Manipulating the Matrix

New Mexico Adventures in
Supercomputing Challenge
Final Report
April 2, 2003

Team Number 66
Silver High School

Team Members:
Rebecca F. Ashton, Senior
Ruben M. Guadiana, Senior
Teacher: Mrs. Peggy Larisch

Acknowledgements

The authors wish to acknowledge the following individuals for the guidance and assistance provided in the overall preparation of this report:

- Mrs. Peggy Larisch – Teacher, Silver High School, Advanced Computer Studies
- Mr. Berry Estes – Nuclear Engineer (Retired), Sandia National Laboratories

Table of Contents

E.0 Executive Summary

The purpose of this project is to create a Java computer application that will encrypt and decrypt data in a secure manner using fundamental matrix multiplication. Consequently, the team developed an original computer program that utilizes matrix mathematical techniques to create a unique encryption/decryption process. Though many people are not aware of it, encryption is used daily, whether it be an individual using a credit card to order products over the Internet, a code employed to transmit secret information to selected military/diplomatic groups, or someone simply signing into an email account with a password. As the use of encryption increases, the need for a more secure and easier to use method of cryptography increases simultaneously. By encrypting data using matrix multiplication and random numbers, an ambiguous key can easily be created.

The method of encryption employed by the program created for this project first builds a character array (arrays act as matrices) containing the position(s) and ASCII value (see Appendix #1) of a specific character in the user-input message. Using a randomly generated seed, a second array, with square dimensions congruent to the length of the character array, is formulated. These two arrays are multiplied together, producing the product array, which has the same dimensions as the original character array. The seed used to create the random array is multiplied by 26 times the length of the message and affixed to the end of the product array. This process is repeated for every character found in the message to be encrypted and the product arrays are saved to a data file. This data is the secure key that can then be passed safely from user to user.

In order to decrypt this data file, the computer program separates the encrypted arrays, uncovers the disguised seed, and recreates the random arrays used during encryption. The inverses of these arrays are then found and multiplied by the product arrays, returning the original character arrays containing character positions and ASCII values. Using this data, the program pieces the original message back together and returns it, unflawed, to the user.

The cryptographic Java computer program created for this project correctly employs matrix multiplication to both encrypt and decrypt data. This mathematical model proved to be effective and the computer program successfully encrypted data into a secure and ambiguous key (see example in Appendix #4) and also correctly decrypted this data. The simplicity of the program allows the user to easily encrypt and decrypt data, only prompting the user for either the message to be encrypted or the location of the data file to be decrypted.

The project conquered its intended tasks, as the cryptographic computer program is both secure and easy to use. Though the techniques used in this project differ greatly from techniques used by other cryptographic programming techniques, the originality and simplicity of the idea resulted in an encryption/decryption technique that is highly successful. To further improve the quality of the project, it is recommended that a superior method be created which enables the program to separate arrays during the decryption process; this, in the opinion of the team, will improve the security of the key.

1.0 Introduction

1.1 Purpose

The Science of Cryptography, or the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form[1], has existed for almost 4000 years. Ancient Egyptians used cryptography to encipher messages on monuments using hieroglyphics; simultaneously, ancient Hebrews enciphered certain words in their scriptures. About 2000 years later, Julius Caesar created the Caesar cipher, a simple monoalphabetic cipher that replaced each letter of a message by another letter in the alphabet a fixed distance away. Since then, numerous other unique methods of encryption have been created, including the Jefferson Cylinder, which was developed in the 1790s by Thomas Jefferson and this device encrypted data using disks with random alphabets. The Wheatstone disc was created in 1817 by Wadsworth and used two concentric wheels to generate a polyalphabetic cipher. The use of encryption, as well as the need for more dependable encryption techniques, increased dramatically during World War II as countries attempted to pass vital messages to allies without enemies intercepting and decrypting the code. One of the most popular cipher machines used at this time was the Enigma Rotor Machine, which was developed by Albert Scherbius, and consisted of a series of rotor wheels with internal cross-connections that provided substitution using a continuously changing alphabet[2].

Advancement in encryption and decryption methods continues today. With the use of the Internet increasing daily, the need for a more secure method of cryptography rises simultaneously. Because online shopping is both convenient and easy, one of the main targets of data encryption on the Internet is credit card numbers[3]. Improved techniques to encrypt passwords are needed to prevent hackers from breaking into personal emails or illegally obtaining confidential information.

In order to investigate this ever-rising demand for superior encryption techniques, the purpose of this project is to create an improved cryptographic system that is both secure and easy to use. The encryption method developed in this project incorporates the fundamentals of matrix multiplication to create a key of random numbers unreadable to the naked eye and almost impossible to decrypt without knowledge of the techniques used to encrypt data. Proper decryption of the key also depends on identifying the inverse of a matrix; multiplying this inverse by the key matrix returns the original matrix, which contains the desired information.

1.2 Scope

The objective of this project is to create a method of encryption and decryption that is both more secure and easier to use than those currently being utilized today. The methods developed in this project to create and break down an unintelligible key will incorporate the American Standard Code for Information Interchange (ASCII) (See Appendix #1), random number generators, and matrix multiplication.

1.3 Computer Program

A Java computer program created by the team members is used to encrypt and decrypt messages for this project. The multiple dimension arrays available in this programming language act as the matrices, which are multiplied together to encrypt and decrypt messages. Because these arrays are capable of holding an infinite number of characters, there is no limit to the length of the message that can be encrypted and decrypted by this program. The first array used in multiplication consists of the positions and the numerical ASCII value (see Appendix #1) of each specific character. The second array is comprised of numbers created by incrementing the seed, a number randomly generated by the computer, by one for each cell in the array. The computer program performs the required matrix based operations – multiplication, adjoint, determinant, seed definition, inverse multiplication – to encrypt/decrypt the user-input data. The program is an original code developed by the team based on the information and materials given in the referenced documents (see References).

2.0 Project Proposal

2.1 Description of Project

This project develops a computer program that has the ability to encrypt and decrypt data using fundamental matrix multiplication (see Section 3.1). ASCII values (see Appendix #1) represent characters in encryption, and are used to identify letters in decryption. These values are the only constants in the program, as the other matrices used in the encryption process are filled with numbers created by incrementing a randomly generated seed by one. These matrices multiply character matrices, which contain position(s) and the ASCII value of a specific character. This multiplication process produces the key, a data file containing only numbers. Because these numbers have been augmented through the matrix multiplication process, they will be meaningless to persons without the decryption techniques.

In order to decrypt the key, the computer program imports and reads the data file created in the encryption process. The data from this file fills a very large matrix where each row represents one product matrix created during encryption. Further separation of this matrix reforms original, one-row character matrices. From these matrices, seeds are uncovered, and random matrices are recreated. An inverse program created by The MathWorks and the National Institute of Standards and Technology and available at http://math.nist.gov/javanumerics/jama then finds the inverses of the recreated random matrices. Character matrices are multiplied by the inverse of their specific random matrices, returning the original character matrix. The program reads these matrices, piecing together the original message using the positions and ASCII values uncovered during the decryption process.

3.0 Analytical Methodology

3.1 Mathematical Bases

Matrix multiplication is used in both the encryption and decryption processes and is thus the basis of the mathematical and analytical models used in this project. For matrix multiplication, it is important to remember that in order for a product of two matrices to be defined, the dimensions of these matrices must satisfy:

$$(n * m)\ (m * p) = (n * p)$$

where (n * m) denotes a matrix with *n* rows and *m* columns[4]. The matrix product of two matrices is formed by multiplying every row of the first matrix with every column of the second matrix[5]. A matrix having one row is used as the first matrix because it represents the size of the character matrices used in the encryption process. Following is an example of the matrix multiplication process:

$$\begin{bmatrix} 5 & 11 & 17 & 10 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 7 & 9 \\ 9 & 5 & 12 & 5 \\ 5 & 8 & 2 & 15 \\ 1 & 2 & 6 & 5 \end{bmatrix} =$$

5 * 3 + 11 * 9 + 17 * 5 + 10 * 1	5 * 4 + 11 * 5 + 17 * 8 + 10 * 2	5 * 7 + 11 * 12 + 17 * 2 + 10 * 6	5 * 9 + 11 * 5 + 17 * 15 + 10 * 5

Each row of the first matrix is multiplied by the corresponding columns of the second matrix. Adding the products of these individual multiplications returns the final answer.

$$\begin{bmatrix} 209 & 231 & 261 & 405 \end{bmatrix}$$

To find the inverse of a matrix, a matrix must be square, meaning it has the same number of rows and columns, i.e. an (n * n) matrix. The formula[6] to find an inverse matrix is:

$$A^{-1} = \frac{\text{adjoint A}}{\text{determinant A}}$$

When calculating the adjoint matrix, each element in the matrix is replaced by its cofactor. The cofactor of an element is the value obtained by finding the determinant formed by the elements not in that particular row or column. After creating the cofactor matrix, a positive or negative sign is applied to each element, starting with a positive charge and alternating for each element. This matrix is then transposed, meaning the first column becomes the first row, the second

column becomes the second row, etc. This new matrix is the adjoint of the original matrix[6]. The process used to calculate the adjoint matrix of a 3 * 3 matrix follows.

$$\begin{bmatrix} \mathbf{7} & \mathbf{1} & \mathbf{5} \\ \mathbf{0} & \mathbf{2} & \mathbf{6} \\ \mathbf{4} & \mathbf{9} & \mathbf{3} \end{bmatrix}$$

The cofactor of 1 in the above matrix:

$$\begin{vmatrix} 0 & 6 \\ 4 & 3 \end{vmatrix} = 0 - 24 = -24$$

The cofactor of 6 in the above matrix:

$$\begin{vmatrix} 7 & 1 \\ 4 & 9 \end{vmatrix} = 63 - 4 = 59$$

The matrix filled with cofactors, positive and negative signs applied:

$$\begin{bmatrix} \mathbf{+(-48)} & \mathbf{-(-24)} & \mathbf{+(-8)} \\ \mathbf{-(-42)} & \mathbf{+(1)} & \mathbf{-(59)} \\ \mathbf{+(-4)} & \mathbf{-(42)} & \mathbf{+(14)} \end{bmatrix} = \begin{bmatrix} \mathbf{-48} & \mathbf{24} & \mathbf{-8} \\ \mathbf{42} & \mathbf{1} & \mathbf{-59} \\ \mathbf{-4} & \mathbf{-42} & \mathbf{14} \end{bmatrix}$$

The cofactor matrix is transposed, the resulting matrix being the adjoint matrix:

$$\begin{bmatrix} \mathbf{-48} & \mathbf{24} & \mathbf{-8} \\ \mathbf{42} & \mathbf{1} & \mathbf{-59} \\ \mathbf{-4} & \mathbf{-42} & \mathbf{14} \end{bmatrix} = \begin{bmatrix} \mathbf{-48} & \mathbf{42} & \mathbf{-4} \\ \mathbf{24} & \mathbf{1} & \mathbf{-42} \\ \mathbf{-8} & \mathbf{-59} & \mathbf{14} \end{bmatrix}$$

When finding the value of the determinant of a matrix, each integer in the first column is multiplied by its cofactor. These values are added together, the resultant being the determinant of the matrix[6]. In order for a matrix to have an inverse, the determinant cannot equal zero[7]. This is due to the fact that the inverse matrix is defined as the adjoint divided by the determinant, so a matrix with a determinant of zero would return an undefined answer[8]. This is how the determinant of the above 3 * 3 matrix is found:

$$\begin{bmatrix} \mathbf{7} & \mathbf{1} & \mathbf{5} \\ \mathbf{0} & \mathbf{2} & \mathbf{6} \\ \mathbf{4} & \mathbf{9} & \mathbf{3} \end{bmatrix} = \mathbf{7}\begin{vmatrix} \mathbf{2} & \mathbf{6} \\ \mathbf{9} & \mathbf{3} \end{vmatrix} + \mathbf{0}\begin{vmatrix} \mathbf{1} & \mathbf{5} \\ \mathbf{9} & \mathbf{3} \end{vmatrix} + \mathbf{4}\begin{vmatrix} \mathbf{1} & \mathbf{5} \\ \mathbf{2} & \mathbf{6} \end{vmatrix}$$

$$= \mathbf{7(-48) + 0(-42) + 4(-4)} = \mathbf{-352}$$

Now that the adjoint matrix and the determinant have been calculated, these values are substituted into the inverse formula. Solving this equation will return the inverse of the original 3 * 3 matrix:

$$\mathbf{A}^{-1} = \frac{1}{-352} * \begin{bmatrix} -48 & 42 & -4 \\ 24 & 1 & -42 \\ -8 & -59 & 14 \end{bmatrix} = \begin{bmatrix} 3/22 & -21/176 & 1/88 \\ -3/44 & -1/352 & 21/176 \\ 1/44 & 59/352 & -7/176 \end{bmatrix}$$

The two mathematical models presented above are the basis of the encryption and decryption processes created by this project. Matrix multiplication is used to encrypt data, creating a key that is unreadable to the naked eye. The inverse matrix is used in the decryption process. Multiplying a product matrix by the inverse of one of the matrices used during multiplication will return the other matrix used during multiplication. In this case, these calculations would return the original character matrices, which will be used to piece the original message back together.

3.2 Computer Applications

When encrypting a message using the computer program created specifically for this project, the user is first prompted to enter the message to be encrypted. After the user inputs the message, it is read by the program and fills an array (arrays act as matrices) initialized to the length of message, including spaces. With the message contained inside an array, each character has a specific numerical position. These position values start at zero and end at *x-1*, where *x* is the numerical length of the message. The program scans the message array, filling a character array with the position(s) of a specific character. The numerical ASCII value of the character is attached to the end of the array containing its corresponding position(s). These character arrays consist of 1 row with *n* +3 columns, where *n* is the total number of times the character is used in the message; three columns are added to the array to compensate for the addition of the ASCII value, disguised seed, and the number used to separate the arrays in the data file (-2). The last two positions of the array remain at zero until after the multiplication of arrays has been completed. A unique array is created for each character that is present in the message. Characters not found in the message are disregarded.

For each character array that is created during the scanning process, a corresponding array of numbers is created in order to complete the matrix multiplication process. Because this array will be used as the multiplier, it must have square dimensions equivalent to number of columns of the character array. This array is created by first generating a random seed and then incrementing this number by one as it fills each cell in the array created specifically for a character. Different random seeds are generated and separate arrays are created for each character array produced by the program.

The character array and random array are then multiplied together, returning a product array with dimensions equivalent to the original character array. Because the seed is transported along with this product array, it must be disguised to look like the other numbers in the array. In order to accomplish this resemblance, the randomly generated seed used to create the

random array is multiplied by 26 times the length of the product array and attached to the end of the array. This final key array contains the masked seed and values from the matrix multiplication process. This procedure is repeated until every character in the message has been transformed into an unrecognizable key array. These key arrays are then saved to a data file, which acts as the lock and key of the encryption process and can be safely passed from user to user (see Figure 3.1).

Flow Chart for Encryption

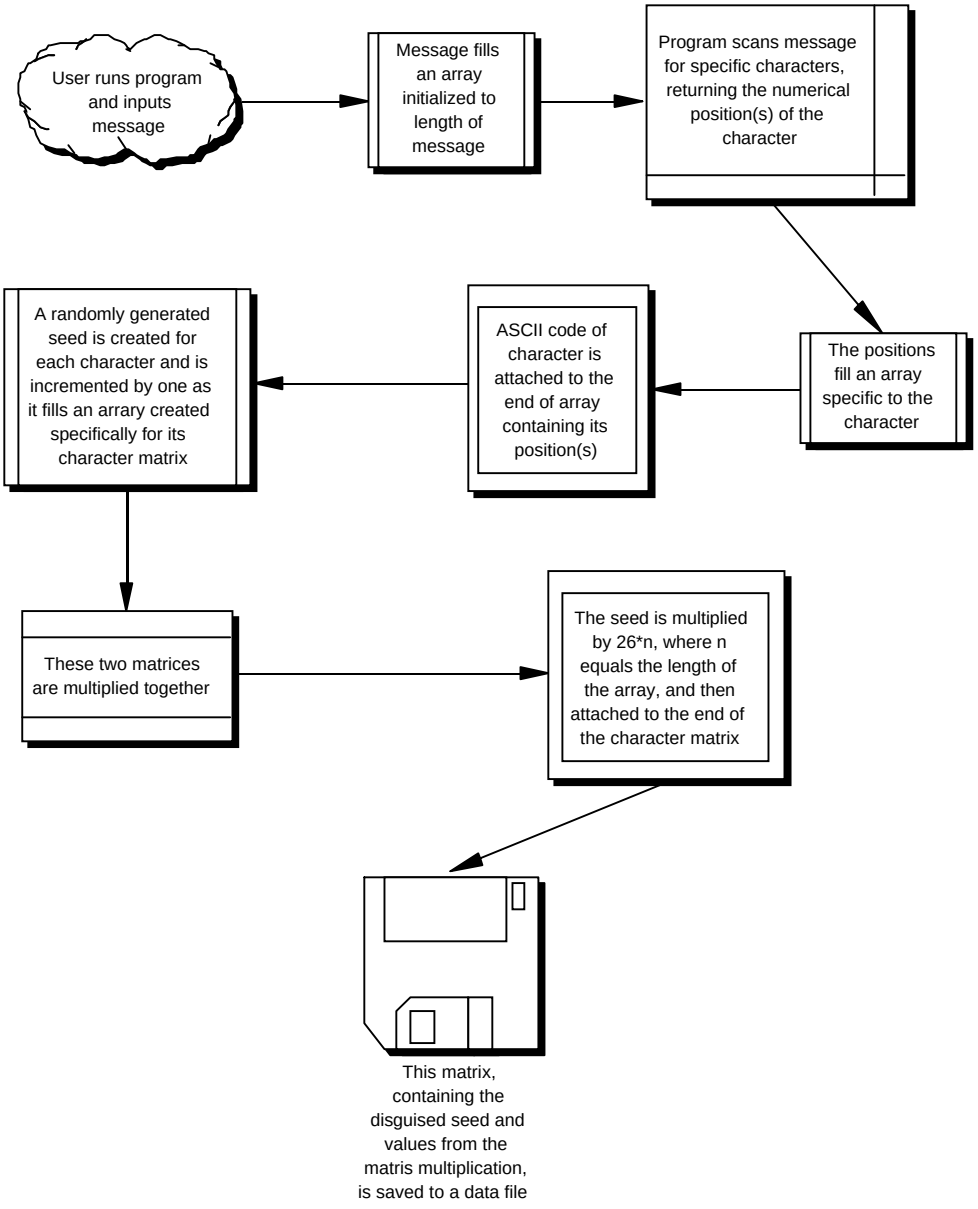

Figure 3.1

In order to decrypt data, a user must have both the key data file and the correct decryption techniques. The decrypt program reads the file and fills a massive array with the data contained in it, skipping a line in the array each time a separator number (-2) is encountered. Because of this separation process, each line of this array represents one of the numerous product arrays created during the encryption process. These rows of this array are then further separated into single row arrays. The disguised seed is removed from its predetermined position and divided by $26(x-1)$, where x is the total number of items in the array; $x^2+(x-2)$ is then subtracted from the resulting number in order to obtain the original seed. Using this seed, the random array is recreated by incrementing this number by one every time it is entered into the newly recreated random array. An inverse program created by The MathWorks and the National Institute of Standards and Technology, available at http://math.nist.gov/javanumerics/jama, then finds the inverse of this array, which is subsequently multiplied by the product array. This multiplication returns the original character array, containing the initial positions and ASCII value of a specific character. After each character array from the key data file has gone through this inverse multiplication process, ASCII values are converted back into their corresponding characters and placed in the correct position. This process pieces the original message back together, which is then returned, unflawed, to the user (see Figure 3.2).

Flow Chart for Decryption

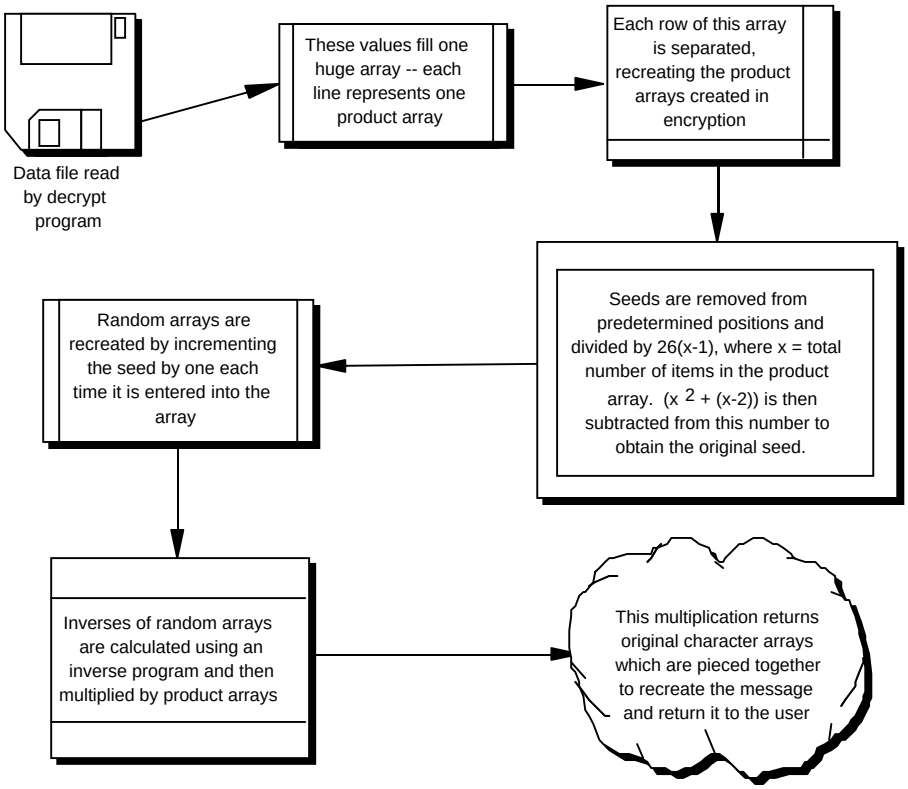

Figure 3.2

4.0 Results

The calculations performed by the computer program allow the user to readily encrypt and decrypt data using matrix multiplication. The values resulting from the encryption process are saved to a data file, which can be safely passed from user to user. Because the values of the data file have been augmented through matrix multiplication, the data can only be decrypted by a person with knowledge of the correct techniques or access the computer program created for this project. Thus, even if the data file was somehow intercepted, the data contained in it is temporarily safe as cracking the code would be very difficult and time consuming as the numbers contained in the data will be incomprehensible without the decryption techniques.

4.1 Computer Calculations

Because the matrix multiplication technique applied to this program consists of both multiplication and division, careful attention to each component of the overall procedure was required in order to ensure correct encryption. To check the accuracy of the values calculated within the program, identical calculations were performed by a graphing calculator. The results of both calculations were compared, and, after the computer calculations were verified to be correct, the encryption process was proven to be complete and correct (see Appendix #4 for Program Output Code). The program's calculation of inverse matrices was also verified in order to ensure correct decryption of data.

Reading the data files proved to be the next major problem encountered while creating the computer program, as the whole data file had to be imported at one time. In order to break the data file into usable arrays, an array of arrays was created. To accomplish this task, a class was written that would read numbers imported from the massive data file and separated them into the original arrays used during encryption (See Appendix #3). First, the numbers were imported into a huge, single-row array. The class read this huge array and separated the arrays created during encryption, placing each one in its own separate object. The computer program used separator numbers (-2), which were placed between arrays during encryption, to distinguish and separate arrays. This array was then further broken down until all original character arrays were reformed into their initial form.

5.0 Conclusions

5.1 Mathematical Models

By multiplying matrices together to encrypt data, a very ambiguous key data file was created. The random number generator used to create the seed of one of the matrices used in multiplication increased the security of the data. The use of matrix multiplication was also a successful mathematical model for decryption because of its inverse properties. By finding the inverse of the random matrix, it was very easy to return encrypted character matrices back to original form. The use of matrix multiplication allowed the program to successfully encrypt and decrypt data and is thus a reliable mathematical model. The results obtained from applying these mathematical models in the computer program were shown to be both correct and acceptable (see Appendix #4).

5.2 Computer Program

The Java computer program created specifically for this project met all defined requirements needed to easily encrypt and decrypt data (See Appendix #2 & #3). The computer program produced a secure key using multiplication of arrays, which represented matrices. Decryption of data also incorporated multiplication of arrays, but the inverse of the random array was first found using a program created by The MathWorks and the National Institute of Standards and Technology and available at http://math.nist.gov/javanumerics/jama. The program successfully completed its intended tasks, as it encrypted data into a form unreadable by the naked eye and also decrypted data, returning it, unflawed, to its original form.

5.3 Results

Matrix multiplication proved to be a successful mathematical technique for modeling data to be encrypted and decrypted. The computer program developed for this project incorporated both constant ASCII values and random number generators, which were used to create an unintelligible and secure key (examples available in Appendix #4). The properties of multiplying product matrices by the inverse of the random matrices used during multiplication also proved to be very beneficial for the decryption process, as this is how original character matrices were found. The individual work and teamwork involved in developing the cryptographic program that employs matrix multiplication to encrypt and decrypt data resulted in the creation of a better method of encryption and decryption that is both secure and easy to use. In order to encrypt data, a user simply types a message into the program, and, after the multiplication has been completed, passes the camouflaged key to another user without worry of premature decryption. If the data file were somehow intercepted, cracking the code would be very difficult and time consuming. The simplicity of the program and complexity of the key make these methods of encryption and decryption very reliable and straightforward.

5.4 Recommendations

To improve the project beyond what had been accomplished to date, it is recommended that classes be created in the program for the different steps of the encryption and decryption processes. By doing so, the program will become more efficient and easier to debug. The program also could be converted into a Graphical User Interface (GUI), which would allow for encryption and decryption over the Internet. To further improve the security of the key, a more advanced method of separating arrays during encryption and decryption needs to be created and implemented in the program.

References

1. Seth, Anuj. “Basics of Cryptography”. http://www.anujseth.com/crypto/basics.html (8 January 2003).

2. Seth, Anuj. “A Brief History of Cryptography”. http://www.anujseth.com/crypto/history.html (8 January 2003).

3. ThinkQuest Team 27158. “Introduction and History of Its Development”. http://www.library.thinkquest.org/27158/history.html (30 October 2002).

4. Weisstein, Eric W. “Matrix Multiplication – from MathWorld”. http://mathworld.wolfram.com/MatrixMultiplication.html (13 November 2002).

5. Lundmark, Hans. “Matrix multiplication: an interactive micro-course for beginners”. http://www.mai.liu.se/~halun/matrix/matrix.html (13 November 2002).

6. Bourne, M. “Method 2”. http://www.np.edu/~bms/Mat_Det/method2.htm (27 January 2003).

7. Mr. Meanie. “Matrix Math”. http://easyweb.easynet.co.uk/~mrmeanies/matrix/matrices.htm (27 January 2003).

8. Gladwin, Thomas. “Matrix algebra”. http://www.ppsw.rug.nl/~gladwin/matalg.html (29 January 2003).

Appendices

Appendix #1 – American Standard Code
for Information Interchange (ASCII)

Appendix #2 – JAVA Program Code, matrix.java

Appendix #3 – JAVA Class Code, MatrixHouse.java

Appendix #4 – Example of Program Output Code

Appendix #1

American Standard Code for Information Interchange (ASCII)

ASCII	Character	ASCII	Character	ASCII	Character	ASCII	Character
0	null	**32**	Space	**64**	@	**96**	`
1	start of heading	**33**	!	**65**	A	**97**	a
2	start of text	**34**	"	**66**	B	**98**	b
3	end of text	**35**	#	**67**	C	**99**	c
4	end of transmission	**36**	$	**68**	D	**100**	d
5	enquiry	**37**	%	**69**	E	**101**	e
6	acknowledge	**38**	&	**70**	F	**102**	f
7	bell	**39**		**71**	G	**103**	g
8	backspace	**40**	(	**72**	H	**104**	h
9	horizontal tab	**41**	)	**73**	I	**105**	i
10	NL line feed, new line	**42**	*	**74**	J	**106**	j
11	vertical tab	**43**	+	**75**	K	**107**	k
12	NP form feed, new page	**44**	,	**76**	L	**108**	l
13	carriage return	**45**	-	**77**	M	**109**	m
14	shift in	**46**	.	**78**	N	**110**	n
15	shift out	**47**	/	**79**	O	**111**	o
16	data link escape	**48**	0	**80**	P	**112**	p
17	device control 1	**49**	1	**81**	Q	**113**	q
18	device control 2	**50**	2	**82**	R	**114**	r
19	device control 3	**51**	3	**83**	S	**115**	s
20	device control 4	**52**	4	**84**	T	**116**	t
21	negative acknowledgement	**53**	5	**85**	U	**117**	u
22	synchronous idle	**54**	6	**86**	V	**118**	v
23	end of trans. Block	**55**	7	**87**	W	**119**	w
24	cancel	**56**	8	**88**	X	**120**	x
25	end of medium	**57**	9	**89**	Y	**121**	y
26	substitute	**58**	:	**90**	Z	**122**	z
27	escape	**59**	;	**91**	[	**123**	{
28	file separator	**60**	<	**92**	\	**124**	|
29	group separator	**61**	=	**93**	]	**125**	}
30	record separator	**62**	>	**94**	^	**126**	~
31	unit separator	**63**	?	**95**	_	**127**	DEL

Appendix #2

JAVA Program Code
matrix.java

```
/* Ruben and Rebes Encryption Program 1.61
     12.8.02 - 3.30.03
     first complete finish.61
*/
import Jama.*;
import TerminalIO.*;
import java.util.*;
import java.io.*;

public class matrix {
   public static void main (String [] args) {
/*****************************************************************************/
       //declare variables
       int count = 0, space, count2 = 0, sizeFinal = 500, count3 = 0, sizeMat = 0;
       int[][] spaceTimes, foreAdd;
       int[] cryptMess = new int[0], finalCode = new int[500], codeIn = new int[0],
randomMatricies = new int[500],
       randIn = new int[0];
       boolean found = false;
       Random generator = new Random();
       KeyboardReader reader = new KeyboardReader();
       String message;   //message before it becomes an array
       char[] array, checker;
/*****************************************************************************/

       //fill finalCode with -1 so it can be trimmed once it is filled
       for (int i = 0; i < finalCode.length; i++)
          finalCode[i] = -1;
/*****************************************************************************/

       //input message to be encrypted
     System.out.println ("Enter message to go into an array: ");
     message = reader.readLine();

     //declare array to be as long as amount of characters in message
     array = new char[message.length()];

     /*fill array with the characters from message*/
     for (int i = 0; i < message.length(); i++){
          array[i] = message.charAt(i);
```

```
    }
/*************************************************************************/
//search array for individual character
      checker = new char[message.length()];
      for (int c = 0; c < array.length; c++){
        for (int i = 0; i < checker.length; i++){
          found = false;
          if (checker[i] == array[c]){
            found = true;
            break;
          } //if loop
        } //for loop
        if (found == false){
          checker[count] = array[c];
          count++;
          ArrayList spaceOccur = new ArrayList();
        int seed = generator.nextInt(99) + 1;
        space = 0;
        for (int i = 0; i < array.length; i++){
           if (array[i] == array[c]){
             spaceOccur.add(space, new Integer(i));   //fills array withoccurances of character
                space++;
             }//if
        } //closes for loop

      int asciiNum = array[c] & 0x00FF;
      spaceOccur.add(spaceOccur.size(), new Integer(asciiNum));//putsascii decimal number at
end of occurance array
//*************************************************************************
      //test out spaceOccur to make sure it fills correctly
      /*uncomment to view the code before encryption
      for (int i = 0; i < spaceOccur.size(); i++)
           System.out.println ((Integer)spaceOccur.get(i) + " ");*/
//*************************************************************************
      //builds x * x matrix out of a random seed
      spaceTimes = new int[spaceOccur.size()][spaceOccur.size()];
      for (int i = 0; i < spaceTimes.length; i++){
         for (int i2 = 0; i2 < spaceTimes.length; i2++){
            spaceTimes[i][i2] = seed;
            seed += 1;
         }
      }//closes for
//*************************************************************************
         // This is the function that multiplies the matricies but doesn't add them
          foreAdd = new int[spaceOccur.size()][spaceOccur.size()];
          for (int i = 0; i < spaceOccur.size(); i++){
```

```
          for (int i2 = 0; i2 < spaceOccur.size(); i2++){
            foreAdd[i][i2] = ((Integer)spaceOccur.get(i)).intValue() * spaceTimes[i][i2];
          }
        }
//*************************************************************************
        //This function takes the multiplied matrix and adds it appropiately and displays code out
        cryptMess = new int[spaceOccur.size() + 2];
        int p;
        for (p = 0; p < spaceOccur.size(); p++){
           for (int i2 = 0; i2 < spaceOccur.size(); i2++){
              cryptMess[p] += foreAdd[i2][p];   // adds the productsof foreAdd to come out with
the code matrix
           }
        }
        cryptMess[p] = (int)seed * (int)cryptMess.length * 26;
        p ++;
        cryptMess[p] = -2;
/*************************************************************************/
        //builds finalCode to house the message after it is finished encrypting
        for (int i = 0; i < cryptMess.length; i++ ){
           finalCode[count2] = cryptMess[i];
           count2++;
        }
/*************************************************************************/
        }//closes if loop
      }//closes for loop
/*************************************************************************/
      //determine the used size of finalCode for use below
      for (int i = 0; i < finalCode.length; i++){
        if (finalCode[i] == -1){
           sizeFinal = i;
           break;
        }
      }
/*************************************************************************/
      System.out.println (" ");
      //print finalCode for testing
      for (int i = 0; i < sizeFinal; i++)
        System.out.print (finalCode[i]);
      System.out.println (" ");
/*************************************************************************/
      //writes finalCode to file
      try{
        FileOutputStream fileOut = new FileOutputStream("crypt.dat");
        DataOutputStream dataOut = new DataOutputStream(fileOut);
        dataOut.writeInt(sizeFinal);
```

```
         for (int i = 0; i < sizeFinal; i++){
            dataOut.writeInt(finalCode[i]);
         }
         dataOut.close();
      }catch(IOException e){
         System.err.println("Error during output: " + e.toString());
    }//closes catch
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
    //START WITH THE DECRYPTION SIDE, AND CRYPT.DAT
      //input encrypted message from file
      try{
       FileInputStream fileIn = new FileInputStream("crypt.dat");
       DataInputStream dataIn = new DataInputStream(fileIn);
         int length = dataIn.readInt(); //fills codeIn array with entirefile
         codeIn = new int[length];
         for (int i = 0; i < length; i++){
          codeIn[i] = dataIn.readInt(); //fills codeIn array with entirefile
         }
    }catch(EOFException e){
      System.out.println("\nAll done.");
    }catch(IOException e){
      System.err.println("Error in input" + e.toString());
    }
//*****************************************************************************
    //declare variables for codeIn
    int p = 0, rowCount = 0, colCount = 0, numOfZero = 0;
//*****************************************************************************
    //count occurances of 0 which determines how many arrays withincodeIn there are
      for (int i = 0; i < codeIn.length; i++){
         if (codeIn[i] == -2)
            numOfZero++;
      }
//*****************************************************************************
    //declare variables for codeIn
      int[][] codeHouse = new int[numOfZero][250];
      int[] newRow = new int[numOfZero];
//*****************************************************************************
      //break codeIn back into the seperate arrays for decryption
      while (p < codeIn.length){
         if (codeIn[p] != -2){
```

```
            codeHouse[rowCount][colCount] = codeIn[p];
            colCount++;
         }
         else if (codeIn[p] == -2){
            colCount = 0;
            rowCount++;
         }
         p++;
      }
//*************************************************************************
      //cut off excess cells from codeHouse
      for (int i = 0; i < numOfZero; i++){
         for (int i2 = 0; i2 < 250; i2++){
            if (codeHouse[i][i2] == 0){
               newRow[i] = i2;
               break;
            }
         }
      }
//*************************************************************************
      //now ready for the class to be implemented
      MatrixHouse[] codeHouses = new MatrixHouse[numOfZero];
      for (int i = 0; i < codeHouses.length; i++)
         codeHouses[i] = new MatrixHouse();
//*************************************************************************
      for (int i = 0; i < codeHouses.length; i++)
         codeHouses[i].setSmall(codeHouse, i, newRow[i]);
//*************************************************************************
//*************************************************************************
//*************************************************************************
     //create matrixhouses for the seeds, to make seeds into matricies
     MatrixHouse[] seeds = new MatrixHouse[codeHouses.length];
     for (int i = 0; i < seeds.length; i++)
        seeds[i] = new MatrixHouse();
//*************************************************************************
     for (int i = 0; i < seeds.length; i++)
        seeds[i].setSeed(codeHouses[i].seed(),codeHouses[i].getL());
//*************************************************************************
      //create a Matrix that will call the inverse of seed[i]
      Matrix[] seedi = new Matrix[codeHouses.length];
      for (int i = 0; i < seedi.length; i++)
         seedi[i] = new Matrix(seeds[i].getRand());
//*************************************************************************
      //find inverse of random[]
      for (int i = 0; i < seedi.length; i++){
         seedi[i] = seedi[i].inverse();
```

```
      }
//*********************************************************************************
      //construct an empty matrix to house the multiplied
      MatrixHouse[] occures = new MatrixHouse[codeHouses.length];
      for (int i = 0; i < occures.length; i++)
         occures[i] = new MatrixHouse();
//*********************************************************************************
      //fill occures with the occurances of the character
      for (int i = 0; i < occures.length; i++)
         occures[i].setOccure(seedi[i], codeHouses[i].getArray(), codeHouses[i].getLength());
//*********************************************************************************
      //determine size of message
     int endSize = 0;
     for (int i = 0; i < occures.length; i++)
        endSize += occures[i].getOLength();
//*********************************************************************************
     //construct a character array to house the message
     char[] mess = new char[endSize];
     for (int i = 0; i < numOfZero; i++)
        occures[i].makePos(endSize);
//*********************************************************************************
     //fill mess with the correct characters and their occurances
     for (int i = 0; i < numOfZero; i++){
        if (occures[i].charAt(i) != '`')
           mess[i] = occures[i].charAt(i);
      }
//*********************************************************************************
     //print mess
     for (int i = 0; i < mess.length; i++)
        System.out.print (mess[i]);
//*********************************************************************************
   }// closes main
}//closes program
```

Appendix #3

JAVA Class Code
MatrixHouse.java

```
/*This is the class needed to house the random matricies
  being pulled in from the file. Actually works and does
  everything needed
*/
import Jama.*;

public class MatrixHouse {
   //declare variables
   private int[] array, occure;
   private int[][] twoD, foreAdd, twoDSeed;
   private double[][] rand, code;
   private int L = 0;
   private char[] har;

   public MatrixHouse(){
      twoD = new int[0][0];
   }

//*****************************************************************
   //methods in order alphabetically

   public char charAt(int count){
      return har[count];
   }

   public int[] getArray(){
      return array;
   }

   public char getChar(){
      char character;
      character = (char)occure[occure.length - 1];
      return character;
   }

   public int getL(){
      return L;
   }

   public int getLength(){
```

```
    return array.length;
}

public int getOLength(){
    return occure.length - 1;
}

public double[][] getRand(){
    rand = new double[twoDSeed.length][twoDSeed.length];
    for (int i = 0; i < twoDSeed.length; i++){
        for (int i2 = 0; i2 < twoDSeed.length; i2++){
            rand[i][i2] = (double)(twoDSeed[i][i2]);
        }
    }
    return rand;
}

public void makePos(int size){
    har = new char[size];
    for (int i = 0; i < har.length; i++)
        har[i] = '`';
    char charactera;
    charactera = (char)occure[occure.length - 1];
    for (int i = 0; i < occure.length - 1; i++)
      har[occure[i]] = charactera;
}

public int seed (){
  L = array.length;
  int s = (array[array.length - 1] / ((array.length + 1) * 26)) - (array.length + 1);
  int c = array.length - 2;
  int g = 0;
  if ( c >= 2) {
     g = (s - ((c * c) + (c - 2)));
  }
  else if ( c < 2){
     g = s;
  }
  return g;
}

public void setOccure(Matrix g, int[] w, int length){
    foreAdd = new int[length][length];
    for (int i = 0; i < length - 1; i++){
        for (int i2 = 0; i2 < length - 1; i2++){
            foreAdd[i][i2] = (int)(w[i] * g.get(i, i2));
```

```
            }
        }
        occure = new int[length - 1];
        for (int i = 0; i < occure.length; i++){
            for (int i2 = 0; i2 < occure.length; i2++){
                occure[i] += foreAdd[i2][i];
            }
            //System.out.print (" " + occure[i] + ":");  //un-comment for printing of occurances
        }
    }

    public void setSeed (int seed,int l){
      //take size of array and build the x * x to multiply
      int root = l - 1;
      twoDSeed = new int[root][root];
      for (int i = 0; i < root; i++){
            for (int i2 = 0; i2 < root; i2++){
                twoDSeed[i][i2] = seed;
                seed += 1;
            }
        }
    }

    public int[] setSmall(int[][] a, int b, int c){
        //set a to littleArray[]
        array = new int[c];
        for (int i = 0; i < c; i++)
            array[i] = a[b][i];
        return array;
    }

}//closes
```

Appendix #4

Example of Program Output Code

"If everything seems under control, you're just not going fast enough." - Mario Andretti

Output Code:

5614571858227670-2288429584264-25646580859705070-21246361252481258601264721270841276961283081289201295321301441307561313681319801325921185602443964475245108454644582046176465324688847244431862106911081492562375903796638342387183909439470398462784621683917004171691378023554036023365063698937472379553843838921260002738075627744585023283533169335033383734171200202506825110351524519455236652787532085362937960228038282932862028911292021965622711327367276212787528129212942199621231872217402176881797418260185461310421786118066182711196021027810404873625161452067525205297353426538795433254785369202780921832227402818395221562345202542755764056217439176701790110530254975612530429099216842424382453333282275420520-2

"Obstacles are those frightful things you see when you take your eyes off your goal." - Henry Ford

Output Code:

2344246225803250-2439844785928-2719672967696-22982730118304093070030991312823157327846-22329723544237912403824285245322225621020212702702017108-2597360786136-22444224668248942512017940-27561676171767267728177836783917894679501800568061181166618541-221953222047322141422235522329622423722517822611922706022800122894222988323082423176523270623364723458823552919136025657457049575245799958474589495942432760-29726995010174103981062298282493264992650526511265172652326529265352654126337482601663096602689571885824-28457861887797410-22080621047212882152914664-2437984417444550449264530245678276642107261100911292115757176-24809748599491014960350105506075110927846-210633107986968-2857887425720-28669951040-210177103088424-212304124648320-214408145729464-2217023671456-2

"Do not worry about your difficulties in Mathematics. I can assure you mine are still greater." - Albert Einstein

Output Code:

10293104221055111050-2648465539984-215783160151624716479167111694317175191882311121312141313161314181315201316221317241318261319281320301321321322341323361324381325401326421327441328461329481242424-259272597786028460790612966180262308322922-240781414324208342734433854403644687453384598946640315122887490

017488-284208847928537685960865448712887712882968888050336-222882508272829
482652-253508539925447654960554445592856412568963614 0-221863220772229113910-
237747380663838538704390233934227664-2125751270010712-239912405684122441880 4
253643192438484450445160458163182 4-25563572058775200-212176124111264612881 93
60-215155155641597316382167919100-210496610576010655410734810814210893610973
011052411131811211255848-241236417514226642781432964381144326241 02-217344176
12178801814811388-213380135289672-22862305432462730-2878289097488-26586670063
44-2324634362080-2591260544680-211282114467488-210113102886344-2