Security Mandatory Exercise 01

This is my report about how i have solved the first mandatory exercise.

- Andreas Nicolaj Tietgen
- anti
- anti@itu.dk

How to run the program?

The program is code in Go. The exact version is 1.19.1. In order to run the program then you have to be in the root folder of this project and type in the following:

```
$ go run src/Main.go
```

If you want to build and run the compiled version then you can do that as well:

```
$ go build -o ./SecEx1 src/Main.go
$ ./SecEx1
```

It is also possible to change Alice's secret. That can be done by using the `n` flag.

```
./SecEx1 -n 2090
```

Questions

Question 1

In the first exercise Alice want to send a message encrypted to Bob. We have Bob's public key, so in order to encrypt the message then we first generate the key $bobPk^{s_a} \mod 2227$ with the following code:

```
func calculateKey(base, prime, secret big.Int) *big.Int {

    result := big.NewInt(0)
    result.Exp(&base, &secret, nil)
    result.Mod(result, &prime)
    return result
}
```

Then we take the output of `calculateKey()`, which is the shared key, and factor it with the message $sharedKey * message$.

Question 2

In this part of the exercise we want to intercept the encrypted message we created in the previous part of the exercise. However, we do not know what Bob's secret is. We only have Bob's public key, denoted as pk_b, with the value of 2227, and Alice's public key, denoted as pk_a.

We then brute force the secret until we have calculated the same key public key as pk_b. When we have Bob's secret then we can call the decrypt method $decrypt(s_b, pk_a, cipher)$. The method for intercepting the message looks like the following:

```go
func intercept(targetPk, pk, cipher big.Int) (secret, message big.Int) {

    fmt.Println("Intercepting message...")

    base := big.NewInt(SHARED_BASE)
    prime := big.NewInt(SHARED_PRIME)
    var limit big.Int = *big.NewInt(1000)
    incrementer := big.NewInt(1)

    for s := *big.NewInt(1); s.Cmp(&limit) < 0; s.Add(&s, incrementer) {

        key := calculateKey(*base, *prime, s)

        if key.Cmp(&targetPk) == 0 {

            fmt.Println("Secret has been found!")
            message := decrypt(s, pk, cipher)
            return s, message
        }
    }

    return *big.NewInt(0), *big.NewInt(0)
}
```

And the code in order to decrypt looks like the following:

```go
func decrypt(secret, pk, cipher big.Int) big.Int {

    fmt.Println("Decrypting message...")
    sharedKey := calculateKey(pk, *big.NewInt(SHARED_PRIME), secret)

    result := big.NewInt(0)
    return *result.Div(&cipher, sharedKey)
}
```

Question 3

In part 3 of the exercise it is Mallory that wants to tamper with the message. We want the message to be 6000 instead of 2000. Since we know that the message is 2000, then we can factor the ciphertext like $ciphertext * 3$.

Now when we decrypt the message then the message is 6000.

Example output

Here you can see an example output of the program when running with a $s_a = 596$

```
./SecEx1 -n 596
Assignment 1:
Encrypting...
Alice's public key is: 3256, and the encrypted message is: 12464000

Assignment 2:
Eve has seen the encrypted message from alice and tries to intercept it...
Intercepting message...
Secret has been found!
Decrypting message...
Bob's secret is: 66, and the decrypted message is: 2000

Assignment 3:
Mallory also sees the message from Alice but do not have the
resources to calculate Bob's secret in order to see the message.
Instead Mallory tries to tamper with the message...
Decrypting message...
Message after Mallory has tampered with the ciphertext: 6000
```