Without a decryption and encryption method for text messages between employees of OCRAI, it is easy for messages containing sensitive information to be received by competitors or by the media. As requested, we have developed a way to encrypt messages using a shared secret key. Our solution will be easy for your engineers to use in an application that will allow all employees to encrypt and decrypt their messages. This will eliminate the problem of sensitive information being shared with the wrong people. We will now explain the algorithm we have developed.

Algorithm:

First, the message is converted to ASCII format. This is done by replacing each character in the message with its corresponding binary ASCII value which will be 8 bits. The table that outlines what each character is converted to can be seen at [http://www.roubaixinteractive.com/PlayGround/Binary_Conversion/The_Characters.asp](http://www.roubaixinteractive.com/PlayGround/Binary_Conversion/The_Characters.asp). The same conversion process is applied to the key. For example, the plaintext “hello” would be 01101000011001011011000110110001101111. The key “key” would be converted to 011010110110010101111001. Because the plaintext may be longer than the key, we need to extend the key so that its length is equal to the length of the plaintext message. To do this, the key is repeated as many times as necessary to make it as long as or longer than the message, and then cut off at the point where the key’s length is equal to the length of the message.

So hello is: \[01101000011001011011000110110001101111\]
and the key is: \[011010110110010101111001\]

The XOR function is then applied. The XOR function essentially applies the following formula to each digit individually:

\[c_i = (m_i + k_i)(mod\ 2)\]

where \(m_i\) is the i-th digit in the message, \(k_i\) is the i-th digit of the key and \(c_i\) is the i-th digit of the resulting ciphertext.

Applying this formula to our example results in the following:

\[0000001100000000000101010000011100001010\]

To decrypt the message, the key is again converted to binary and expanded to the length of the ciphertext the same way we expanded it for the encryption. The XOR function is then applied to the ciphertext and the key. This will result in the following formula to be applied for each digit of the ciphertext:

\[m_i = (c_i + k_i)(mod\ 2)\]
where \( m_i \) is the \( i \)-th digit in the message, \( k_i \) is the \( i \)-th digit of the key and \( c_i \) is the \( i \)-th digit of the resulting ciphertext. In our example, this would return \( 011010000110010110110001101100011011111 \). This is finally returned to its original content by converting from this binary form back to the corresponding ASCII characters.

This entire process can be done by hand, but attached is Python code that will allow you to do this on your computer. You can copy and paste it into your Python terminal or copy it into a Sage worksheet (which uses Python). The code that will be used to help encrypt and decrypt the message will have a few functions\(^1\). The first function will be used to change the text into binary digits. To use this function, declare a variable and using the toBin function, insert your text in quotes as a string. For example, to convert the message “hello world” to its binary form, you would write the following code:

```python
message = toBin("hello world")
```

Use the same function to declare a key variable change the key into binary. An example of this using the key “secret” is done with the following code:

```python
key = toBin("secret")
```

To create the ciphertext, use the XOR function with the first parameter as the message and the second parameter as the key.\(^2\) This is done using the following code:

```python
cipher = XOR(message, key)
```

The resulting ciphertext is then displayed by using the print command:

```python
print cipher
```

Finally, to have the message decrypted you would send the key and the cipher message to the recipient. Then the recipient will change the key into binary then use the XOR function to get the binary representation of the plaintext. Then the recipient will use the toText function to change the binary representation of the plaintext into characters.

```python
decrypt = XOR(cipher, key)
decrypt_text = toText(decrypt)
```

Then by using the print function the recipient will be able to see the plaintext:

```python
print "Decrypted message: " + decrypt_text
```

This process will easily allow employees to encrypt and decrypt messages on their phones, and will prevent any other person from making sense of the message without the key. It is important, however, that the key is not reused, as anyone will be able to determine the key with both the plaintext and the ciphertext, thereby making the continued use of the key a security risk. An added advantage of this particular algorithm is that it will enable the use of all punctuation and characters that are often used in text messages. The adaption of this method by your engineers will enable your employees to
communicate securely with each other.
def toBin(text):
    result = ""
    for c in text:
        binary = str(bin(ord(c)))[2:]
        cut = binary
        while len(cut) < 8:
            cut = '0' + cut
        result += cut
    return result

def toText(binary):
    result = ""
    for i in range(0, len(binary) // 8):
        string = binary[i*8:i*8+8]
        num = int(string, 2)
        char = chr(num)
        result += char
    return result

def XOR(plain, key):
    result = ""
    key_length = len(key)
    counter = 0
    for c in plain:
        keyChar = key[counter % key_length]
        if c == keyChar:
            result += '0'
        else:
result += '1'
counter += 1
return result

#EXAMPLE
message = toBin("test")
key = toBin("blah")
cipher = XOR(message, key)
print "Cipher in Binary: " + ci
decryp = XOR(cipher, key)
print "Decrypted Binary: " + decryp
print "Decrypted message: " + toText(decryp)

1. It should be noted that the above functions are designed to give you a visual view of the binary values by converting the bits into strings of 0s and 1s. If you want a more direct encryption where the binary is saved as bits, use the following to both encrypt and decrypt (replace “^” with “^^” if using Sage):

```python
def simplePad(text, key):
    result = ""
    key_length = len(key)
    counter = 0
    for c in text:
        numText = ord(c)
        numKey = ord(key[counter%key_length])
        result += chr(numText ^ numKey)
    return result
```

2. The supplied XOR function expands (repeats) the key automatically as it XORs the two values.