295
CD	205	Í	15	0.38%	3075	518985.947	-0.03508137
CE	206	Î	19	0.48%	3914	664469.512	-0.03055118
CF	207	Ï	15	0.38%	3105	530206.441	-0.0369732
D0	208	Ð	21	0.53%	4368	750206.363	-0.03055118
D1	209	Ñ	17	0.43%	3553	613753.193	-0.0400669
D2	210	Ò	20	0.50%	4200	729682.909	-0.03380422
D3	211	Ó	12	0.30%	2532	442405.943	-0.03821928
D4	212	Ô	13	0.33%	2756	484278.319	-0.02514247
D5	213	Õ	27	0.68%	5751	1016258.26	-0.02720295
D6	214	Ö	14	0.35%	2996	532394.959	-0.0489617
D7	215	×	16	0.40%	3440	614707.645	-0.0285545
D8	216	Ø	9	0.23%	1944	349310.198	-0.03186314
D9	217	Ù	15	0.38%	3255	588108.911	-0.02015755
DA	218	Ú	11	0.28%	2398	435647.049	-0.03055118
DB	219	Û	13	0.33%	2847	520042.818	-0.023745
DC	220	Ü	16	0.40%	3520	646468.962	-0.02720295
DD	221	Ý	10	0.25%	2210	408073.266	-0.03186314
DE	222	Þ	14	0.35%	3108	576972.803	-0.02160964
DF	223	ß	13	0.33%	2899	541051.674	-0.0285545
E0	224	à	7	0.18%	1568	294198.632	-0.02720295
E1	225	á	20	0.50%	4500	848787.85	-0.01641202
E2	226	â	14	0.35%	3164	599933.725	-0.03821928
E3	227	ã	14	0.35%	3178	605743.956	-0.0285545
E4	228	ä	10	0.25%	2280	436844.419	-0.0285545
E5	229	å	10	0.25%	2290	441034.584	-0.02160964
E6	230	æ	15	0.38%	3450	667867.122	-0.02160964
E7	231	ç	9	0.23%	2079	404527.422	-0.03055118
E8	232	è	11	0.28%	2552	499097.585	-0.02015755
E9	233	é	12	0.30%	2796	549594.291	-0.023745
EA	234	ê	18	0.45%	4212	832113.733	-0.02514247

EB	235	ë	12	0.30%	2820	559914.686	0.03508137 -
EC	236	ì	20	0.50%	4720	941851.473	0.02514247 -
ED	237	í	8	0.20%	1896	380220.721	0.03821928 -
EE	238	î	20	0.50%	4760	959292.131	0.01793157 -
EF	239	ï	20	0.50%	4780	968072.461	0.03821928 -
F0	240	ð	15	0.38%	3600	732669.593	0.03821928 -
F1	241	ñ	7	0.18%	1687	345013.592	0.03055118 -
F2	242	ò	18	0.45%	4356	895188.104	0.01641202 -
F3	243	ó	10	0.25%	2430	501796.889	0.03508137 -
F4	244	ô	14	0.35%	3416	708801.875	0.02160964 -0.0285545
F5	245	õ	13	0.33%	3185	664036.384	-
F6	246	ö	18	0.45%	4428	927589.29	0.02720295 -
F7	247	÷	18	0.45%	4446	935779.586	0.03508137 -
F8	248	ø	16	0.40%	3968	839116.34	0.03508137 -
F9	249	ù	17	0.43%	4233	899364.391	0.03186314 -
FA	250	ú	22	0.55%	5500	1174025.69	0.03380422 -
FB	251	û	9	0.23%	2259	484450.386	0.04128494 -
FC	252	ü	17	0.43%	4284	922978.231	0.02015755 -
FD	253	ý	20	0.50%	5060	1095197.07	0.03380422 -
FE	254	þ	12	0.30%	3048	662746.441	0.03821928 -
FF	255	ÿ	20	0.50%	5100	1113997.73	0.02514247 - 0.03821928