

```

1  #!/bin/bash
2  FILES=*.crt
3  for f in $FILES
4  do
5      openssl x509 -inform der -in $f -noout -text -modulus
6  done

```

This generates a file with all moduli. Let us try something simple! Common moduli! For each pair, we check if $\gcd(N_i, N_j) \neq 1$. If so, we have found a factor. Turns out two moduli have a common factor, so we can factor each of them and decrypt their traffic:

```

1  p1 = 14624978432954754503530834093025436424528887629721656242433314177008841
2  q1 = 13641703641026442859999577157189894593018657302316348067195648485637594
3
4  p2 = 15907293165802485134279783331528054615493943045046723135320654093506275
5  q2 = 13641703641026442859999577157189894593018657302316348067195648485637594

```

We can now generate two PEM-keys

```

1  d1 = gmpy.invert(e, (p1 - 1)*(q1 - 1))
2  key = RSA.construct((long(p1*q1), long(e), long(d1)))
3  f = open('privkey.pem', 'w')
4  f.write(key.exportKey('PEM'))
5  f.close()

```

Putting it into Wireshark, we obtain two images:



ASIS{easy_Common_Factor_iS_reAlly_Forensic_N0t_Crypto!!!!}

I totally agree.

Alice, Bob and Rob

We have developed a miniature of a crypto-system. Can you break it?

We only want to break it, don't get so hard on our system!

This is McEliece PKC. The ciphertexts are generated by splitting each byte in blocks of four bits. The following matrix is used as public key:

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

The ciphertext is generated as $\mathbf{m}G + \mathbf{e}$, which is a function from 4 bits to a byte. \mathbf{e} is an error (or perturbation) vector with only one bit set. This defines a map $f: \mathbb{F}_4 \rightarrow \mathbb{F}_8$. So, each plaintext byte is two ciphertext bytes.

We can first create a set of codewords

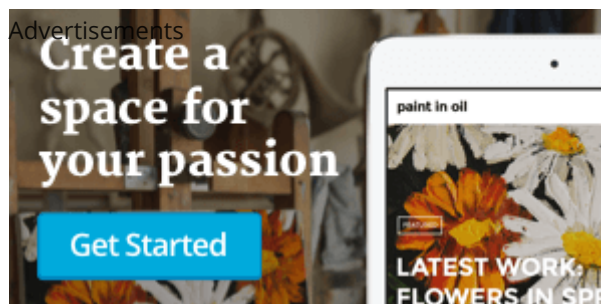
```
1 P = numpy.matrix([[1, 1, 0, 0, 0, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1], [0, 1,
2 image = {} # set of codewords
3 for i in range(0, 2**4):
4     C = (numpy.array([int(b) for b in (bin(i))[2:].zfill(4)]) * P % 2).tolist
5     image[int(''.join([str(c) for c in C]), 2)] = i
```

Then, go through each symbol in the ciphertext, flip all possible bits (corresponding to zeroing out \mathbf{e}) and perform lookup in the set of codewords P (compute the intersection between the Hamming ball of the ciphertext block and P).

```
1 f = open('flag.enc', 'r')
2 out = ''
3 for i in xrange(18730/2):
4     blocks = f.read(2)
5     j = ord(blocks[0])
6     C1 = 0
7     C2 = 0
8     for i in range(0, 8):
9         if j ^ 2**i in image:
10             C1 = image[j ^ 2**i] << 4
11     j = ord(blocks[1])
12     for i in range(0, 8):
13         if j ^ 2**i in image:
14             C2 = image[j ^ 2**i]
15     out += chr(C1+C2)
16 f=open('decrypted', 'w')
17 f.write(out)
```

Turns out it is a PNG:

ASIS{*new_ASIS_CTF_So_new_McEliece_Cryptosystem!!*}



[Blog at WordPress.com.](#)