Group A

[redacted]

Math 485

Project 4

September 11, 2015

OCRAI Technical Report

In response to recent security breaches, OCRAI has asked that we create a nontrivial method of encryption to keep company information secure. Because the security breaches have occurred when company employees have accidentally sent text messages to the wrong recipient, OCRAI has asked that our encryption system be capable of encrypting a plaintext message of up to 140 characters in length (the length of a standard text message) and outputting a ciphertext that can be sent in, at most, 700 characters (the length of five text messages). The user must be able to input the message in standard characters.

As a result, we have created an algorithm that converts each letter into a three-digit number. To do so, we begin by randomly choosing a letter to be assigned the value of 1. The next letter in the alphabet is assigned the value of 2; the next, 3. This continues throughout the alphabet until each letter has been assigned a numeric value. We then choose any three-digit number between 000 and 999 that is equivalent to that letter's value, mod 26. This three-digit number is the ciphertext translation of the plaintext letter. We then take each letter of the outgoing message and convert the letter into its three-digit ciphertext equivalent. The three-digit numbers are written one after the other, not separated by spaces, periods, or other marks. Because of this, punctuation marks may be included or left out, as desired.

However, the space character is replaced by a period followed by a space ('. ') in order to more clearly show the beginning and ends of words.

As an example, suppose we assign the letter *D* a value of 1, *E* a value of 2, and so forth. Then we can choose any three-digit number equivalent to 1 (mod 26) to represent *D*, any three-digit number equivalent to 2 (mod 26) to represent *E*, etc. For instance, we might choose 027, 391, or 625 to represent *D*, and 080, 184, or 730 to represent *E*. By following this pattern, we could represent the alphabet as follows:

Letter	A	B	C	D	E	F	G
Numeric Value	24	25	26	1	2	3	4
Ciphertext	596	805	000	391	730	523	394

This encryption system creates roughly 1.53 x 10^{55} different ciphers. There are 26 ways to shift the alphabet (i.e., to assign a numeric value to each consecutive letter of the alphabet). Because there are approximately 38 different values between 000 and 999 for each numeric value (mod 26), we can assign one of these 38 three-digit cipher-values to each letter. Therefore, with 26 shifts of the alphabet, and 38 different ways to represent each of the 26 letters, there are a total of 26 x (26^38) ways to encrypt the alphabet, or 1.53 x 10^{55} different ciphers. While this encryption algorithm creates more than a septendecillion ciphers, the receiver can relatively easily decrypt the messages once they have the key.

The sender must begin each message with the key words "I am". According to the alphabet shift listed above (with *D* equal to 1, etc.), "I am" could be encrypted as

"162. 596790". From this key, the receiver can take the first value (mod 26) and see that 162 $\equiv$ 6 (mod 26), and therefore the letter *I* is assigned the numeric value of 6. The receiver may then number the rest of the alphabet—*J* would be assigned the numeric value of 7, etc. He or she will then find the value (mod 26) of each three-digit "letter" throughout the message. By matching this value with its associated letter according to the alphabet shift indicated by the key, the receiver can fully decrypt the message.

This algorithm allows us to encrypt any text messages exchanged between company employees. Moreover, because each plaintext character is replaced only by three ciphertext characters (with exception of the space character, which is replaced only by two characters), the maximum message length would be 420 characters, well within the limit requested by OCRAI. In accordance with OCRAI's desires that such information leaks not happen again, we are confident that this encryption algorithm will provide increased security to the company.

Group B

09/10/15

From the best IMC team: [redacted]

To: ORCAI

The Sasquatch Cipher

We were sorry to hear about the potential disaster that could happen to your company with sensitive information being sent to wrong recipients. However, we were excited at the opportunity to create a completely unique and practically unbreakable cipher for your company in order to ensure security of this important information. In fact, this cipher is so good, it is easier to spot Sasquatch than it is to crack the cipher; hence the name.

First we will explain the key to the cipher. The key has two components that are separated by a period (i.e. 123.1, the first component being "123" and the second component being "1"). First, we will address the first component of the key.

The first part of the key will be given as any random number that is up to 9, unique digits composed of integers 1 to 9. For example: "123", "47892", or "159". Remember, the first part of the key cannot consist of any repeated integers such as "115," "3345," or "777777."

Next, we use each digit of the first part of the selected key, in order, one at a time, and assign them to the first few letters of the alphabet. Example: for the key whose first part "742," we assign A-7, B-4, C-2. Next, we proceed to go back to the next smallest integer and assign it to the next letter. So we return to our example and we would assign D-1. We simply count upwards, and assign each integer to the next letter. Remember however, once an integer has already been assigned to a letter, you may not use that integer again. Simply skip it and use the next highest integer. Thus in our example, E-3 (since C-2), F-5, G-6, and so on and so forth until you reach the letter Z. We will then assign the next highest integer to [ ] (space, being able to have spaces between the words in our plain-text message).

Example: Key is 485.3 (NOTE: we have only covered the "485" component of our key)

A-4, B-8, C-5, D-1, E-2, F-3, G-6, H-7, I-9, J-10, K-11, L-12, M-13, N-14, O-15, P-16, Q-17, R-18, S-19, T-20, U-21, V-22, W-23, X-24, Y-25, Z-26, [ ]-27.

We can then express a phrase by assigning each letter and space to its' corresponding number and separating each number by a period in order to keep the order.

Example (using the key above):

plaintext: Josh Bundy should not be confused with Ted Bundy.

after assigning plaintext to integers:

10.15.19.7.27.8.21.14.1.25.27.19.7.15.21.12.1.27.14.15.20.27.8.2.27.5.15.14.3.21.19.2.1.27.23.9.
20.7.27.20.2.1.27.8.21.14.1.25

Once we have assigned each plaintext letter to a corresponding number, we use the second component of our cipher key. We will proceed to send each number through a quadratic equation in the form of x^2+b, the second component of our key representing b in the equation. In the example above, the second component of our key is "3" so the equation through which we will send each number is x^2+3.

Example (continuing from above)

final ciphertext:

103.228.364.52.732.67.444.199.4.628.732.364.52.228.444.147.4.732.199.228.403.732.67.7.732.
28.228.199.12.444.364.7.4.732.532.84.403.52.732.403.7.4.732.67.444.199.4.628

Thus, once we have a ciphertext and the key, in order to decipher the message we would go backwards, taking the inverse of the equation and assigning each letter of the alphabet to its' corresponding number, based on the first component of the key.

We can assure you that this cipher is very secure and will help your company succeed in all of their endeavors by keeping their trade secrets and important information safe and in the right hands. Good luck!

Group C

Encryption Method Report

Math 485

[redacted]

September 2015

Dear OCRAI:

IMC is a global leader in the creation of cryptosystems to create secure communication methods and we are glad to assist you. As your organization has learned in an unfortunate matter, without a secure cryptosystem to communicate, it is very easy for information to be captured or misplaced. Our team has created a custom protocol with which to secure your information so that it can easily be encrypted, transmitted or stored, and then decrypted as necessary. Initially, we felt that we might attempt a system using a variation of a cipher on the English alphabet but we then developed the method described further in this letter that allows for information to be encrypted with only a doubling of the message size and using a 10 digit key. We feel that this system will be most secure and useful as OCRAI conducts daily business operations.

In Kerckhoffs' La Cryptographie Militaire, he discusses that the security of any cryptographic system should be based on the key and not on the obscurity of the encryption algorithm. (Kerckhoffs 1883) This was one of the main ideas that guided our creative process of inventing an encryption algorithm.

Our idea is a simple one that simply involves a process of scrambling the order of the plaintext, mixed with random dummy text. Below we outline the algorithm involved in encryption, after which an example of encryption will be provided.

Encryption

1. Create a plaintext message which has 140 characters or fewer.
2. Randomly generate a 10-digit key using alphanumeric characters (0-9).
3. To ensure security, the plaintext message must be padded to reach a character length of 150. This padding is accomplished by adding randomly generated alphabetic characters after the plaintext and QQ to denote the end of your plaintext portion and the beginning of the padding text.
4. Randomly generate dummy text that is equivalent in length to your plaintext message.

5. Assign each letter of your plain, and dummy, text a number (0-25) based on its position in the alphabet.
6. Add the numerical values of your dummy text to the numerical values plaintext to obtain a new, obfuscated, plaintext document.
7. Insert the dummy text in the even positions of your plaintext, thus every character of the original message is now separated by a character of the dummy text. Note: the first character of this obfuscated text should be a character of your plaintext, not the dummy text.
8. The text is then broken into 10 character blocks. This is another reason why padding text needs to be added after our plaintext message, to reach a multiple of 10 for the length of the entire text to encrypt.
9. The first n characters of each block are then moved from the front of the block to the back of the block. n is determined by the integer value of the first character of the key.
10. The first character of the entire text will then be moved from the front to the back. Effectively shifting the text by one character to enhance security and ensure the the 10 character blocks are not the same for the second scrambling.
11. After this first character shift, separate the text again into 10 character blocks and repeat steps 7 and 8, however this time using the second integer value of the key to determine the number of characters that are moved from the front of the 10 digit block, to the back of the same block.
12. Continue this process until all of the values of the key have been used in scrambling the blocks

Example

1. Plaintext: “Hello I am currently studying math” length = 29, remove spaces.
2. Key: 4837291021
3. Dummy text: “aptent taciti sociosqu seesguita” length = 29, remove spaces.
4. Add QQ and padding text at the end of your plain text to reach 150 characters, thereby increasing security.
    a. Therefore your plaintext would now be “...studying mathQQajdfinaklisd…”
    b. for simplicity and readability, the padding has not been included in this example.

5. Plaintext value, using (0-25) for (a-z) values:

“7-4-11-11-14-8-0-12-2-20-17-17-4-13-19-11-24-18-19-20-3-24-8-13-8-12-0-19-7...”

Dummy text value:

“0-15-19-4-13-19-19-0-2-8-19-8-18-14-2-8-12-18-16-20-18-4-4-18-6-20-8-19-0...”

5. Added for new plaintext value. plain(1) + dummy(1) = new(1) - … - plain(n)+dummy(n) = new(n):

“7-19-30-15-27-27-19-12-4-28-36-25-22-37-21-19-38-36-35-40-21-28-12-31-12-32-8-38-7...”

6. Intersperse new plaintext and dummy text values. p(1), d(1), p(2), d(2), … , p(n), d(n):

“7-0-19-15-30-19-15-4-27-13-27-19-19-19-12-0-4-2-28-8-36-19-25-8-22-18-37-14-21-2-19-8-38-12-36-18-35-16-40-20-21-18-28-4-12-4-31-18-12-6-32-20-8-8-38-19-7-0...”

7. Take the ever 10 characters of the text and separate them into blocks (in this step the QQ has been added to reach a multiple of 10, this would already be included when encrypting a full message interspersed evenly with the dummy text):

“7-0-19-15-30-19-15-4-27-13-27-19-19-19-12-0-4-2-28-8-36-19-25-8-22-18-37-14-21-2-19-8-38-12-36-18-35-16-40-20-21-18-28-4-12-4-31-18-12-6-32-20-8-8-38-19-7-0-16-16...”

8. The first n characters are then moved from the front to the back:

a. for simplicity, the rotation of only one block is shown.

n = 4 (see first value in key)

“7-0-19-15-30-19-15-4-27-13” => “30-19-15-4-27-13-7-0-19-15”

9. This leaves us with the following ciphertext after the first key value (n) has been used to shift ALL of the blocks in the message:

“30-19-15-4-27-13-7-0-19-15-12-0-4-2-28-8-27-19-19-19-22-18-37-14-21-2-36-19-25-8-36-18-35-16-40-20-19-8-38-12-12-4-31-18-12-6-21-18-28-4-38-19-7-0-16-16-32-20-8-8...”

a. We then move the first character to the back of the ciphertext to ensure that the blocks are all different on our next scrambling:

“30-19-15-4-27-13-7-0-19-15-12-0-4-2-28-8-27-19-19-19-22-18-37-14-21-2-36-19-25-8-36-18-35-16-40-20-19-8-38-12-12-4-31-18-12-6-21-18-28-4-38-19-7-0-16-16-32-20-8-8...”

=>

“19-15-4-27-13-7-0-19-15-12-0-4-2-28-8-27-19-19-19-22-18-37-14-21-2-36-19-25-8-36-18-35-16-40-20-19-8-38-12-12-4-31-18-12-6-21-18-28-4-38-19-7-0-16-16-32-20-8-8-30...”

10. We then repeat this process with the second numerical value of the key:

“19-15-4-27-13-7-0-19-15-12-0-4-2-28-8-27-19-19-19-22-18-37-14-21-2-36-19-25-8-36-18-35-16-40-20-19-8-38-12-12-4-31-18-12-6-21-18-28-4-38-19-7-0-16-16-32-20-8-8-30...”

a. (second key value is 8)

“19-15-4-27-13-7-0-19-15-12” => “15-12-19-15-4-27-13-7-0-19”

b. The new ciphertext is then shifted:

“15-12-19-15-4-27-13-7-0-19-19-22-0-4-2-28-8-27-19-19-8-36-18-37-14-21-2-36-19-25-12-12-18-35-16-40-20-19-8-38-4-38-4-31-18-12-6-21-18-28-8-30-19-7-0-16-16-32-20-8...”

=>

“12-19-15-4-27-13-7-0-19-19-22-0-4-2-28-8-27-19-19-8-36-18-37-14-21-2-36-19-25-12-12-18-35-16-40-20-19-8-38-4-38-4-31-18-12-6-21-18-28-8-30-19-7-0-16-16-32-20-8-15...”

11. This process is continued until the entire key has been iterated through once.

Dummy characters need to be added at the end of your plaintext before interspersing it with the dummy text, causing your plain text to be 150 characters in length consistently. After the dummy text has been added to the values of your plaintext, and interspersed, you will have a ciphertext of 300 characters, no matter the initial message size. In addition to this, QQ will be added after the end of your plain text, before the padding text, to ensure easy decryption and additional security.

In conclusion, this method of encryption allows us to rely wholly on the key as our life line of security. While the algorithm is quite obscure, the key ensures that the message is practically impossible to decrypt without knowledge of the key. We feel that this makes the encryption method simple and secure, allowing it to be easily implemented and used regularly.

Sincerely,

IMC

Citations:

Kerckhoffs, A. (1883). *La Cryptographie Militaire.*

Group D [redacted]

OCRA, Inc Technical Report

We received a letter from OCRA, Inc, explaining a problem they had. They have had several text messages containing sensitive and classified information sent to clients and rivals by mistake. OCRA, Inc has asked us to write an encryption method so that the information passed through text can remain classified to those outside of the company. Eventually, a team of engineers will write software to automatically encrypt and decrypt these messages between OCRA, Inc employees. The messages will remain encrypted to anyone outside the company receiving these texts by mistake. We have created an encryption method by which 140 characters will expand to close to 280 characters, and be represented solely by numerals. We want to make sure that our instructions and method cannot be misinterpreted, so as to be clear to the engineers who will eventually write this program. We would be very grateful if you would read this report, and encrypt a message to us using our encryption method. This will help us check our instructions. Thank you for your time and cooperation.

Here are the steps to our encryption method. This method requires an encryption key, consisting of up to ten (10) alpha-numeric characters.

"Stapler" and "2wood"

are examples of two possible encryption keys. This is our first step.

Step two: We begin by creating a 6x6 matrix. The rows and columns are numbered based on the following: We count the number of characters in our encryption key; that number is then the label of the first row and column. We continue labeling the rows and columns in ascending numeric order to 9. Once 9 is reached, if there are remaining rows and columns to label, we continue numbering with 0, 1, 2, etc. For example:

"Stapler" has 7 characters. Thus, our matrix looks like this:

	7	**8**	**9**	**0**	**1**	**2**
7						
8						
9						
0						
1						
2						

"2wood" has 5 characters, so our matrix looks like this:

	5	**6**	**7**	**8**	**9**	**0**
5						
6						
7						
8						
9						
0						

*Note: if we choose an encryption key with 10 characters, we begin labeling rows and columns starting with 0, 1, 2, etc.

Step Three: We then start filling in our matrix. We begin by writing our encryption key, starting in the first row, filling in left to right. We leave out any duplicate letters or numbers that may appear in our key.

For example, our "stapler" matrix begins this way:

	7	**8**	**9**	**0**	**1**	**2**
7	S	T	A	P	L	E
8	R					
9						
0						
1						
2						

And our "2wood" matrix begins like this:

	5	**6**	**7**	**8**	**9**	**0**
5	2	W	O	D		
6						
7						
8						
9						
0						

Step 4: we then completely fill our matrix, starting from a to z, omitting any letters in our encryption key, then we fill in the numerals 0 to 9, also omitting any numerals occurring in our key.

Our complete "stapler" matrix then looks like this:

	7	**8**	**9**	**0**	**1**	**2**
7	S	T	A	P	L	E
8	R	B	C	D	F	G
9	H	I	J	K	M	N
0	O	Q	U	V	W	X
1	Y	Z	0	1	2	3
2	4	5	6	7	8	9

And our complete "2wood" matrix looks like this:

	5	**6**	**7**	**8**	**9**	**0**
5	2	W	O	D	A	B
6	C	E	F	G	H	I
7	J	K	L	M	N	P
8	Q	R	S	T	U	V
9	X	Y	Z	0	1	3
0	4	5	6	7	8	9

Now that our matrix is completed, we can use it both to encrypt and decrypt messages.

Step 5: Encryption. We start with the message we want to encrypt. This message can contain numbers, letters, and spaces. We will call this the "plaintext." We start with the first character of the plaintext, and proceed as follows:

- If the character is a number or letter, we locate it on the matrix.

    - We then replace this character with its location, referred to by first its row number, then its column number. For example, the letter “t” in our “stapler” matrix, is on row 7, and column 8. We will therefore replace “t” with “78” in our ciphertext.
- If the character is a space:
    - We first look at the numbers labeling the rows and columns. We notice that there are 4 numerals from 0 to 9 that we have not used to label the matrix. These numerals are all designated to represent spaces in the plaintext. These are interchangeable, so each time we come to a space, we pick one of these four numerals at random. For example, a space in our plaintext, when encrypting using our “stapler” matrix will be replaced by any of the following: 3, 4, 5, or 6. These are the numerals not used in labeling the rows and columns of the “stapler” matrix.

For example, if our plaintext is, “The duck swims on the lake,” and we are encrypting this message using our “stapler” matrix, the resulting cryptext will be:

78977248009899057701989177407925789772471799072

If we are encrypting the same plaintext on our “2wood” matrix, the resulting cryptext will be:

88696645889657618756607887357792886966177597666

By using this encryption method, we have found a way to represent a common message that looks nothing like the original, and would be quite hard for someone to decrypt without the encryption key. We hope that this method can be efficiently coded to the cell phones of the employees at OCRA, Inc. In this way, their information can remain confidential and within their company. Any outside cell phones receiving an encrypted text will receive nothing but a string of numbers, that they will not be able to easily cipher.

Group E

TO: OCRAI Board of Directors

[redacted]

DATE: 11 September 2015

SUBJECT: Project 4

INTRODUCTION:

OCRAI has recently experienced many security breaches due to text messages received by unintended recipients. The texts contained important company information that needs to remain confidential, such as trade secrets. Fortunately, none of this classified information was disclosed to the media or our competitors. Had such a thing occurred, the company would have experienced much embarrassment and potential harm. In order to prevent such a catastrophe from arising, OCRAI has contracted with us to devise a method for keeping all correspondence within the company secure.

PARAMETERS:

In order to make the encryption and decryption methods compatible with text messaging the following parameters must hold within our method:

- ❿ It must take a text message of up to 140 characters (characters in 1 standard text message) and output a text message of up to 700 characters (characters in 5 standard text messages).
- ❿ It must have an encryption key less than or equal to 10 letters long.
- ❿ It must only contain symbols compatible with standard keyboards.

METHOD

We have developed an encryption method in order to manipulate the plaintext of a message such that those desiring to intercept the message will be unable to read it. The encryption method operates as follows:

1. The application counts the letters in each word. (ex: cow = 3)
2. Each letter in the word is shifted forward by the number yielded in step 1. (ex: cow >> frz) (note: If the end of the alphabet is reached, return to the beginning. tax >> wda)
3. Following this shift, the entire message is then shifted again by a random positive integer. This step is the encryption key. (ex: If key = 5, cow >> frz >> kwe)
4. The recipient's application will then decrypt the ciphertext message.

In order to decrypt the message, the encryption method is reversed. For example, if the key = 3, then shift each letter in the ciphertext back 3 steps in the alphabet. Following that, count the number of letters in each word and perform step 2 above with shifting the letters backward in the alphabet instead of forward.

With such a dynamic encryption method that changes with each word, the safety of OCRAI's communication and intellectual property is ensured.

CONCLUSION:

The protection of sensitive data and information is essential for the well-being of a business. The cipher we have created involves shifting each letter according to the length of each word and then shifting the entire alphabet depending on the encryption key. Through the use of this encryption method, OCRAI's confidential information will remain secure and the business will be protected from potential embarrassment and harm.

Group F

Independent Mathematical Contractors, Inc.

136 TMCB
Provo, UT 84602
11 September 2015

OCRA Creative Recursive Acronyms, Inc.

485 Primality Way
Provo, UT 84604

Dear OCRAI,

Privacy when sending messages is very essential in any business. We are glad you came to us to help in your time of need. We propose a simple Electronic Codebook (ECB) solution to your encryption problem. It is a simple system that can work for text messages and can be extended to handle your future needs.

The encryption method is simple. Each character is encoded in ASCII and can be represented as two hexadecimal digits or eight binary digits (bits). A conversion table for many of the commonly used characters is listed below.

Char	Hex	Bin	Char	Hex	Bin	Char	Hex	Bin	Char	Hex	Bin
<space>	20	0010 0000	5	35	0011 0101	M	4D	0100 1101	f	66	0110 0110
!	21	0010 0001	6	36	0011 0110	N	4E	0100 1110	g	67	0110 0111
“	22	0010 0010	7	37	0011 0111	O	4F	0100 1111	h	68	0110 1000
#	23	0010 0011	8	38	0011 1000	P	50	0101 0000	i	69	0110 1001
$	24	0010 0100	9	39	0011 1001	Q	51	0101 0001	j	6A	0110 1010
%	25	0010 0101	:	3A	0011 1010	R	52	0101 0010	k	6B	0110 1011
&	26	0010 0110	;	3B	0011 1011	S	53	0101 0011	l	6C	0110 1100
‘	27	0010 0111	?	3F	0011 1111	T	54	0101 0100	m	6D	0110 1101
(	28	0010 1000	@	40	0100 0000	U	55	0101 0101	n	6E	0110 1110
)	29	0010 1001	A	41	0100 0001	V	56	0101 0110	o	6F	0110 1111
*	2A	0010 1010	B	42	0100 0010	W	57	0101 0111	p	70	0111 0000
+	2B	0010 1011	C	43	0100 0011	X	58	0101 1000	q	71	0111 0001
,	2C	0010 1100	D	44	0100 0100	Y	59	0101 1001	r	72	0111 0010
-	2D	0010 1101	E	45	0100 0101	Z	5A	0101 1010	s	73	0111 0011
.	2E	0010 1110	F	46	0100 0110	\	5C	0101 1100	t	74	0111 0100

/	2F	0010 1111	G	47	0100 0111	_	5F	0101 1111	u	75	0111 0101
0	30	0011 0000	H	48	0100 1000	a	61	0110 0001	v	76	0111 0110
1	31	0011 0001	I	49	0100 1001	b	62	0110 0010	w	77	0111 0111
2	32	0011 0010	J	4A	0100 1010	c	63	0110 0011	x	78	0111 1000
3	33	0011 0011	K	4B	0100 1011	d	64	0110 0100	y	79	0111 1001
4	34	0011 0100	L	4C	0100 1100	e	65	0110 0101	z	7A	0111 1010

For example, the word "Text" would be written as seen below:
5 4 6 5 7 8 7 4 Which could then be converted to the following:
0101 0100 0110 0101 0111 1000 0111 0100 This is the code of the plaintext.

To encrypt the plaintext, we need a key which consists of four hexadecimal digits (labeled Hex in the chart). A table listing the hexadecimal digits with their binary representations is listed below. In our example, we use 74AF as our key.

Hex	Bin		Hex	Bin		Hex	Bin		Hex	Bin
0	0000		1	0001		2	0010		3	0011
4	0101		5	0101		6	0100		7	0111
8	1000		9	1001		A	1010		B	1011
C	1100		D	1101		E	1110		F	1111

We write the key in terms of bits.
7 4 A F
0111 0100 1010 1111

We write the key under the plaintext and keep repeating the key until we reach the end of the text.
0101 0100 0110 0101 0111 1000 0111 0100 plain text written as bits
0111 0100 1010 1111 0111 0100 1010 1111 The key repeated

Now we perform a bitwise exclusive-or or add each column using modulo 2. (In other words, 0+0=2, 1+1=0, 0+1=1+0=1)

0101 0100 0110 0101 0111 1000 0111 0100 plain text written as bits
<u>0111 0100 1010 1111 0111 0100 1010 1111</u> The key
0010 0000 1100 1010 0000 1100 1101 1011 This is now the ciphertext.

To decrypt the message, we will write the key beneath and add the digits mod2 again.

0010 0000 1100 1010 0000 1100 1101 1011	The ciphertext.
0111 0100 1010 1111 0111 0100 1010 1111	The key
0101 0100 0110 0101 0111 1011 0111 0100	This is the original plain text written as bits.

Using our key in table 1, we can convert the bits back into hexadecimal
54 65 78 74 and then convert those into characters and we get
T e x t.

So, even if the encryption method is known, your message will remain secure as long as no one knows they key. In addition, our encryption method can be strengthened by increasing the key size. It can also be extended to provide encryption for any electronic communication. Finally, with a little added complexity, we can convert our encryption algorithm to Cipher Block Chaining (CBC) to allow even more secure encryption for audio and image messages. We hope you return to us for your future encryption needs.

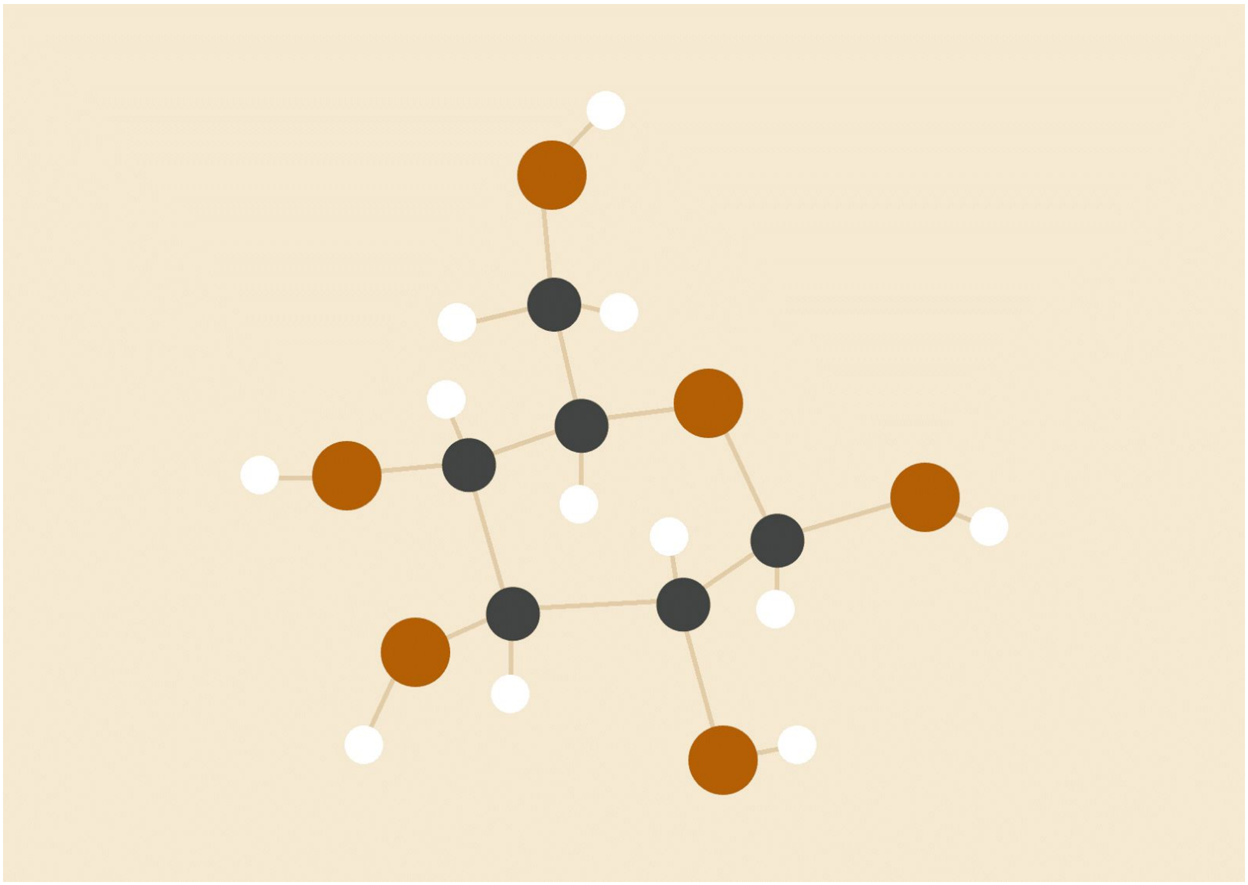
Sincerely,

Independent Mathematical Contractors, Inc.

Group G

Cryptography Report

A report on our cipher



[redacted]

09.010.2015

Math 485 - Cryptography

INTRODUCTION

We were presented with the problem of keeping communications secure between the OCRA Creative Recursive Acronyms, Inc. We were asked to create a cryptosystem, and likewise an encryption key, for the engineers at OCRAI to create a mobile application to encrypt text communication. The requirements for our cryptosystem stated that given a plaintext of standard text message size, the ciphertext must be between 500 to 700 characters. In addition, the ciphertext should be able to be typed on a standard keyboard. We were also asked that the encryption key be no longer than 10 letters long, should the key be using a sequence of letters; the reason behind this is that it would slow down the processing power of the application.

OUR CRYPTOSYSTEM - THE PROCESS OF OBTAINMENT AND HOW IT WORKS

Our hope for our cryptosystem was that it would be one that we would find secure and reliable. We were trying to find a cryptosystem that would take some time to decrypt, but wouldn't be too overbearing for the application's processing power. We were also hoping to create a system where the key could be manipulated and not very predictable, thereby creating a complex manner in which to encode a message.

We first considered ways to use a sequence of letters as our key. This design involved assigning a unique sequence of five letters to each letter of the alphabet. The concern we had with this method was that the assignment was very predictable and difficult to randomize, and thus unable to securely protect the messages. Likewise, it did not allow for a wide variety of encryption keys, which meant that if the key was discovered, there would a limited amount of keys to try before every key would be broken.

We also considered a method that involved taking an irrational number and assigning the values of each digit placement to a unique letter in the alphabet. However, as we tried this method, we realized that we found ourselves in a similar situation to our first method; the number of possible numerical values that we could truly use would limit the amount of encryption keys. We also realized that this method would be very difficult to manipulate, defeating one of the purposes we were aiming to achieve.

Another method we had considered was using a displacement key. This particular method would allow for better manipulation. As we experimented with this method, we realized that by using a displacement key, the amount of time it would take to encrypt or decrypt a short message was more than intended for the purposes of this particular project. The reason behind this is based on the fact that this particular method requires the decryptor to test every possible displacement for consistencies. This method would create a very secure key; however, as already stated, it would be outside the parameters of what this particular assignment entailed.

After the above attempts, we decided to use a cryptosystem that uses random digit tables to encrypt and decrypt our codes. The idea behind us using a random digit table was that it had the potential to securely encrypt data and the messages we would send. The use of a random digit table for our cryptosystem follows a simple process. On the left hand side of the random digit table, like the one we have attached to the end of this particular report, is a column titled lines. We would choose a line, and then select an entry. Depending on the digit table, there will be more entries per line or fewer. Our example table, as you will notice, contains eight entries per line. After selecting a line and an entry on that line, the encryptor would begin to assign a plaintext letter to each unique five digit value, starting with *a*. The next unique five digit value would be assigned to *b*, the next to *c*, and so forth until the whole alphabet was covered. This method we are describing involves reading from left to right, starting on the selected row and entry. The key will totally depend on which row was used and which entry it starts on. Especially with unique random digit tables, this method can prove to be a bit of a challenge.

CONCLUSION

In summary, we solved the problem of secure text communication by use of a random digit table to generate an encryption key. This method satisfies the ciphertext character limit, in addition to producing multiple encryption keys simply by choosing a new line on the table or by generating a completely different table. The application design should be able to select a line on a random digit table, and assign unique five digit values to each letter in the alphabet, starting with *a*. In the decryption of the text, the same line used to encrypt the message should be used. In utilizing our design to create the mobile application, we encourage the engineers to code and program the generation of various random digit tables to heighten the security of text communication. The reason of this would be to have unique random digit tables built into the programming, ensuring that tables will remain private to the company not for the public use.

871364336755892330636222456027950525458041842558925589243367871361486354580751865589229077957614705287136418425602747052622245602755892290775602762224558928159854580710359505295761138734184281507290777518675186545808186862224560277518654580957614705254580957617518662224290774705295052957611387355892815986222456027751865458061683735922907795761470529090854580815989576133063545802907755892622246168356027622243306361683815989505295761622245589262224616835602795052871366222455892433679576147052950526168329077330636222461683290777518675186622244705214863751862907761683735927518654580330634184255892

1297594591003600242855588819404356313048967670036000360945911297533586130485171900360477811325806928129759676781940069285558881940003604778181940555880036041098130482382243563132587070896767451444778151719517191304835963003608194051719130481325806928130481325851719555884778106928435631325870708003604109855588819405171913048791774839447781132580692856934130484109813258124281304847781003605558879177819405558802428791774109843563132585558800360555887917781940435631297555588003609459113258069284356379177477810242855588791774778151719517195558806928335865171947781791774839451719130480242896767003603390684568634949416546514875038456806340100567137926411571959776297762063408596897762063408750345984100569281306340465146349446514875037655097762928130548194292941658456857195571959776294831859684651494831859682641171379634944651437033054811005671379422118456805481634949416597762422119281346514942929776246514634949429284568948312641197762977625083063494928139776233906977625719505481928139776245984845689281306340465149483185968100567137992813859681005684568571958456887503634949776284568942929416597762928138750346514875036349410056941659776257195422111005663494941659776292813875038750397762977626349494165977629281397762459848456892813063403

TABLE B
Random digits

Line								
101	19223	95034	05756	28713	96409	12531	42544	82853
102	73676	47150	99400	01927	27754	42648	82425	36290
103	45467	71709	77558	00095	32863	29485	82226	90056
104	52711	38889	93074	60227	40011	85848	48767	52573
105	95592	94007	69971	91481	60779	53791	17297	59335
106	68417	35013	15529	72765	85089	57067	50211	47487
107	82739	57890	20807	47511	81676	55300	94383	14893
108	60940	72024	17868	24943	61790	90656	87964	18883
109	36009	19365	15412	39638	85453	46816	83485	41979
110	38448	48789	18338	24697	39364	42006	76688	08708
111	81486	69487	60513	09297	00412	71238	27649	39950
112	59636	88804	04634	71197	19352	73089	84898	45785
113	62568	70206	40325	03699	71080	22553	11486	11776
114	45149	32992	75730	66280	03819	56202	02938	70915
115	61041	77684	94322	24709	73698	14526	31893	32592
116	14459	26056	31424	80371	65103	62253	50490	61181
117	38167	98532	62183	70632	23417	26185	41448	75532
118	73190	32533	04470	29669	84407	90785	65956	86382
119	95857	07118	87664	92099	58806	66979	98624	84826
120	35476	55972	39421	65850	04266	35435	43742	11937
121	71487	09984	29077	14863	61683	47052	62224	51025
122	13873	81598	95052	90908	73592	75186	87136	95761
123	54580	81507	27102	56027	55892	33063	41842	81868
124	71035	09001	43367	49497	72719	96758	27611	91596
125	96746	12149	37823	71868	18442	35119	62103	39244
126	96927	19931	36089	74192	77567	88741	48409	41903
127	43909	99477	25330	64359	40085	16925	85117	36071
128	15689	14227	06565	14374	13352	49367	81982	87209
129	36759	58984	68288	22913	18638	54303	00795	08727
130	69051	64817	87174	09517	84534	06489	87201	97245
131	05007	16632	81194	14873	04197	85576	45195	96565
132	68732	55259	84292	08796	43165	93739	31685	97150
133	45740	41807	65561	33302	07051	93623	18132	09547
134	27816	78416	18329	21337	35213	37741	04312	68508
135	66925	55658	39100	78458	11206	19876	87151	31260
136	08421	44753	77377	28744	75592	08563	79140	92454
137	53645	66812	61421	47836	12609	15373	98481	14592
138	66831	68908	40772	21558	47781	33586	79177	06928
139	55588	99404	70708	41098	43563	56934	48394	51719
140	12975	13258	13048	45144	72321	81940	00360	02428
141	96767	35964	23822	96012	94591	65194	50842	53372
142	72829	50232	97892	63408	77919	44575	24870	04178
143	88565	42628	17797	49376	61762	16953	88604	12724
144	62964	88145	83083	69453	46109	59505	69680	00900
145	19687	12633	57857	95806	09931	02150	43163	58636
146	37609	59057	66967	83401	60705	02384	90597	93600
147	54973	86278	88737	74351	47500	84552	19909	67181
148	00694	05977	19664	65441	20903	62371	22725	53340
149	71546	05233	53946	68743	72460	27601	45403	88692
150	07511	88915	41267	16853	84569	79367	32337	03316

TABLE B

Random digits (continued)

Line								
151	03802	29341	29264	80198	12371	13121	54969	43912
152	77320	35030	77519	41109	98296	18984	60869	12349
153	07886	56866	39648	69290	03600	05376	58958	22720
154	87065	74133	21117	70595	22791	67306	28420	52067
155	42090	09628	54035	93879	98441	04606	27381	82637
156	55494	67690	88131	81800	11188	28552	25752	21953
157	16698	30406	96587	65985	07165	50148	16201	86792
158	16297	07626	68683	45335	34377	72941	41764	77038
159	22897	17467	17638	70043	36243	13008	83993	22869
160	98163	45944	34210	64158	76971	27689	82926	75957
161	43400	25831	06283	22138	16043	15706	73345	26238
162	97341	46254	88153	62336	21112	35574	99271	45297
163	64578	67197	28310	90341	37531	63890	52630	76315
164	11022	79124	49525	63078	17229	32165	01343	21394
165	81232	43939	23840	05995	84589	06788	76358	26622
166	36843	84798	51167	44728	20554	55538	27647	32708
167	84329	80081	69516	78934	14293	92478	16479	26974
168	27788	85789	41592	74472	96773	27090	24954	41474
169	99224	00850	43737	75202	44753	63236	14260	73686
170	38075	73239	52555	46342	13365	02182	30443	53229
171	87368	49451	55771	48343	51236	18522	73670	23212
172	40512	00681	44282	47178	08139	78693	34715	75606
173	81636	57578	54286	27216	58758	80358	84115	84568
174	26411	94292	06340	97762	37033	85968	94165	46514
175	80011	09937	57195	33906	94831	10056	42211	65491
176	92813	87503	63494	71379	76550	45984	05481	50830
177	70348	72871	63419	57363	29685	43090	18763	31714
178	24005	52114	26224	39078	80798	15220	43186	00976
179	85063	55810	10470	08029	30025	29734	61181	72090
180	11532	73186	92541	06915	72954	10167	12142	26492
181	59618	03914	05208	84088	20426	39004	84582	87317
182	92965	50837	39921	84661	82514	81899	24565	60874
183	85116	27684	14597	85747	01596	25889	41998	15635
184	15106	10411	90221	49377	44369	28185	80959	76355
185	03638	31589	07871	25792	85823	55400	56026	12193
186	97971	48932	45792	63993	95635	28753	46069	84635
187	49345	18305	76213	82390	77412	97401	50650	71755
188	87370	88099	89695	87633	76987	85503	26257	51736
189	88296	95670	74932	65317	93848	43988	47597	83044
190	79485	92200	99401	54473	34336	82786	05457	60343
191	40830	24979	23333	37619	56227	95941	59494	86539
192	32006	76302	81221	00693	95197	75044	46596	11628
193	37569	85187	44692	50706	53161	69027	88389	60313
194	56680	79003	23361	67094	15019	63261	24543	52884
195	05172	08100	22316	54495	60005	29532	18433	18057
196	74782	27005	03894	98038	20627	40307	47317	92759
197	85288	93264	61409	03404	09649	55937	60843	66167
198	68309	12060	14762	58002	03716	81968	57934	32624
199	26461	88346	52430	60906	74216	96263	69296	90107
200	42672	67680	42376	95023	82744	03971	96560	55148

Independent Mathematical Contractors, Inc.

[redacted]

Group H

136 TMCB
Provo, UT 84602

September 9, 2015

OCRAI
OCRA Creative Recursive Acronyms, Inc.
485 Primality Way
Provo, UT 84604

Security Cipher Technical Report

Introduction:

We are technical contractors who have been commissioned by your company to create a nontrivial method of encrypting plaintext. Your company has recently had several major security breaches when sending text messages, so it is our job to make sure that this doesn’t happen anymore. After creating this key, your engineers will use our cryptosystem to create a mobile application to encrypt all messages before they are sent. This cryptosystem we created roughly doubles the number of characters. We have created the following method of encrypting plaintext, and believe that it will help protect the sensitive information contained in your text messages.

Solution:

When faced with the task of encrypting your data, we had two distinct ideas. The first idea was to use a cipher that incorporated a system based on the properties of odd and even numbers. We ultimately decided that this method wasn't as secure as our second option, which is the one we decided to use. As we worked on improving security, our basic encryption method stayed the same, but we altered the complexity of the key.

The encryption method that we designed incorporates a table of all the desired letters and symbols. The table that we used includes, in this specific order: letters A-Z (in alphabetical order), space, “.” , “,” , and “?” . The way that the letters and symbols are inputted into the table is determined by the key given.

A generic example of the table is given below:

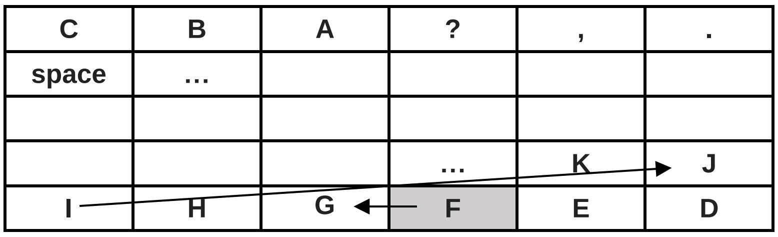
	1	2	3	4	5	6
7	A	B	C	D	E	F
8	G	H	I	J	K	L
9	M	N	O	P	Q	R
10	S	T	U	V	W	X
11	Y	Z	space	.	,	?

Each column and row is labeled with a number and it is these numbers that are paired together (column, row) and used to encrypt the letters and symbols of the plaintext. For example, the letter A would be represented as 17. When writing the ciphertext, no spaces are included in between the numbers because spaces are encrypted as well. Using the above table, the phrase "Have a good day." would be 28174105731117311183939473114717111411. A code without spaces increases the strength of the encryption and makes it more difficult to crack without the key.

The key of the code is essential to the security of the encryption method and allows for multiple variations of the encryption table used. An example key is F28LU35216. There are two distinct parts to the key. The first half tells the coder how to organize the encryption table. The first character in the key is the first letter put into the encryption table, and the number that comes next in the key is the table cell to put that letter in. Our example key would assign the letter F to cell 28 in the encryption table.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

The two letters that come next decide which direction the letters (in order) will be filled into the table. The first letter can be either R (right) or L (left) which determines the direction along the row. The L in our example key means that we are moving right to left along the row, and the letter “G” would be put in cell 27, letter “H” would be put in cell 26, etc. The second letter can be U (up) or D (down), which determines how to move from one row to the next. In our example key, the U means that letter “J” would go in cell 24, letter “K” would go in cell 23, etc. Here is how the first half of the key would change our table:



C	B	A	?	,	.
space	…				
			…	K	J
I	H	G	F	E	D

The second half of the key rearranges, or permutes, the columns of the encryption table. The first number in the second half of the key is the column number that is moved into the first column of the encryption table, the second number is moved into the second column and so forth. The column number that is not included in the permutation is placed in the last column. For our example, column 3 will be the first column.

3	5	2	1	6	4
A	,	B	C	.	?
Y	W	Z	space	V	X
S	Q	T	U	P	R
M	K	N	O	J	L
G	E	H	I	D	F

We then change the columns back to the original numbering (1-6) and we now have the final encryption table. Our final table for this encryption key would be:

	1	2	3	4	5	6
7	A	,	B	C	.	?
8	Y	W	Z	space	V	X
9	S	Q	T	U	P	R
10	M	K	N	O	J	L
11	G	E	H	I	D	F

We then use this encryption table to encode the message. Using this new table, the phrase "Have a good day." would be: 31117582114871111410410511485111171857.

In order to decrypt this message, the numbers will have to be read as pairs or triples. A number between 1 and 6 will always come before a number from 7 to 11, which keeps 111 from being interpreted erroneously.

Conclusion:

As can be seen, the technique used to manipulate our data is far from trivial. It involves several permutations of an original chart, and then relies on those permutations to encode (and decode) the text. If even one column is switched erroneously or one number is placed out of order, the entire system will be broken and the individual is unlikely to understand the encoded message being sent. In addition, the fact that each letter has the possibility of being represented by two or three numbers, depending on its position in the table, makes the code stronger. It will be initially unclear if the string of characters represents words, letters or numbers, etc. The increased length of these messages because of the multiple numbers used for each plaintext character will increase the number of possible solutions to our code, and will thereby increase the difficulty of the problem to be solved. At ten characters, the keys are within the bounds of your processing and monetary limits. We believe this will be well worth the added security of this difficult code. We hope that your team will be satisfied with this and that you will let us know about any feedback or concerns you have at your earliest convenience.

IMC

IMC Independent Mathematical Contractors, Inc.

136 TMCB

Provo, UT 84602

---

11 September 2015

OCRA Creative Recursive Acronyms, Inc.

485 Primality Way

Provo, UT 84604

Dear OCRAI:

We received your letter concerning your request for a method of preventing security breaches of text messages sent by your employees. We have put together a method for encrypting plaintext English messages that we believe will protect the security and confidentiality of your company's information. Included in this report is a description of the method for encrypting messages, a description of the method for choosing an encryption key, and three sample ciphertexts encrypted using the method described with three different encryption keys.

Our method of encryption randomly assigns three two-digit numbers to each of the characters letters A through Z (not case-sensitive), space, period, comma, and question mark, for a total of 30 characters. (Any numbers communicated through the method must be spelled out.) To choose the encryption key, we first assign three two-digit numbers from 00-89 to each of the 30 plaintext characters. This should be done randomly either by hand, with a random number generator, or with a computer program we have created which is also available for purchase. Once the encryption key is chosen, we substitute each character in the plaintext with one of the three corresponding two-digit ciphertext numbers. This is done randomly by hand or with a random number generator. The plaintext is then properly encrypted and the ciphertext is ready to be sent. The resulting ciphertext should appear as a string of numbers without spaces.

For example, say we wish to encrypt the message "Robert S. Andrews" using this method. Our program generates the following encryption key:

Plaintext → Ciphertext
A → 00, 30, 60
B → 01, 31, 61
D → 03, 33, 63
E → 04, 34, 64
N → 15, 45, 75

O → 16, 46, 76
R → 18, 48, 78
S → 19, 49, 79
T → 20, 50, 80
W → 23, 53, 83
Space → 26, 56, 86
Period → 28, 58, 88

After replacing each plaintext character with a ciphertext number (randomly chosen of the three), the ciphertext could appear

“4816610418802679585600453378462319” or “1876013478205619888630150348537 9”

depending on which of the three two-digit ciphertext numbers is chosen for each plaintext character.

We believe that this method will secure communication over employees’ mobile phones. We have decided on this method because it is more secure than a simple shift cipher. There are three numbers for each letter, making it difficult to discover all the possible numbers that represent a certain letter. Furthermore, the numbers that are chosen do not have to be in any order. For example, A → 1, B → 2, etc. The numbers assigned are random with no pattern that is easily discovered. Finally, space also has numbers associated with it, meaning that a person cannot use spaces to find where words end and begin, which would make the ciphertext easier to decrypt.

Yours sincerely,

[redacted]

[redacted]

[redacted]

Group J

IMC
Independent Mathematical Contractors, Inc.
136 TMCB
Provo, UT 84602

---

11 September 2015

OCRA Creative Recursive Acronyms, Inc.
485 Primality Way
Provo, UT 84604

Dear OCRAI,

We received your inquiry about a solution to your security breaches. We recognize that the problem was that company employees were unintentionally and erroneously sending text messages – which contained sensitive company information and trade secrets – to the wrong recipients. We also recognize the potential for this to cause acute embarrassment to the company and to harm future revenue streams if received by the wrong person. We have faced the problem of making sure that sensitive information sent out to employees through text messages is secure and protected. This is why we have created a special cipher that would be easy to implement as well as painless to understand for less cipher-savvy workers. This cryptosystem can be used to create a mobile application which will encrypt all text messages before being sent and decrypt any messages received from other company employees as desired.

Our cryptosystem changes a plaintext message into a string of numbers. The cipher uses a random assignment of numbers to letters. We used a random number generator to assign the numbers 11 through 36 to the letters A through Z. The assignment is as follows: A=11, B=17, C=14, D=16, E=34, F=24, G=29, H=18, I=19, J=35, K=12, L=28, M=20, N=31, O=33, P=22, Q=30, R=26, S=25, T=27, U=15, V=36, W=32, X=13, Y=21, Z=23. We first multiply each of the above numbers by the randomly chosen encryption key number. This number should be a two digit prime integer. When the multiplication results in a three digit number instead of a four digit number, we add a zero at the beginning. Now, all of our letters are represented by a four digit number. We then substitute the number representation for each letter in our plaintext. We delete any spaces that may be between letters or words and get rid of all punctuation. Thus, we are left with a ciphertext that is a string of numbers.

To decrypt a message, we must know the encryption key, a two digit prime integer. We first separate the string of numbers of the ciphertext into four digit numbers. This can be done by putting a space after each set of four numbers. We then divide each of those numbers by the encryption key, which should always result in a two digit number between 11 and 36. We then substitute the assigned letter (which can be found above) for each two digit number. Then, we use common sense to group the letters into meaningful words and sentences. We now have the plaintext message.

We have created a cryptosystem and cipher that can be used with any randomly chosen encryption key to encrypt text messages sent among employees. This system is small enough that a text message (of up to 140 characters) doesn't result in an encrypted message of more than 5 text messages (or 720 characters); however, it is also secure enough that someone without the encryption key would not be able to decrypt the message and intercept the information. It makes it so that any information leaked on accident will not be understood and taken advantage of by the people who could accidentally receive the text. After the careful and thoughtful work we have invested into this project, we expect that the code will start to pay back through the impossibility of breaking it by anyone who doesn’t have the key. This makes us feel secure and satisfied with the work we accomplished. Our code will be easy to implement and will be able to be used to solve the problem of information insecurity of text messages sent within your company.

Sincerely,

[redacted]
Encryption specialists
IMC Inc.

Group K

IMC
Independent Mathematical Contractors, Inc.
136 TMCB
Provo, UT 84602

10 September 2015

OCRA Creative Recursive Acronyms, Inc.
485 Primality Way
Provo, UT 84604

Dear OCRAI:

In response to your request for a nontrivial method of encrypting plaintext English messages we have established a system of encryption which is not only compatible with a standard computer or mobile device keyboard, but benefits from the use.

Our company took a long look at the text messages your members sent to the wrong recipients and devised a system which could have prevented the erroneous breach in security. However, instead of focusing on the messages themselves, our engineers noticed something far more trivial: The time stamp. The time stamp serves as a marker of when the text message was sent, but we created a second way to utilize it: An encryption key.

A time stamp contains three to four numbers indicating the time of the text message's departure as well as an acronym indicating whether the text left "ante meridiem" or "post meridiem." We have utilized these numbers to create a series of one thousand four hundred and forty unique tables which may each individually encrypt the English alphabet and by extension, your company's private conversations.

We do this by constructing the same number of columns in a table as there are hours in the time stamp. The rows begin at the number of minutes and proceed by repeatedly adding the amount contained in the ones place.

For example, if the time stamp read 4:17 AM then four columns would be constructed.

1	2	3	4
…	…	…	…

We would then construct the first row, having it correspond to the number of minutes, in this case, seventeen.

	1	2	3	4
17	…	…	…	…

After which, we begin to add seven to form new rows until we have created enough cells to hold all twenty-six letters of the English alphabet. This pattern would remain the same for any table; for example the rows for 1:03 would increase by increments of three, 2:55 would increase by five, and 7:00 would increase by ten as would any other time stamp ending in zero.

	1	2	3	4
17	**…**	**…**	**…**	**…**
24				
31				
38				
45				
52				
59				

Each of these elements can now hold a number which we create by adding the value of its column and row. For example, the cell at the intersection of column three and row thirty-one would give thirty-four, as we see below.

	1	2	3	4
17	**18**	**19**	**20**	**21**
24	**25**	**26**	**27**	**28**
31	**32**	**33**	**34**	**35**
38	**39**	**40**	**41**	**42**
45	**46**	**47**	**48**	**49**
52	**53**	**54**	**55**	**56**
59	**60**	**61**	**62**	**63**

After this we assign a letter to each number, beginning at A, going left to right and down until we reach Z.

	1	2	3	4
17	**A**	**B**	**C**	**D**
24	**E**	**F**	**G**	**H**
31	**I**	**J**	**K**	**L**
38	**M**	**N**	**O**	**P**
45	**Q**	**R**	**S**	**T**
52	**U**	**V**	**W**	**X**
59	**Y**	**Z**		

However, this is only if the time stamp is ante meridiem. Instead if the time stamp is post meridiem then the last column begins at one and the last row begins with the minute digits. For example 4:17 PM would look like this:

	4	3	2	1
59	**63**	**62**	**61**	**60**
52	**56**	**55**	**54**	**53**
45	**49**	**48**	**47**	**46**
38	**42**	**41**	**40**	**39**
31	**35**	**34**	**33**	**32**
24	**28**	**27**	**26**	**25**
17	**21**	**20**	**19**	**18**

So if Paul Revere were to send Benjamin Franklin a text message at 4:17 AM reading "The Red Coats are coming!" this encryption method would change that to read

"49,28,25,47,25,21,20,41,18,49,48,18,47,25,20,41,39,32,40,27"

With the time stamp of 4:17 AM on his text message, Benjamin Franklin would be able to construct a matrix identical to the one above and therefore be able to decrypt Paul Revere's message by attaching a preassigned letter to each number. Naturally, the time it takes to construct, encrypt, and send a message would not allow for the above situation to happen simply through human ability. However, a mobile application with the encryption key would easily be able to send, encrypt, and decrypt a text message within a matter of seconds.

Due to the nature of the construction it is almost impossible to decrypt the messages without both the encryption method as well as the key. Furthermore, as there are no punctuation markings besides commas in between letters there would be little way to distinguish one word from another.

Unfortunately in some of the tables duplicate numbers arise as in the matrix we construct for 12:01 AM:

	1	2	3	4	5	6	7	8	9	10	11	12
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

	1	2	3	4	5	6	7	8	9	10	11	12
1	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**
2	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**
3	**Y**	**Z**										

It is clear that the above situation would engender difficulty decrypting a message as multiple letters map to one number, such as C, N, and Y all mapping to four. We solved this problem by including first one then two apostrophes as duplicates arise as is clear from the table below:

	1	2	3	4	5	6	7	8	9	10	11	12
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
2	**3'**	**4'**	**5'**	**6'**	**7'**	**8'**	**9'**	**10'**	**11'**	**12'**	**13'**	**14**
3	**4"**	**5"**	**6"**	**7"**	**8"**	**9"**	**10"**	**11"**	**12"**	**13"**	**14'**	**15**

So if the above text message from Paul Revere would have been sent at 12:01 AM instead it would read:

"10',9,6,8',6,5,4,5',2,10',9',2,8',6,4,5',3',10,4',8"

Thus, as you can see, we have developed a complex yet elegant method of substituting numbers for letters using merely the time stamp on your employees' text messages. Moreover, the largest possible—yet unlikely—output from one hundred forty plaintext characters is six hundred ninety-nine ciphertext characters, less than seven hundred characters and therefore within all of your requirements for the encryption method.

Finally, if we work by Kerckhoff's principle we may assume that your adversaries will know of this system, but not necessarily the encryption key. Therefore, there will be no mention of the time stamp as the necessary key; instead we will maintain that the key is a random series of three or four numbers necessary to decrypt the text message. Only the creators of the mobile application will need understand the significance of the time stamp to the encryption, anyone else will simply identify it as an ordinary time stamp.

We hope this encryption method meets with your approval and wish to continue to do business with your company in the future.

Sincerely Yours,

[redacted]

[redacted]

Math 485
Project 1

Group L

11 September 2015

OCRAI
OCRA Creative Recursive Acronyms, Inc.
485 Primality Way
Provo, UT 84604

In today's world, it is simple to send and receive messages and other data. Cell phones, computers with email capability, and tablet devices with an internet connection are ubiquitous. Perhaps because of the easiness of data transfer and the fact that it is now an everyday part of most people's lives, it seems that more and more often sensitive data is made available to people who were not intended to read it. Our client, OCRAI, has been subject to potential leaks of sensitive company data and trade secrets through erroneous text messaging by company employees. We will solve this problem by developing an encryption/decryption process that can be implemented on the phones of all employees of OCRAI, resulting in the secure encryption of all text messages sent and the proper decryption of all messages encrypted by this process.

We begin the encryption of a message with a discussion of which characters are allowed in a plaintext message. In the system we have chosen, the letters of the alphabet (no distinction between uppercase and lowercase), spaces, commas, periods, exclamation points, and question marks may all be encrypted. We will walk through how to use this system by applying the cipher on an example block of plaintext. The plaintext we will use is,

HOW ARE YOU TODAY, FRIEND?

The use of Comic Sans font throughout the rest of this report will indicate either plaintext or ciphertext.

Our first step to encrypt this message is to choose our key. We will use the 6-digit numeric key, (972658). Next we write the integers that correspond to each character in the plaintext in the line above the block to be encrypted.

The integers corresponding to each character are given by the following table.

PT/CT	Int	PT/CT	Int	PT/CT	Int	PT/CT	Int
A	0	I	8	Q	16	Y	24
B	1	J	9	R	17	Z	25
C	2	K	10	S	18	_	26

[redacted]

Math 485
Project 1

D	3	L	11	T	19	,	27
E	4	M	12	U	20	.	28
F	5	N	13	V	21	!	29
G	6	O	14	W	22	?	30
H	7	P	15	X	23	Not used	Not used

Thus we would write,

7 14 22 26 0 17 4 26 24 14 20 26 19 14 3 0 24 27 26 5 17 8 4 13 3 30
H O W _ A R E _ Y O U _ T O D A Y , _ F R I E N D ?

Let each number be p_n, where n is the index of characters starting with the first character of the message as 1.
We then process these numbers representing the characters of the message by using the following formula to obtain the encrypted number.

$c_n = 2(k_n) + 3(p_n)(k_{n+1}) \pmod{31}$

Where k_n is the nth digit of the key and $k_1 = k_7 = k_{13} = k_{19}$... etc. In general, $k_{n+6} = k_n$

Thus, $p_1 = 7$ which represents the letter H is changed to

$c_1 = 2*(9) + 3*(7)*(7) \pmod{31}$
$= 165 \pmod{31}$
$= 10 \pmod{31}$

p_2 is changed to

$c_2 = 2*(7) + 3*(14)*(2)$
$= 5 \pmod{31}$

Continuing to change each of the numbers, we end up encrypting our original message as

10-5-28-30-10-10-9-15-2-5-25-5-14-5-27-12-28-1-6-13-0-8-16-0-12-1-17

Writing the message as the characters represented by these numbers we have

kf.?kkjpcfzfof,m.bgnaiqambr

At this point, our encryption is complete. To decrypt this message we simply work in reverse or what we just did to encrypt the message. We would begin by rewriting the encrypted message in a string of integers from 0-30 as in the table above. As we would expect, this results in the string of numbers,

10-5-28-30-10-10-9-15-2-5-25-5-14-5-27-12-28-1-6-13-0-8-16-0-12-1-17

Knowing the formula to change a plaintext character into a ciphertext character is given by

$c_n = 2(k_n) + 3(p_n)(k_{n+1}) \pmod{31}$

We can rewrite this in terms of the plaintext integer, p_n

$$p_n = \frac{1}{3k_{n+1}}(c_n - 2k_n)\ (mod\ 31)$$

This gives us a method of changing the ciphertext integers into the plaintext integers, though it does require us to find the multiplicative inverse of $3k_{n+1}$ for each n. Since we have 31 characters and 31 is a prime number, the gcd($3k_{n+1}$,31) will always be 1, which makes this division possible.

Plugging our first ciphertext integer, $c_1 = 10$, into the given decryption formula, we obtain

$p_1 = (1/21)*(10 - 2*9) \pmod{31}$

To find the multiplicative inverse of 21, we use the extended Euclidean Algorithm[1].

31 = 1(21) + 10
21 = 2(10) + 1

Working backwards, we find

1 = 21 - 2(10)
1 = 21 - 2(31-21)
1 = 3(21) - 2(31) => 3(21) = 1 + 2(31)

Thus the multiplicative inverse of 21 (mod 31) is 3. Using this result, we obtain

$p_1 = 3*(10-18) \pmod{31} = -24 \pmod{31} = 7 \pmod{31}$

[1] See https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm to learn how to use this algorithm

So $p_1 = 7 = H$. For our second ciphertext integer ($c_2 = 5$) we have

$p_2 = (1/6)*(5 - 2*7) \pmod{31} = -5(-9) \pmod{31} = 45 \pmod{31} = 14 \pmod{31}$

So $p_2 = 14 = O$.

Continuing this pattern, we can decrypt the whole message and get back our original plaintext, HOW ARE YOU TODAY, FRIEND?

As we have shown, with our system it is a straightforward matter to encrypt text messages. The implementation of this system on each phone of the employees of OCRAI should be simple. Without the key, recipients of these messages will require significant computing power to break the encryption. Thus unintended recipients of messages containing sensitive company information- who will be without the decryption protocol and key- will not be able to read the messages. Please direct any questions, comments, or concerns to the IMC team found at 136 TMCB, Provo, UT 84602 or through email to [redacted] or [redacted]

Cryptography Project

[redacted]

There exists a problem in this world. If things of value are put in the open, people are likely to steal them. Therefore, we must take security measures to ensure that those things do not happen. Furthermore, in recent history, the problem of keeping valuable information secret has been getting harder and harder. Therefore, we have created a cryptographic system that, with limited processing power, encrypts a message of ASCII characters using a key of length nine, expanding the length of the original message by five.

Before encrypting, take the plain text message and convert it into its bits. Also, take each character of the key and convert it into the bits (but keep each character's bits separate). Reference the modified ASCII table in the appendix to do this. The first step in encryption uses the first five characters from the key. Take each character from the original message (suppose it is of the form "abcdefg", where each bit is a "1" or a "0"), and modify it like this with a copy of the keys:
key1': a*****g
key2': *b***f*
key3': **c****
key4': ***d***
key5': ****e**

Then put the modified keys back together in order (key1'key2'...key5'). Note that the asterisks are the original bits of the keys. The only thing that happens is that some of the bits are overwritten by the character from the plaintext. This lengthens the message by a factor of five.

The next step in encryption is to do a transposition. We use the next character from the key, which we transform into decimal according to the rules of binary (remember it should be a seven digit binary number). The character used here should not be divisible by seven, so that when the bits are turned back into characters, we will get a very different set of characters. This allows for over one hundred possible keys here.

For the sake of explaining the transposition, we will say that the amount is an arbitrary quantity "k." First, the end of the message is indicated with a character. Then the first "k" bits are moved to after that point in the message (they are deleted from the part before the special character and reproduced following it). The next "k" bits are left alone and a different character is used to indicate the end of the first "k" characters. Then the first "k" bits after the first special character are placed at the end of the message, deleting them again from the first part of the message. The same special character placed after the first "k" bits is placed after the next "k" bits.

This pattern repeats until the action would involve the character indicating the end of the message. If the action was going to leave the characters alone, nothing is done, and all the special characters are removed and the message is converted from bits to their ASCII designations (seven bits at a time, using the table in the appendix). If the action was going to move the characters, all characters before the end of message character are moved to the end, and all the special characters are removed and the bits are converted to their ASCII designations.

Once the table is used to change the bits into characters, we will do a separate transposition. We will use the next three characters of the key in decimal form. Since the length of the partially encrypted message is about six hundred, each character of the key should be less than one hundred. Therefore, the part of the key that matters in decryption has approximately 100*100*100*100 = 100^4 = 100,000,000 configurations. Suppose the decimal forms of each key are “a,” “b,” and “c.” Then we create an empty message, which we will call “ciphertext.” To transpose the message, we take the first “a” characters out of the original message, and put them at the end of “ciphertext.” Then we take the last “b” characters out of the original message, and put them at the end of “ciphertext.” Then we take the first “c” characters out of the original message, and put them at the end of “ciphertext.” We take the last “a” characters out of the original message, and put them at the end of “ciphertext.” We cycle through “a,” “b,” and “c” and alternate between taking characters from the start and the end. Once the number of remaining characters are less than the key currently in use, we put the rest at the end of “ciphertext.” We have now encrypted our message.

In decryption, each the order of the steps is reversed. We undo the second transposition first. Therefore, the first step in decryption uses the last three characters of the key, translated to decimal form. We will split the encrypted message into two smaller messages. We will distinguish them from each other by using the terms “front,” and “back.”

Suppose the three numbers from the key are “a,” “b,” and “c.” To undo this step, we take the first “a” characters from the ciphertext, and put them at the end of “front.” Then we take the next “b” characters and put them before the beginning of “back.” Then we take the next “c” characters from the ciphertext and put them at the end of “front.” Then we take the next “a” characters from the ciphertext and put them before the beginning of “back.” This pattern is continued until the length of the rest of the message is less than the current key being used. Then the partially decrypted message is acquired by putting the front, the rest of the message, and the back together in that order. Generally, we start with the first number from the key and take that many characters from the ciphertext, alternating between putting it at the end of “front” and the beginning of “back,” starting with “front.” Then the next key is used, and we cycle through the keys.

Remember that one way to undo any transposition is to transpose a made up message and see where each character went, and then to pull the transposed message back accordingly. It works like the matching section of a test. This is not the most efficient way to undo the transposition, but it definitely works and is easy to understand.

Now each character from the partially decrypted text is transformed into the bits according to the ASCII table. The next step in decryption is to undo the second step of encryption. We will explain how this works using some mathematical terms. Assuming the sixth character of the key has a decimal form “k,” we will say that x is the length of the partially decrypted text divided by 2*k, and if the remainder “r” is greater than k, then add r - k to x. Therefore, if r <= k, the length of the partially decrypted text is given by 2kx + r, and if r > k, it is 2k(x - r + k) + r. The number x is important because it tells us the “halfway point” of the message. This tells us the number of bits that were “left alone” in the second step of encryption. Once again, we will split the partially decrypted message into two messages. “Front” is the first x bits of the partially decrypted message, and “back” is the rest of the partially decrypted message. We also create a separate, empty message called “final.” Then to reconstruct the original series of bits, we take the first k bits from “back,” and put them in “final.” Then we take the first k bits from “front” and put them at the end of“final.” Then we take the next k bits from “back” and put them at the end of “final,” the next k bits from “front” and put them at the end of “final.” So we go back and forth, taking the first k bits from “back,” then “front,” (removing them) and then placing them at the end of “final.”

The final step in decrypting is to undo the first part of encryption. We will take thirty-five bits at a time from the partially decrypted text. The original seven bits are the first bit, the ninth bit, the seventeenth bit, the twenty-fifth bit, the thirty-third bit, the thirteenth bit, and the seventh bit of the thirty-five bits in that order. Once we have all of the original bits, we can get the original message by taking seven bits at a time and transforming them back into characters using the modified ASCII table.

We have described a cryptosystem that encrypts mostly using transpositions. The message is transformed to bits through a modified ASCII system, the message is stretched to a message five times as long, and then the bits are moved around, and transformed back into characters, which are then extremely different because the blocks of seven bits were split up and put in different places. Then the characters are transposed again. The key needed to decrypt has a hundred million different configurations, which strong for a ten digit key. Therefore, the encrypted information will remain secure and, even if others knew the complex cryptosystem, it would be incredibly difficult to decrypt.

EXAMPLES, ENCRYPTION PART ONE:
We will use 1010011101010010000011011001 and a key of “stars” which we transform into:
s = 1110011
t = 1110100
a = 1100001
r = 1110010
s = 1110011
For future reference, the first seven digits of the plain text bits are 1010011, the next seven are 1010100, the third group of seven is 1000001, and the last seven are 1011001.
We work one byte at a time. For a byte “abcdefg,” we encode using the key “stars” as follows:
s* = a11001g
t* = 1b101f0
a* = 11c0001
r* = 111d010
s* = 1110e11
And then our resulting byte is encrypted by s* t* a* r* s*
In this case, we obtain
s* = 1110011
t* = 1010110
a* = 1110001
r* = 1110010
s* = 1110011
And we get 11100111010110111000111100101110011
We repeat for the bytes 1010100, 1000001, and 1011001.
s* = 1110010
t* = 1010100
a* = 1110001
r* = 1110010
s* = 1110111
11100101010100111000111100101110111

s* = 1110011
t* = 1010100
a* = 1100001
r* = 1110010
s* = 1110011
11100111010100110000111100101110011

s* = 1110011
t* = 1010100
a* = 1110001
r* = 1111010
s* = 1110011

11100111010100111000111110101110011

So our partially encrypted message that originally was 1010011101010010000011011001 becomes
11100111010110111000111100101110011111001010101001110001111001011101111110011101010011000
0111100101110011111001110101001110001111101011100 11

EXAMPLES, DECRYPTION TO FIRST PART:
In order to decrypt, we split the series of bits into smaller sets of 35 bits.
11100111010110111000111100101110011
11100101010100111000111100101110111
11100111010100110000111100101110011
11100111010100111000111110101110011

We will work one set at a time. We split the first set into five sets of seven bits.
1110011
1010110
1110001
1110010
1110011
Remembering the positions of the original bits based on the general map:
a*****g
*b***f*
c**
d
****e**
The original bits were 1010011.
We repeat for the other sets of bits.

11100101010100111000111100101110111 becomes
1110010
1010100
1110001
1110010
1110111
And the original bits were 1010100.

11100111010100110000111100101110011 becomes
1110011
1010100
1100001
1110010
1110011
And the original bits were 1000001.

11100111010100111000111110101110011 becomes
1110011
1010100
1110001
1111010
1110011
And the original bits were 1011001.
Therefore, the original message was:
1010011 1010100 1000001 1011001
Which we can compare this with the original:
1010011 1010100 1000001 1011001
We have successfully encrypted and decrypted this message.

EXAMPLES, ENCRYPTION PART TWO:
Partially encrypted bits: 1101000101010111010101110100101111101010101010010
First we are going to put a marker at the end, we will use for the sake of being able to see it the character *.
Let's say the corresponding part of our key says "5," so we will select the first five bits and put them after the asterisk, deleting them from the left side of the asterisk.
00101010111010101110100101111101010101010010*11010
Notice that the "11010" has been moved from the front of the message to the part after the asterisk in its original order.
We will leave the next five where they are. We can put an ampersand after them to indicate that they are fixed. Then we select the five after them, which is "01011," and put them after "11010."
00101&1010101110100101111101010101010010*1101001011
Place an ampersand five characters after the first ampersand, select the five letters after that, and place them at the end.
00101&10101&10010111110101010101010010*110100101101110
Just keep going.
00101&10101&10010&01010101010010*11010010110111011111
00101&10101&10010&01010&0010*1101001011011101111110101
Since the length of the rest of the characters before the asterisk is four, we're done. If we were to reach the asterisk while cut/pasting bits, we would cut/paste as many of the bits as we could - even if it was just the last two - three.
Since we're done, we remove the ampersands and the asterisk.
0010110101100100101000101101001011011101111110101

We will do another example.
Partially encrypted bits: 0010101110101011010001010110110101010111011101001011101O
First we put the asterisk at the end.
001010111010101101000101011011010101011101110100101110 10*
Let's say this time the key indicates each block should have a length of 3.

Each step of the process will be completed here but without much explanation.
010&11101010110100010101101101010101110111010010111010*001
010&010&1011010001010110110101010111011101001011101O*001111
010&010&101&0001010110110101010111011101001011101O*001111101
010&010&101&101&0110110101010111011101001011101O*001111101000
010&010&101&101&011&0101010111011101001011101O*001111101000011
010&010&101&101&011&101&0111011101001011101O*001111101000011010
010&010&101&101&011&101&101&1101001011101O*001111101000011010011
010&010&101&101&011&101&101&100&1011101O*001111101000011010011110
010&010&101&101&011&101&101&100&110&10*001111101000011010011110101
010&010&101&101&011&101&101&100&110&*001111101000011010011110101 10
0100101011010111011011001100011111010000110100111 1010110

This step disrupts the bits in case of a frequency of resulting characters.

EXAMPLES, DECRYPTION TO SECOND PART:
We will demonstrate decryption by reproducing
“0010101110101011010001010110110101010111011101001011101O” from
“010010101101011101101100110001111101000011010011110101 10” and knowing that the key is 3.
The first thing that we do is calculate how long the first part and the second part must have been (indicated by the two sides of the asterisk).
010&010&*101*&101&*011*&101&*101*&100&*110*&001&*111*&101&*000*&011&*010*&011&*110*&101&*10*
The sets of bits between asterisks would be moved to the end, and the sets of bits between ampersands would remain where they are. Therefore, for a set of bits this length, there should be nine sets of bits that stay on the left half of the bits, since there are nine sets of bits between ampersands. This is equivalent to the division mentioned in the paper. This is a more visual representation.
The first nine sets of bits are *010*&010&*101*&101&*011*&101&*101*&100&*110*, so we can remove all the special characters in this set of bits.
010010101101011101101100110
And notice that the remainder of the message is
&001&*111*&101&*000*&011&*010*&011&*110*&101&*10*, However, we will leave the special characters here. We will place the first half of the message in front.
010010101101011101101100110&001&*111*&101&*000*&011&*010*&011&*110*&101&*10*
Notice that the first set of bits surrounded by special characters is &001&. We will place this at the front of the message, delete the special characters, and place the marker (*) three bits after the end of 001.
001010(*)010101101011101101100110*111*&101&*000*&011&*010*&011&*110*&101&*10*
Therefore the (*) is placed after the first six bits. The next set enclosed by asterisks or ampersands is *111*. So we will place *111* after the (*) and remove the asterisks, and place another (*) three bits after the end of *111*.
001010(*)111010(*)101101011101101100110&101&*000*&011&*010*&011&*110*&101&*10*
We proceed according to this pattern.
001010(*)111010(*)101101(*)1010111011011001 10*000*&011&*010*&011&*110*&101&*10*

001010(*)111010(*)101101(*)000101(*)011101101100110&011&*010*&011&*110*&101&*10*
001010(*)111010(*)101101(*)000101(*)011011(*)101101100110*010*&011&*110*&101&*10*
001010(*)111010(*)101101(*)000101(*)011011(*)010101(*)101100110&011&*110*&101&*10*
001010(*)111010(*)101101(*)000101(*)011011(*)010101(*)011101(*)100110*110*&101&*10*
001010(*)111010(*)101101(*)000101(*)011011(*)010101(*)011101(*)110100(*)110&101&*10*
001010(*)111010(*)101101(*)000101(*)011011(*)010101(*)011101(*)110100(*)101110(*)*10*
Now, since the last set of characters were changed (and therefore originally at the end of the message), we can just erase all special characters to get:
00101011101010110100010101101101010101110111010010111010
Note that this is equivalent to
00101011101010110100010101101101010101110111010010111010
Which I copy/pasted from the original, while the one above it I simply deleted the special characters.
Therefore, we have achieved the original set of bits from the encrypted set and the key.
We will do it again for a slightly different case; we will use the first set of bits we encrypted this time.
The original set of bits is 11010001010101110101011101001011111010101010010, the key is five, and the encrypted set of bits is 00101101011001001010001011010010110111011111110101.
00101&10101&*10010*&01010&*00101*&10100&*10110*&11101&*11111*&0101&
This time we have run into an issue. Since the last four bits are between ampersands, and four is not five, we will have to re-do some of the symbols to adjust. However, we first count that there are four complete sets between ampersands, so the first four sets of bits are fine and we can delete the special characters.
00101101011001001010
Since there would be four more bits, we take the next four bits from the next set and place them at the end of this re-transposition.
001011010110010010100010
Now we take all the rest of the bits from the original encrypted message and delete the special characters.
110100101101110111110101
We will take the first five of these bits and put them on the front of the re-transposition. We mark the spot five bits after the end of this addition with (*).
1101000101(*)1010110010010100010
The rest of the characters to add are 0101101110111110101. Take the next five, place them after the last (*), and put another (*) five bits after the end of the recently inputted five bits.
1101000101(*)0101110101(*)10010010100010
Continue with the pattern.
011101111110101
1101000101(*)0101110101(*)0111010010(*)010100010

1111110101
1101000101(*)0101110101(*)0111010010(*)1111101010(*)0010

10101
1101000101(*)0101110101(*)0111010010(*)1111101010(*)101010010

Now remove all of the (*) and we are done.
1101000101010111010101110100101111101010101010010
Notice that it is equivalent to
1101000101010111010101110100101111101010101010010
Which we said it would be.

EXAMPLES, ENCRYPTION PART THREE
We will use three numbers from the key. So if the corresponding section of the key was 3, 4, 2, then it would work like this:
Plain text: THISISAMESSAGE
Our first step is to take the first three characters at the beginning of the plain text and to put them as they are into our cipher text.
Plain text: SISAMESSAGE
Cipher text: THI
Our next step is to take the last four characters at the end of the plain text and to put them as they are and attach them to the end of the cipher text.
Plain text: SISAMES
Cipher text: THISAGE
Then we take the first two characters at the beginning of the plain text and attach them to the end of the cipher text.
Plain text: SAMES
Cipher text: THISAGESI
Then we take the last three characters at the end of the plain text and attach them to the end of the cipher text.
Plain text: SA
Cipher text: THISAGESIMES
Since the next step would be to take the first four characters at the beginning of the plain text, but there are less than four characters left, we will take all of them and put them at the end.
Plain text:
Cipher text: THISAGESIMESSA
We have encrypted the message.
Some details to note: Each of the numbers in the key refers to how many characters are taken at a time. First it is three, then four, then two, then three, then four, two, and so on, repeating in this fashion.
The first time characters are taken from the plain text, they are taken from the beginning, and then they are taken from the end, and then they are taken from the beginning.
While this does not effectively scramble a message that much, because this is the last step, the message already does not make very much sense and is a meaningless jumble of characters. We will do a second example that demonstrates this.

Suppose we have the following:
Partially encrypted text: aSjfI(43$0- \?/BnT,wQPty091Vcbr2[:'os3UhjblK5.!eI
Key: 592

This means that we will take five characters, nine characters, two characters, and repeat.
Partially encrypted text: aSjfI(43$0- \?/BnT,wQPty091Vcbr2[:'os3UhjblK5.!eI

Partially encrypted text: (43$0- \?/BnT,wQPty091Vcbr2[:'os3UhjblK5.!eI
Cipher text: aSjfI
Partially encrypted text: (43$0- \?/BnT,wQPty091Vcbr2[:'os3Uh
Cipher text: aSjfIjblK5.!eI
Partially encrypted text: 3$0- \?/BnT,wQPty091Vcbr2[:'os3Uh
Cipher text: aSjfIjblK5.!eI(4
Partially encrypted text: 3$0- \?/BnT,wQPty091Vcbr2[:'
Cipher text: aSjfIjblK5.!eI(4os3Uh
Partially encrypted text: nT,wQPty091Vcbr2[:'
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B
Partially encrypted text: nT,wQPty091Vcbr2[
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'
Partially encrypted text: Pty091Vcbr2[
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'nT,wQ
Partially encrypted text: Pty
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'nT,wQ091Vcbr2[
Partially encrypted text: y
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'nT,wQ091Vcbr2[Pt
Partially encrypted text:
Cipher text: aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'nT,wQ091Vcbr2[Pty

EXAMPLES, DECRYPTION TO THIRD PART:
Now we will decrypt these messages. In the plain text "THISISAMESSAGE" there are 14 characters. We will create a dummy text that is 14 characters long as follows:
"*1* *2* *3* *4* *5* *6* *7* *8* *9* *10* *11* *12* *13* *14*"
Notice that if we encrypt this with the key, it will tell us where each number goes. Therefore, to decrypt, we take the letter at each position and put it in the order of its corresponding number.
This can also be used to encrypt.

Dummy text: *1* *2* *3* *4* *5* *6* *7* *8* *9* *10* *11* *12* *13* *14*
Encrypted dummy text:

Dummy text: *4* *5* *6* *7* *8* *9* *10* *11* *12* *13* *14*
Encrypted dummy text: *1* *2* *3*

Dummy text: *4* *5* *6* *7* *8* *9* *10*
Encrypted dummy text: *1* *2* *3* *11* *12* *13* *14*

Dummy text: *6* *7* *8* *9* *10*
Encrypted dummy text: *1* *2* *3* *11* *12* *13* *14* *4* *5*

Dummy text: *6* *7*
Encrypted dummy text: *1* *2* *3* *11* *12* *13* *14* *4* *5* *8* *9* *10*

Dummy text:
Encrypted dummy text: *1* *2* *3* *11* *12* *13* *14* *4* *5* *8* *9* *10* *6* *7*

Now we need the encrypted text. We will line it up with our encrypted dummy text.
T H I S A G E S I M E S S A
01 02 03 11 12 13 14 04 05 08 09 10 06 07
Therefore, in re-organizing our message, we do the following:
T H I S I S A G E M E S S A
01 02 03 04 05 11 12 13 14 08 09 10 06 07

T H I S I S A S A G E M E S
01 02 03 04 05 06 07 11 12 13 14 08 09 10

T H I S I S A M E S S A G E
01 02 03 04 05 06 07 08 09 10 11 12 13 14

And we have reproduced our original plain text, “THISISAMESSAGE.”
Additionally, we will demonstrate a different system to do this on the other encrypted message:
aSjfIjblK5.!eI(4os3Uh3$0- \?/B:'nT,wQ091Vcbr2[Pty
Notice that the first five characters must still be the same, but the next nine characters came from the end. We will split these two sets of characters into their own sequences.
(4os3Uh3$0- \?/B:'nT,wQ091Vcbr2[Pty
aSjfI
jblK5.!eI
The next two should follow the first five, and the five after that should precede the last nine.
3$0- \?/B:'nT,wQ091Vcbr2[Pty
aSjfI(4
os3UhjblK5.!eI
The next nine should follow the first seven characters, and the two after that should precede the last fourteen.
nT,wQ091Vcbr2[Pty
aSjfI(43$0- \?/B
:'os3UhjblK5.!eI
The next five should follow our first split set of characters, and the nine after that precede the second.
Pty
aSjfI(43$0- \?/BnT,wQ
091Vcbr2[:'os3UhjblK5.!eI
The next two should follow the first set, and the remaining character (since it is less than the amount we are taking) precedes the second set.

aSjfI(43$0- \?/BnT,wQPt
y091Vcbr2[:'os3UhjblK5.!eI
Now we place the second half after the first half:
aSjfI(43$0- \?/BnT,wQPty091Vcbr2[:'os3UhjblK5.!eI
And we can compare with the original character sequence:
aSjfI(43$0- \?/BnT,wQPty091Vcbr2[:'os3UhjblK5.!eI
The result is the same! Therefore, we have encrypted and successfully decrypted.

Sample problem
Encrypt the message “Is this on?” (without the quotes) with the key abcdeäåéü.
You should get “Ö{ÉôJ[ëgÆl@rfmH¡Å>[É£¢[0H¡é'l@Ä/n9köÉ!¥{ÉôlaJpn|@üg{V¡,”

Appendix
Modified ASCII Table

¡ 0000000	É 0010000	(space) 0100000	0 0110000	@ 1000000	P 1010000	` 1100000	p 1110000
ü 0000001	æ 0010001	! 0100001	1 0110001	A 1000001	Q 1010001	a 1100001	q 1110001
é 0000010	Æ 0010010	“ 0100010	2 0110010	B 1000010	R 1010010	b 1100010	r 1110010
â 0000011	ô 0010011	# 0100011	3 0110011	C 1000011	S 1010011	c 1100011	s 1110011
ä 0000100	ö 0010100	$ 0100100	4 0110100	D 1000100	T 1010100	d 1100100	t 1110100
à 0000101	ò 0010101	% 0100101	5 0110101	E 1000101	U 1010101	e 1100101	u 1110101
å 0000110	û 0010110	& 0100110	6 0110110	F 1000110	V 1010110	f 1100110	v 1110110
ç 0000111	ù 0010111	‘ 0100111	7 0110111	G 1000111	W 1010111	g 1100111	w 1110111
ê 0001000	ÿ 0011000	(0101000	8 0111000	H 1001000	X 1011000	h 1101000	x 1111000
ë 0001001	Ö 0011001	) 0101001	9 0111001	I 1001001	Y 1011001	i 1101001	y 1111001
è 0001010	Ü 0011010	* 0101010	: 0111010	J 1001010	Z 1011010	j 1101010	z 1111010
ï 0001011	¢ 0011011	+ 0101011	; 0111011	K 1001011	[1011011	k 1101011	{ 1111011
î 0001100	£ 0011100	, 0101100	< 0111100	L 1001100	\ 1011100	l 1101100	| 1111100
ì 0001101	¥ 0011101	- 0101101	= 0111101	M 1001101	] 1011101	m 1101101	} 1111101
Ä 0001110	á 0011110	. 0101110	> 0111110	N 1001110	^ 1011110	n 1101110	~ 1111110
Å 0001111	í 0011111	/ 0101111	? 0111111	O 1001111	_ 1011111	o 1101111	ñ 1111111