Week 07 Lab

Applied Cryptography (6G6Z0024)

Killian O'Brien

Lab exercises solutions

```
In [1]: import sympy as sp
```

Q9.2 a.

Define the parameters given in the question

```
In [2]: p,q,e,M = 3,7,5,10
        p,q,e,M
```

```
Out[2]: (3, 7, 5, 10)
```

The modulus, n, for the system is given by

```
In [3]: n = p*q
        n
```

```
Out[3]: 21
```

The Euler totient function value $\phi(n)$ is given by

```
In [4]: phi_n = (p-1)*(q-1)
        phi_n
```

```
Out[4]: 12
```

The private key, d, will be the multiplicative invers of e modulo $\phi(n)$, and can be found from the extended Euclidean algorithm for calculating $\gcd(e, \phi(n)) = 1$.

```
In [5]: s = sp.gcdex(e,phi_n)
        s
```

```
Out[5]: (5, -2, 1)
```

```
In [6]: d = s[0]
        d
```

Out[6]: 5

Let's test out the relationship of the pair e, d,

```
In [7]: sp.Mod(e*d,phi_n)
```

Out[7]: 1

The message M is encrypted to the ciphertext C as follows

```
In [8]: C = sp.Mod(M**e,n)
        C
```

Out[8]: 19

Then this ciphertext is decrypted back to the plaintext $M = 10$ by calculating $C^d \pmod{n}$.

```
In [9]: sp.Mod(C**d,n)
```

Out[9]: 10

Q9.2 b.

Similar narrative as in the previous question. Just showing the calculated values here:

```
In [10]: p,q,e,M = 7,17,11,11
         p,q,e,M
```

```
Out[10]: (7, 17, 11, 11)
```

```
In [11]: n = p*q
         phi_n = (p-1)*(q-1)
         n,phi_n
```

```
Out[11]: (119, 96)
```

```
In [12]: s = sp.gcdex(e,phi_n)
         d=s[0]
         d
```

Out[12]: 35

```
In [13]: C = sp.Mod(M**e,n)
         C
```

Out[13]: 114

```
In [14]: sp.Mod(C**d,n)
```

Out[14]: 11

Q9.3

```
In [15]: C,e,n = 20, 13, 77
         C,e,n
```

```
Out[15]: (20, 13, 77)
```

The prime factorisation of n is $n = 7 \cdot 11$. So we have $p = 7, q = 11$.

In [16]:
```
p,q=7,11
phi_n = (p-1)*(q-1)
p,q,phi_n
```

Out[16]:
```
(7, 11, 60)
```

So the private key d is the multiplicative inverse of e modulo 60. Find this with the extended Euclidean algorithm

In [17]:
```
s = sp.gcdex(e,phi_n)
d = s[0]
d
```

Out[17]: -23

But then reduce this modulo 60 to get

In [18]:
```
d = sp.Mod(d,phi_n)
d
```

Out[18]: 37

Then the plaintext M is obtained by decrypting C using this private key.

In [19]:
```
M = sp.gcd(C**d,n)
M
```

Out[19]: 1

So the original plain-text message was $M = 1$.

Q9.4

In [20]:
```
e,n = 65, 2881
e,n
```

Out[20]:
```
(65, 2881)
```

Assuming that $n = p \cdot q$ for some pair of primes p, q we can find them by trial division using the set of primes up to $\sqrt{n}$.

In [21]:
```
L = sp.primerange(1,sp.sqrt(n))
L
```

Out[21]:
```
<generator object primerange at 0x7f057413af60>
```

In [22]:
```
for p in L:
    if sp.Mod(n,p) == 0: # this condition is true if p divides n
        q = n//p
        break
p,q,p*q
```

Out[22]:
```
(43, 67, 2881)
```

Then the Euler totient function value $\phi(n)$ is given by $\phi(n) = (p-1) \cdot (q-1)$

In [23]:
```
phi_n = (p-1)*(q-1)
phi_n
```

Out[23]:
```
2772
```

Then the private key d will be the multiplcative inverse of e modulo $\phi(n)$

In [24]:
```
s = sp.gcdex(e,phi_n)
s
```

Out[24]:
```
(725, -17, 1)
```

In [25]:
```
d=s[0]
d
```

Out[25]: 725

So the associated private key is $d = 725$.

Note that instead we could have used Sympy's `factorint` command to factor n directly as

In [26]:
```
sp.factorint(n)
```

Out[26]:
```
{43: 1, 67: 1}
```

Of course, note that a suitably large n, this `factorint` command will probably be of no practical use. Let's try such a large N and see

In [27]:
```
N = 10**(300) + 27386487263485762398475711
N
```

Out[27]:
```
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000027386487263485762398475711
```

I let this command run for about 10 minutes, but it could not find any factors within that time. Who knows when it might return any factors .... If you do execute this command on your computer. You can interrupt the kernel to stop the coomputation using the Kernel menu above.

In [28]:
```
sp.factorint(N, verbose=True) # verbose=True print status messages about the cal
```

```
Factoring 10000..(291 other digits)..75711
Trial division with ints [2 ... 32768] and fail_max=600
Check for termination
Trial division with primes [1805 ... 3610]
Pollard's p-1 with smoothness bound 1805 and seed 3610
Pollard's rho with retries 1, max_steps 1805 and seed 3610
Trial division with primes [3610 ... 7220]
Pollard's p-1 with smoothness bound 3610 and seed 7220
Pollard's rho with retries 1, max_steps 3610 and seed 7220
Trial division with primes [7220 ... 14440]
Pollard's p-1 with smoothness bound 7220 and seed 14440
Pollard's rho with retries 1, max_steps 7220 and seed 14440
Elliptic Curve with B1 bound 10000, B2 bound 1000000, num_curves 50
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[28], line 1
----> 1 sp.factorint(N, verbose=True) # verbose=True print status messages about
the calculation

File ~/sage/local/var/lib/sage/venv-python3.12.4/lib/python3.12/site-packages/sym
py/ntheory/factor_.py:1438, in factorint(n, limit, use_trial, use_rho, use_pm1, u
se_ecm, verbose, visual, multiple)
   1436 while(1):
   1437     try:
-> 1438         factor = _ecm_one_factor(n, B1, B2, num_curves)
   1439         ps = factorint(factor, limit=limit - 1,
   1440                        use_trial=use_trial,
   1441                        use_rho=use_rho,
   1442                        use_pm1=use_pm1,
   1443                        use_ecm=use_ecm,
   1444                        verbose=verbose)
   1445         n, _ = _trial(factors, n, ps, verbose=False)

File ~/sage/local/var/lib/sage/venv-python3.12.4/lib/python3.12/site-packages/sym
py/ntheory/ecm.py:267, in _ecm_one_factor(n, B1, B2, max_curve)
    263     # We want to calculate
    264     # f = R.x_cord * S[delta].z_cord - S[delta].x_cord * R.z_cord
    265     f = (R.x_cord - S[delta].x_cord)*\
    266         (R.z_cord + S[delta].z_cord) - alpha + beta[delta]
--> 267     g = (g*f) % n
    268 #Swap
    269 T, R = R, R.add(S[D], T)

KeyboardInterrupt: 
```

```
In [ ]: 
```