

Part 1

Sending the message: "2000" as an encrypted message from Alice to Bob with ElGamal

```
int clear = 2000;
```

For the sender Alice to send the message 2000 to the receiver Bob, you will need for Alice and Bob to agree on a shared prime p and generator g

```
int g = 666;  
int p = 6661;
```

...then you will need Bob to choose generate a public key (using g and p but for this part Bob has already done that "in secret") ...

```
int pubBob = 2227;
```

...then Alice will generate a private key, by choosing a random integer, between 0 and $p-1$, where p is the shared prime.

```
int privAlice = new Random().Next() % p;  
WriteLine($"privAlice: {privAlice}");
```

... finally Alice generates a scalar $B^y \bmod p$ which can be applied to the cleartext, to encrypt the message.

```
int scalar = (int)BigInteger.ModPow(pubBob, privAlice, p);  
int cipher = scalar * clear;  
WriteLine($"cipher: {cipher}");
```

Output:

```
privAlice: 5660  
cipher: 5468000
```

For both steps I've created methods in a class to handle this neatly. So the setup and solution could have looked like so:

```
int g = 666;
int p = 6661;
var elgamal = new ElGamal(g, p);
int clear = 2000;
int pubBob = 2227;
int privAlice = elgamal.GeneratePriv();
int cipher = elgamal.Encrypt(clear, pubBob, privAlice);
```

Part 2

For this part, I initially generate the cipher, and the public keys of both Alice and Bob

```
int clear = 2000;
int pubBob = 2227;

int privAlice = elgamal.GeneratePriv(); // this value is unknown to Eve
int cipher = elgamal.Encrypt(clear, pubBob, privAlice);

int pubAlice = elgamal.GeneratePub(privAlice);
```

Eve can bruteforce the private key of Bob, by trying every x in $g^x \bmod p$ and checking if it produces the same output as Bob's public key.

```
var candidates = elgamal.CrackMsg(cipher, pubBob, pubAlice);
```

Eve can print the found privatekeys and see which messages they produce, when decrypting.

```
WriteLine("Candidates:");
foreach (var (priv, msg) in candidates)
    WriteLine($"priv:{priv} msg:{msg}");
```

Output:

```
Candidates:
priv:66 msg:2000
priv:6726 msg:2000
priv:13386 msg:2000
priv:20046 msg:2000
priv:26706 msg:2000
priv:33366 msg:2000
priv:40026 msg:2000
priv:46686 msg:2000
```

Part 3

Since I intercepted the message, I see no reason why I should consider that ciphertext and not just create a new message from scratch.

I can do that by the following.

Mallory chooses a random number from 0 to $p-1$

```
int priv = elgamal.GeneratePriv();
```

Apply Mallory's private key to the function $x \Rightarrow g^x \bmod p$ to get the public key

```
int pub = elgamal.GeneratePub(priv);
```

Encrypt the message "6000" using Mallory's private key and Bob's public key

```
int cipher = elgamal.Encrypt(6000, 2227, priv);  
WriteLine($"cipher: {cipher}");
```

We can check if we retrieve the correct clear text, when Bob decrypts the cipher, by one of the cracked private keys of Bob from part 2

```
WriteLine($"clear: {elgamal.Decrypt(cipher, pub, 66)}");
```

Output:

```
cipher: 9510000  
clear: 6000
```