Group L

Dear OCRAI,

In response to the security breaches your company has faced, our team proposes the following three-step encryption process to enable your company to more securely send messages to one another. Our cryptosystem uses a randomly chosen 10-digit number to encode plaintext English into a ciphertext (you could also choose this key yourself, of course). The same key is then used to decode the ciphertext. We will provide examples along the way to illustrate how the cryptosystem might be used.

For our cryptosystem, we use the following alphabet:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, " " (space), "." (period), "," (comma), " ' " (apostrophe), " " "(quote), "?" (question mark), "!" (exclamation point)

We will use these three steps to encrypt the plaintext message:

1) Apply an affine cipher
2) Reorder the letters
3) Add in extra characters

Suppose we had the key 0413277130 and we wanted to encrypt the plaintext:

*this is an example of how our cryptosystem works*

We begin by performing an affine cipher. The last 4 digits of the key, 7130, tell us how we should use the affine cipher. The first two digits, 71, correspond to $\alpha$ and the next two digits, 30, correspond to $\beta$. If $\alpha$ is 0, 43 or 86, the affine cipher will send all characters to the same character. Use 1 for $\alpha$ in these cases.We work with modulo 43 because there are 43 characters in our alphabet. Then, we apply the equation

$$x \rightarrow \alpha x + \beta \ (mod\ 43)$$

where $x$ is the numerical representation of the character to be encoded. For example, the character “t” is represented by 29 because it is the 30th letter of our alphabet, with the first position represented by 0. Then,

$$29 \rightarrow 71(29) + 30 \ (mod\ 43) \equiv 2089 \ (mod\ 43) \equiv 25 \ (mod\ 43)$$

The character corresponding to this number is “P”, so we would replace all instances of “t” in our plaintext with “P”. When we encode all of the letters of the example plaintext, we obtain

PXI"6I"69T6Z891!GZ6EK6XEN6EAC6MC !PE" "PZ16NECV"

We will refer to this text as ciphertext1.

Next, we will scramble the letters in ciphertext1. To do this, we look at the first two digits of the key, which are 04 in our case. We will call this our “jump number.” We will first number each character in the ciphertext1, starting at 0, as follows:

| P | X | I | " | 6 | I | " | 6 | 9 | T | 6 | Z | 8 | 9 | 1 | ! | G | ... | V | " |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 47 | 48 |

As we create the scrambled version of ciphertext1, we will begin by taking all of the characters in ciphertext1 that correspond to a number congruent to 0 (*mod* <jump number>), starting with the smallest and working to the largest. For our example, we would start with all of the numbers congruent to 0 (*mod* 4), starting with the character corresponding to 0 and ending with the character corresponding to 48. Here is what we have for our ciphertext2 so far:

P698GKNC "ZE

Next, we will put all of the characters congruent to 1 (*mod* <the jump number>), then keep going in that order until the last set of characters will be all of the characters congruent to (<the jump number> - 1) *mod (*<the jump number>). So, in our example, the last block of text will be all of the characters with numbers that are congruent to 3 (*mod* 4). Here is what ciphertext2 will look like when we are done reordering all the characters:

P698GKNC "ZEXIT9Z666! 1CI"616XEMP"6V"6Z!EEACEPN"

You might notice that if the jump number is longer than the length of the message, the letters will not be scrambled. For this reason, we require that the jump number be less than (or equal to) the length of the message by using the <jump number> (*mod* <message length>). We also require that the <jump number> (*mod* <message length>) not be equal to 0 since (*mod* 0) is undefined. So if this happens, we use 37 instead.

In other words, if we have a jump number of 6, then we would write every 6th character as our new ciphertext, starting with the first character in the message. When we get to the end of the message, we re-start with the second character in the message and write every 6th character from there on, and so on until we have included every character in the original message. We don’t want to jump over the entire message, so if the jump number in the key is longer than the message, divide the given jump number by how many characters are in the message and take the remainder to be the jump number instead. And jumping by 0 characters doesn’t make any sense

(because jumping by 1 is already the same as the original order), so if the jump number in the key happened to be 0, we would use 37 instead.

The final part of the encryption process is to insert random characters, thereby creating noise for any attacker hoping to decode the message. We will use the third through sixth digits in the key to determine how many random characters to insert before each character in ciphertext2. In order to ensure that the final ciphertext length will not exceed 800 characters, we can add at most 4 characters before each character in ciphertext 2. So, it is possible that the number from the random number generator (assuming you are using one) will need to be adjusted to account for this by taking the number (*mod* 5). For our example, these digits are "1327". Since 7 is not between 0 and 4, we would take the remainder from dividing 7 by 5 and use that number instead (this is the same as saying 7 (*mod* 5)). So in our case, our new number would be 2.

The first number in this section (the third number in the key) is the number of random characters to insert before the first character of our ciphertext. So in our example, we would begin by adding 1 random character in front of the first character in ciphertext2. The second number is the number of random characters to insert before the second character. The third number is the number of random characters to insert before the third character. The fourth number is the number of random characters to insert before the fourth character. Then we would start over at the first number to determine how many characters go before the fifth character. And so forth. Here is an example of what ciphertext3 might look like:

RPO586BZ9B08NG.P"KVYN.WC0 0D "H,Z!TEBXJB2I3?T139CZ
2OP6GR6DO67!SJ6 ZR1VECDIMOB"4965S1N6RW3XF5EIJMHP
AAN"5R6EBV'"TEJ6X0ZXQ!1EEH!EQ4AZIC7EUCCPQBNN!"

This would be the ciphertext that would get sent by the sender.

To decrypt the message, we need to do the following:

1) Remove the random letters
2) Unscramble the encrypted letters
3) Apply the reverse affine cipher

First, we need to find and remove the extra letters inserted by the sender. Looking at the third through sixth numbers of the key, we can determine how many letters were inserted before each character. Again, if there is a number that is not between 0 and 4 (inclusive), we will need to divide by 5 and take the remainder. In our example, these numbers are "1327." So, we will begin by removing 1 character from ciphertext3. The next character is part of ciphertext2. Next, we remove 3 characters. The next character is part of ciphertext2. Next, we remove 2 characters.

The next character is part of ciphertext2. Next we remove 2 characters (Since 7 divided by 5 has a remainder of 2). The next character is part of ciphertext2. We continue by repeating this pattern to remove all excess characters.

Next, we need to unscramble the letters. We will begin with a grid, where the number of columns is equal to the “modified jump number”, which is <the jump number given in the key> (*mod* <the length of the message>), or 37 if that number is 0. The number of rows is equal to the length of ciphertext2 divided by the number of columns, rounded up to the nearest whole number. In our example, since the modified jump number is 4 and the length of ciphertext2 is 48, we would have 4 columns and 12 rows. We write the letters of ciphertext2 starting in the upper right hand corner, working down the column, then starting again at the top of the next column and so forth until we run out of letters. Using ciphertext2, which is

P698GKNC "ZEXIT9Z666! 1CI"616XEMP"6V"6Z!EEACEPN"

We would have the following:

| | | | |
|---|---|---|---|
| P | X | I | " |
| 6 | I | " | 6 |
| 9 | T | 6 | Z |
| 8 | 9 | 1 | ! |
| G | Z | 6 | E |
| K | 6 | X | E |
| N | 6 | E | A |
| C | 6 | M | C |
| | ! | P | E |
| " | | " | P |
| Z | 1 | 6 | N |
| E | C | V | " |

Reading from the top right corner across the rows, starting at the next row after the end of each previous row, we have our ciphertext1, which is

PXI"6I"69T6Z891!GZ6EK6XEN6EAC6MC !PE" "PZ16NECV"

If the number of columns does not evenly divide the length of the ciphertext2, the last row will only have a number of cells equal to the remainder when dividing the length of ciphertext2 by the column number. To illustrate how this would work, suppose we had a ciphertext2 that was "TIEGHSSEIASSMA", which has a length of 14 and our jump number was 4. We would have 4 columns (our jump number) and 4 rows (since 14/4 = 3.5, which rounds up to 4). Since 14/4 has a remainder of 2, we will only have two cells in the last row. We would still write the letters, starting in the top left corner, working down the column, then go to the next column, as illustrated below:

| | | | |
|---|---|---|---|
| T | H | I | S |
| I | S | A | M |
| E | S | S | A |
| G | E | | |

Reading across the rows, the decoded text would be "THISISAMESSAGE".

Finally, we will apply the reverse affine cipher. We will use the equation

$$x \to \alpha^{-1} \cdot (x - \beta) \, (mod \, 43)$$

Where $\alpha^{-1}$ represents the number such that $\alpha \cdot \alpha^{-1} \equiv 1 \pmod{43}$. In our example, $\alpha$ is 71, and since $71 \cdot 20 \equiv 1 \pmod{43}$, 20 is our $\alpha^{-1}$. Our $\beta$ is 30. Then we apply the equation

$$x \to 20 \cdot (x - 30) \, (mod \, 43)$$

to each character in ciphertext1, which will yield our plaintext message.

(Note: While we chose $\alpha$ in such a way so that $\alpha^{-1}$ will exist, the process of actually finding this number is nontrivial. While we do not go into detail here on how to find $\alpha^{-1}$ by hand, a guess and check method will work, as $\alpha^{-1}$ will always be between 1 and 42, inclusive.)

While this can all be done by hand, we understand that as busy employees and administrators, it may be too time-consuming to encrypt and decrypt all your messages personally, so we are happy to provide you with a computer program that will perform the needed changes for you and

that as an added bonus, generates random characters to add in as well as random encryption keys. This will certainly save you the effort of choosing your own.

We hope that our cryptosystem will be sufficient for your needs.

Yours sincerely,

[redacted]
Independent Math Contractors