```
#!/bin/bash
FILES=*.crt
for f in $FILES
do
  openssl x509 -inform der -in $f -noout -text -modulus
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:

```
p1 = 14624978432954754503530834093025436424528887629721656242433314177008841
q1 = 13641703641026442859999577157189894593018657302316348067195648485637594

p2 = 15907293165802485134279783331528054615493943045046723135320654093506275
q2 = 13641703641026442859999577157189894593018657302316348067195648485637594
```

We can now generate two PEM-keys

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

Putting it into Wireshark, we obtain two images:

$\mathbf{ASIS}\{easy\_Common\_Factor\_iS\_re4l1y\_Forensic\_N0t\_Crypto!!!!\}$

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 McElice 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 pertubation) vector with only one bit set. This defines a map $f : \mathbb{F}_4 \to \mathbb{F}_8$. So, each plaintext byte is two ciphertext bytes.

We can first create a set of codewords

```
P = numpy.matrix([[1, 1, 0, 0, 0, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1], [0, 1,
image = {} # set of codewords
for i in range(0, 2**4):
    C = (numpy.array([int(b) for b in (bin(i))[2:].zfill(4)]) * P % 2).tolis
    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 ciperhext block and P).

```
f = open('flag.enc','r')
out = ''
for i in xrange(18730/2):
    blocks = f.read(2)
    j = ord(blocks[0])
    C1 = 0
    C2 = 0
    for i in range(0, 8):
        if j ^ 2**i in image:
            C1 = image[j ^ 2**i] << 4
    j = ord(blocks[1])
    for i in range(0, 8):
        if j ^ 2**i in image:
            C2 = image[j ^ 2**i]
    out += chr(C1+C2)
f=open('decrypted','w')
f.write(out)
```

Turns out it is a PNG:

$\mathbf{ASIS}\{new\_ASIS\_CTF\_So\_new\_McEliece\_Cryptosystem!!\}$