Task 2 - Example

Server Configuration showcasing the encryption standards required by a server. The client and server communicate regarding intended ciphersuites to use. Here is a sample showing the ciphersuites the server can use. In this case there is a flag to prefer server cipher suites

```
//Create a configuration detailing supported cipherSuites and Elliptic Curves
cfg := &tls.Config{
    MinVersion:               tls.VersionTLS12,
    CurvePreferences:         []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
    PreferServerCipherSuites: true,
    CipherSuites: []uint16{
        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
        tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_RSA_WITH_AES_256_CBC_SHA,
    },
}
```

Here we start the server ready to listen to requests on port 4043. Note it lists down the configuration the client must use.

```
Run: go build serve.go
<4 go setup calls>
CONNECTING via TLS on PORT 4043
Elliptic Curve Diffie Hellman Exchange (ECDHE) : tls.CurveP521
tls.CurveP384
tls.CurveP256
Acceptable CipherSuites:
tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
tls.TLS_RSA_WITH_AES_256_GCM_SHA384
tls.TLS_RSA_WITH_AES_256_CBC_SHA
```

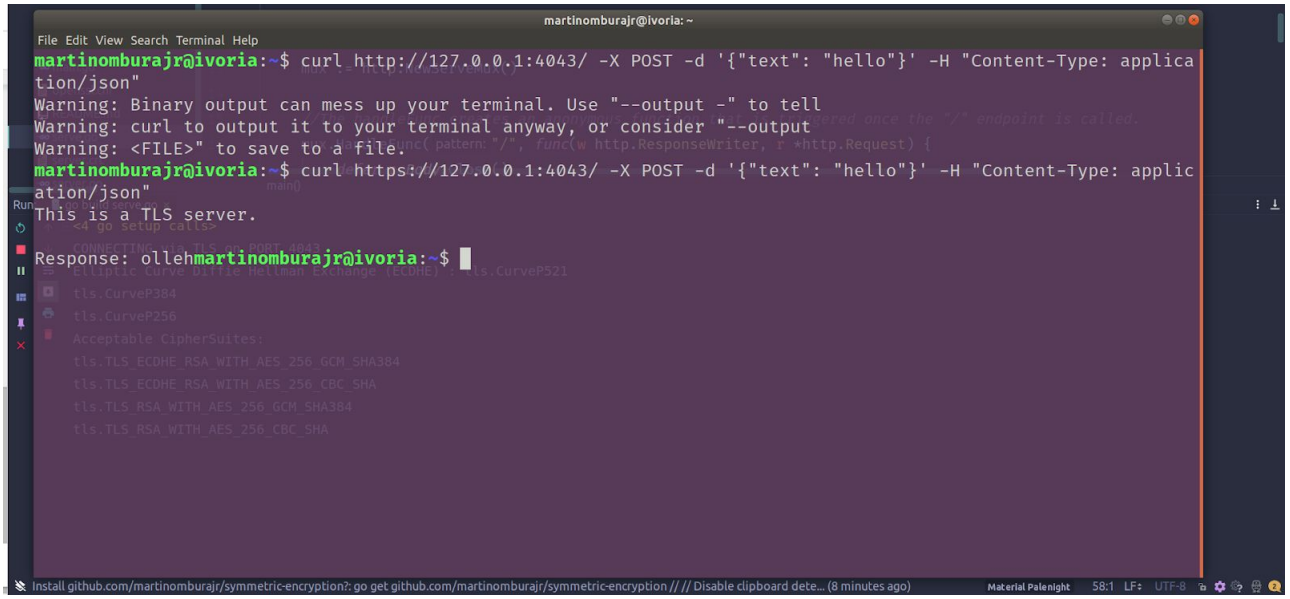
We then open up a terminal and perform a call to the server using the following address 127.0.0.1:4043. If we use http instead of https as the scheme, watch how the server responds.

```
Run: go build serve.go
<4 go setup calls>
CONNECTING via TLS on PORT 4043
Elliptic Curve Diffie Hellman Exchange (ECDHE) : tls.CurveP521
martinomburajr@ivoria: ~
File Edit View Search Terminal Help
martinomburajr@ivoria:~$ curl http://127.0.0.1:4043/ -X POST -d '{"text": "hello"}' -H "Content-Type: applica
tion/json"
Warning: Binary output can mess up your terminal. Use "--output -" to tell
Warning: curl to output it to your terminal anyway, or consider "--output
Warning: <FILE>" to save to a file.
martinomburajr@ivoria:~$
2018/10/22 21:46:25 http: TLS handshake error from 127.0.0.1:54572: tls: oversized record received with length 21536
```

The terminal throws warnings, and the server shown in the background throws a TLS handshake error indicating an issue with the TLS handshake. Therefore the request has not gone through. The handshake

comprises of the client creating a key from a selected algorithm that both client and server agreed to. See ciphersuite agreement. The client encrypts the key with the servers public key

Let us restart our server and use the correct protocol (https)



Here we see no error on both the client and server side as both have agreed to the secure handshake and can communicate using AES under the hood. The server performs a simple text reversal of the clients request and responds with the reversed text. The server here is able to decrypt the key with its private key and use it to read the message and respond via the text.

Files c2.go and s2.go represent a naive symmetric encryption that takes place under the hood of the serve.go file

In this case shown below, a cipher is being generated by the key and the input text

```go
    block, cipherErr := aes.NewCipher(key)

    if cipherErr != nil {
        fmt.Errorf( format: "Can't create cipher:", cipherErr)

        return
    }

    iv := make([]byte, aes.BlockSize)

    if _, randReadErr := io.ReadFull(rand.Reader, iv); randReadErr != nil {
        fmt.Errorf( format: "Can't build random iv", randReadErr)

        return
    }
```