Lord Marcel,

Nearly 40 years ago, Knights Ron Rivest, Adi Shamir, and Leonard Adleman created a tool for exchanging information between kingdoms. I must use this tool to securely send information from the land of Adobe as there are many eyes watching our movements.

Your secret code was recently sent via text. The public key is (n = 597782322723352841, e = 65537).

Use your code to decipher my decimal messages by converting to hexadecimal and then to text. As an example, 454340527577251408658717599750907599391754895552930391484449 is an unencrypted base 10 birthday message.

Please await my next letter for instructions.

God speed,
Earl of Tillay

P.S. 16172526842824332 486461013351423796 372855876819966270 173013308343788726 175628482683316976 51457655426288031 112532939420200262 103102526154905960 212110150840885429 116036075239775253

****
Creation of the RSA public-key cryptosystem.

Choose 2 Primes:
p = 608461363
q = 982449107
(9-bit encryption)

n = p*q = 597782322723352841

Euler's totient function, φ(x), counts integers up to x that are relatively prime to x. We must choose an encryption key with no factors in common with x = (p-1)(q-1) = 597782321132442372

The factors of x = 597782321132442372 are $2^2 \times 3^3 \times 89 \times 379 \times 1973 \times 5711 \times 14563$

Therefore, we may choose prime encryption key, $e = 2^{16} + 1 = 65537$.

Calculating the modular multiplicative inverse using the extended Euclidean algorithm:
$d = e^{-1} \pmod{x} = 162568843699643261$

n = public key = 597782322723352841
e = 65537
d = Josh's private key = 162568843699643261

For instance, let's encrypt the message "Hi Josh!". In ASCII hex this is 4869204a6f736821. Each digit is 2^4 = 16 bits.

n is an 18 digit number where $2^{59} < n < 2^{60}$ so we can have at most 59 bits of information in each block of text we send. Since we have chunks of hex data, we will transmit the data using blocks of 56 bits or 14 hexidecimal characters. In the previous message we have 8 ASCII characters: 48 69 20 4a 6f 73 68 21 representing 16 hexidecimal values. Therefore, I will pad the end of the message with spaces to make it a multiple of 14: "Hi Josh!   " = '4869204a6f7368212020202020202020'. I have 2 blocks to encode:

m_1 = 4869204a6f73682 = 20381785731855208 in decimal.
m_2 = 21202020202020 = 9323996581470240 in decimal.
$c_1 = m_1^e \pmod{n} = 20381785731855208^{65537} \pmod{597782322723352841} = 452705572751101392$
$c_2 = m_2^e \pmod{n} = 9323996581470240^{65537} \pmod{597782322723352841} = 158811280628252673$

These encrypted blocks of information will be sent to Josh. He must convert these back to the original message with his private key:

Step 1: calculate the decimal plaintext.

$m_1 = c_1^d \pmod{n} = 452705572751101392^{162568843699643261} \pmod{597782322723352841} =$
20381785731855208

$m_2 = c_2^d \pmod{n} = 158811280628252673^{162568843699643261} \pmod{597782322723352841} =$
9323996581470240

This is fairly easy to do using Wolfram Alpha, e.g., https://www.wolframalpha.com/input/?i=158811280628252673%5E162568843699643261+(mod+597782322723352841)

Step 2: convert the decimal value to hexadecimal.

20381785731855208 = 4869204a6f73682 in hexadecimal.
9323996581470240 = 21202020202020 in hexadecimal.

There are many websites that can do this, e.g., http://www.rapidtables.com/convert/number/decimal-to-hex.htm

Step 3: convert the hex to ASCII.

4869204a6f73682 = ‘Hi Josh’
21202020202020 = ‘!   ’

This website and many others can do this. http://www.rapidtables.com/convert/number/hex-to-ascii.htm

****

Jay’s Python code for creating and decrypting messages

```
import binascii

def multinv(modulus, value):
    """
    Multiplicative inverse in a given modulus

    https://stackoverflow.com/questions/8539441/private-public-
    encryption-in-python-with-standard-library
    """
    x, lastx = 0, 1
    a, b = modulus, value
    while b:
        a, q, b = b, a // b, a % b
        x, lastx = lastx - q * x, x
    result = (1 - lastx * modulus) // value

    return result + modulus if result < 0 else result

def keygen(e):
    """
    Generate public and private keys from primes up to N.

    https://stackoverflow.com/questions/8539441/
    private-public-encryption-in-python-with-standard-library
    """

    # prime values selected from The first fifty million primes
    # https://primes.utm.edu/lists/small/millions/
    prime1 = 608461363
    prime2 = 982449107
    print('prime1, p: %s' % prime1)
    print('prime2, q: %s' % prime2)

    n = prime1 * prime2
    print('n = p*q = %s' % n)
    totient = (prime1 - 1) * (prime2 - 1)
    print('totient: %s' % totient)

    return n, multinv(totient, e)

def decrypt():
    encrypted_code = '235906921140746933 519413736914785423 232330658812077945'

    e = 65537
    n, d = keygen(e)

    msg_pieces = encrypted_code.split()
    print(msg_pieces)
    count = 0
    decoded_blocks = ''
    for code in msg_pieces:
        count += 1
        print('******Block %i*******' % count)
        print('Encrypted decimal message: %s' % code)
        dec_plaintext = pow(int(code), d, n)   # decode using a private key
        print('Decrypted decimal message: %s' % dec_plaintext)
        hex_plaintext = "{0:x}".format(dec_plaintext)
        print('Decrypted hex message: %s' % hex_plaintext)
        decoded = binascii.unhexlify(str.encode(hex_plaintext))
```

```python
        decoded_blocks += decoded.decode("utf-8")
        print('Decrypted ASCII message: %s\n' % decoded)

    print('\nDecrypted Message: ')
    print(decoded_blocks)


if __name__ == "__main__":

    #decrypt()

    # hex_msg = binascii.hexlify(b'Happy 30th Birthday Josh!')
    # print(hex_msg)
    # print('Decimal int %s' % int(hex_msg, 16))
    # 4543405275772514086587175997509075993917548955529303914844449

    message = 'Lord Marcel, you are in danger! Arm yourself and prepare for battle!'
    # message = 'Hi Josh!       '
    # message = '38deg 34\' 48" N 89deg 59\' 10" W at 10am on July 15'
    # message = 'Time and place'
    # message = 'No. The appropriate weaponry will be provided during your quest.'
    # message = '/c3.pdf'

    # Pad message if necessary
    chunks = int(14 / 2)
    if len(message) % chunks != 0:
        chars_in_last_block = len(message) % chunks
        spaces_to_add = chunks - chars_in_last_block
        print('There are %s chars in last block. Adding %s spaces.' %
              (chars_in_last_block, spaces_to_add))
        message = message + ' ' * (chunks-chars_in_last_block)
        print('New message length is %s, with %i %s-character blocks.' %
              (len(message), len(message)/chunks, chunks))

    e = 65537
    n, d = keygen(e)
    print('n = %i, d = %i, e = %i.' % (n, d, e))
    # n = 597782322723352841
    # d = 162568843699643261

    print('\nMessage: %s\n' % message)
    msg_pieces = [message[i:i + chunks] for i in range(0, len(message), chunks)]
    count = 0
    code_blocks = ''
    decoded_blocks = ''
    for msgs in msg_pieces:
        count += 1
        hex_chars = map(hex, map(ord, msgs))
        hex_value = "".join(c[2:4] for c in hex_chars)
        print('******Block %i******' % count)
        print('Unencrypted ACSII message: "%s"' % msgs)
        print('Unencrypted hex message: %s' % hex_value)
        i = int(hex_value, 16)
        print('Unencrypted decimal message: %s' % i)
        code = pow(i, e, n)  # encode using a public key
        code_blocks += str(code) + ' '
        print('Encrypted decimal message: %s' % code)
        dec_plaintext = pow(code, d, n)  # decode using a private key
        print('Decrypted decimal message: %s' % dec_plaintext)
        hex_plaintext = "{0:x}".format(dec_plaintext)
        print('Decrypted hex message: %s' % hex_plaintext)
        decoded = binascii.unhexlify(str.encode(hex_plaintext))
        decoded_blocks += decoded.decode("utf-8")
        print('Decrypted ASCII message: %s\n' % decoded)

    print('Encrypted Message: ')
    print(code_blocks)

    print('\nDecrypted Message: ')
    print(decoded_blocks)
```

****

First message: “Lord Marcel, you are in danger! Arm yourself and prepare for battle.”

From the Python code above:

```
There are 5 chars in last block. Adding 2 spaces.
New message length is 70, with 10 7-character blocks.
prime1, p: 608461363
prime2, q: 982449107
n = p*q = 597782322723352841
totient: 597782321132442372
n = 597782322723352841, d = 162568843699643261, e = 65537.

Message: Lord Marcel, you are in danger! Arm yourself and prepare for battle!

******Block 1******
Unencrypted ACSII message: "Lord Ma"
Unencrypted hex message: 4c6f7264204d61
Unencrypted decimal message: 21514635326803297
Encrypted decimal message: 16172526842824332
Decrypted decimal message: 21514635326803297
Decrypted hex message: 4c6f7264204d61
Decrypted ASCII message: b'Lord Ma'

******Block 2******
Unencrypted ACSII message: "rcel, y"
Unencrypted hex message: 7263656c2c2079
Unencrypted decimal message: 32197434602692729
Encrypted decimal message: 486461013351423796
Decrypted decimal message: 32197434602692729
Decrypted hex message: 7263656c2c2079
Decrypted ASCII message: b'rcel, y'

******Block 3******
Unencrypted ACSII message: "ou are "
Unencrypted hex message: 6f752061726520
Unencrypted decimal message: 31372504349173024
Encrypted decimal message: 372855876819966270
Decrypted decimal message: 31372504349173024
Decrypted hex message: 6f752061726520
Decrypted ASCII message: b'ou are '

******Block 4******
Unencrypted ACSII message: "in dang"
Unencrypted hex message: 696e2064616e67
Unencrypted decimal message: 29675957956734567
Encrypted decimal message: 173013308343788726
Decrypted decimal message: 29675957956734567
Decrypted hex message: 696e2064616e67
Decrypted ASCII message: b'in dang'

******Block 5******
Unencrypted ACSII message: "er! Arm"
Unencrypted hex message: 6572212041726d
Unencrypted decimal message: 28554459248423533
Encrypted decimal message: 175628482683316976
Decrypted decimal message: 28554459248423533
Decrypted hex message: 6572212041726d
Decrypted ASCII message: b'er! Arm'

******Block 6******
Unencrypted ACSII message: " yourse"
Unencrypted hex message: 20796f75727365
Unencrypted decimal message: 9140718873506661
Encrypted decimal message: 51457655426288031
Decrypted decimal message: 9140718873506661
Decrypted hex message: 20796f75727365
Decrypted ASCII message: b' yourse'

******Block 7******
Unencrypted ACSII message: "lf and "
Unencrypted hex message: 6c6620616e6420
Unencrypted decimal message: 30511586744362016
Encrypted decimal message: 112532939420200262
Decrypted decimal message: 30511586744362016
Decrypted hex message: 6c6620616e6420
Decrypted ASCII message: b'lf and '

******Block 8******
Unencrypted ACSII message: "prepare"
Unencrypted hex message: 70726570617265
Unencrypted decimal message: 31650977394291301
Encrypted decimal message: 103102526154905960
Decrypted decimal message: 31650977394291301
Decrypted hex message: 70726570617265
Decrypted ASCII message: b'prepare'

******Block 9******
Unencrypted ACSII message: " for ba"
Unencrypted hex message: 20666f72206261
Unencrypted decimal message: 9119828096868961
Encrypted decimal message: 212110150840885429
Decrypted decimal message: 9119828096868961
Decrypted hex message: 20666f72206261
Decrypted ASCII message: b' for ba'

******Block 10******
Unencrypted ACSII message: "ttle!  "
Unencrypted hex message: 74746c65212020
Unencrypted decimal message: 32779106200395808
Encrypted decimal message: 116036075239775253
Decrypted decimal message: 32779106200395808
Decrypted hex message: 74746c65212020
Decrypted ASCII message: b'ttle!  '

Encrypted Message:
16172526842824332 486461013351423796 372855876819966270 173013308343788726
175628482683316976 51457655426288031 112532939420200262 103102526154905960
212110150840885429 116036075239775253

Decrypted Message:
Lord Marcel, you are in danger! Arm yourself and prepare for battle!
```