

Gestor de Contraseñas

Práctica de SEGURIDAD DISEÑO SOFTWARE

Integrantes:

Jorge Segovia Tormo

Javier Molpeceres Gómez

Francisco José Maciá Esclapez

Indice

Otras alternativas

Nuestra Práctica

- Certificados
- Comunicación
- Seguridad
 - Bcrypt
 - AES
- Usuarios y cuentas
- Optativo
 - Doble autenticación con Correo
 - Conocimiento 0
 - Log de la aplicación Servidor
 - Información adicional Usuarios
 - Cifrado archivos con contraseña maestra
- Metodología de trabajo
- Puesta en marcha
- Ejemplo de ejecución

Otras alternativas



Dashlane Password Manager

Dashlane es una de las aplicaciones gratuitas para la gestión de contraseñas mejor posicionadas gracias a sus múltiples características y no está de más decir que en año 2015 fue elegida como una de las mejores aplicaciones por el Play Store y por el App Store.

- Ofrece encriptación AES-256
- Genera alertas de seguridad a través de correo electrónico o mensajes de texto
- Incluye un generador de contraseñas seguras
- Posee respaldo constante de las contraseñas



Keeper Password Manager

Keeper es otra de las opciones gratuitas para ambientes Windows que nos ofrece una integridad y seguridad a la hora del manejo de nuestras contraseñas. Gracias a Keeper contaremos con una aplicación que ha sido desarrollada con los más altos estándares de seguridad ya que usa el cifrado AES-256 como base de encriptación.

- Cuenta con un generador de contraseñas seguro
- Cuenta con aplicación móvil y la posibilidad de usar el acceso con huella aumentado los niveles de seguridad
- Incluye verificación en dos pasos gracias a la tecnología Keeper DNA®
- Cuenta con encriptación AES-256 y PBKDF2



LastPass Password Manager

Es una aplicación multiplataforma con la que todas nuestras contraseñas estarán seguras gracias a sus características avanzadas y niveles de encriptación AES-256.

- Usa nivel de encriptación AES de 256 bits incluyendo algoritmos hash con sal y PBKDF2 SHA-256.

- Posibilidad de compartir las contraseñas de forma segura
- Cuenta con factor de autenticación de doble factor
- Posee un generador de contraseñas

Nuestra Práctica

Certificados

Hemos creado los certificados

Usando: **Key considerations for algorithm "RSA" \geq 2048-bit**

```
openssl genrsa -out server.key 2048
```

```
openssl ecparam -genkey -name secp384r1 -out server.key
```

```
openssl req -new -x509 -sha256 -key server.key -out server.crt -days 3650
```

Para la comunicación algo similar en base a esto en el cliente y servidor:

```

func main() {
    log.SetFlags(log.Lshortfile)

    cer, _ := tls.LoadX509KeyPair("server.crt", "server.key")

    config := &tls.Config{Certificates: []tls.Certificate{cer}}
    ln, _ := tls.Listen("tcp", ":443", config)

    defer ln.Close()

    for {
        conn, _ := ln.Accept()

        go handleConnection(conn)
    }
}

//conexion entre el cliente y el servidor
func handleConnection(conn net.Conn) {
    defer conn.Close()
    r := bufio.NewReader(conn)
    for {
        msg, _ := r.ReadString('\n')

        println("Mensaje recibido:")
        println(msg)
        println("mensaje a responder(enviar):")
        var linea string
        fmt.Scanf("%s\n", &linea)

        conn.Write([]byte(linea + "\n"))
        //n, _err := conn.Write([]byte(linea + "\n"))

    }
}

```

Comunicación

Para la comunicación hemos realizado un tunel TLS con cabeceras entre cliente y servidor. Para ello hemos creado una estructura con estos campos:

Servidor:

```

type peticion struct {
    Tipo    string    `json:"tipo"`
    Cookie  string    `json:"cookie"`
    Usuario usuario    `json:"usuario"`
    Clave   string    `json:"clave"`
    Cuentas []]cuenta `json:"cuenta"`
}

type usuario struct {
    Name      string    `json:"nombre"`
    Correo    string    `json:"correo"`
    Cuentas []]cuenta `json:"cuentas"`
}

type cuenta struct {
    Usuario      string `json:"usuario"`
    Contraseña   string `json:"contraseña"`
    Servicio     string `json:"servicio"`
}

```

Lo importante sería la cabecera tipo que sería la que redigiría las peticiones del cliente. Por otro lado, tenemos la cookie para la sesión, usuario que es un tipo de usuario y cuentas.

Cliente: El cliente tendría la misma petición y la estructura respuesta donde recibe si la petición ha sido realizada correctamente o no:

```

type peticion struct {
    Tipo    string    `json:"tipo"`
    Cookie  string    `json:"cookie"`
    Usuario usuario    `json:"usuario"`
    Cuentas []]cuenta `json:"cuentas"`
    Clave   string    `json:"clave"`
}

type respuesta struct {
    Estado      string `json:"estado"`
    Cookie      string `json:"cookie"`
    TipoCuerpo  string `json:"tipocuerpo"`
```

Para hacerlo único y que el hash no coincida con el de otro usuario utilizamos como usuario la suma de "usuario"+"contraseña":

```
password := []byte(nombre + contraseñaUsuario)
```

A la hora de comprobar el usuario, lo comprobamos recorriendo todos los archivos y realizando un compare hash de "usuario"+"contraseña":

```
//el nombre de los archivos está almacenado en el ficheros usuarios.json
var listaUSR = JSONtoUsuariosBD(leerArchivo("usuarios.json"))

for _, obj := range listaUSR {
    err := bcrypt.CompareHashAndPassword([]byte(obj.Name), []byte(usuario.Name
))
    if err == nil {
        //Primer paso autenticación
        //Compronación correo
    }
}
```

AES

Para todos los datos que necesitamos poder recuperar sin proporcionarlos nosotros como sería el caso del hash hemos utilizado AES.

Para cifrar y descifrar:

```

func encriptar(datosPlanos []byte, key []byte) string {
    plaintext := datosPlanos

    block, err := aes.NewCipher(key)
    if err != nil {
        panic(err)
    }

    ciphertext := make([]byte, aes.BlockSize+len(plaintext))
    iv := ciphertext[:aes.BlockSize]
    if _, err := io.ReadFull(rand.Reader, iv); err != nil {
        panic(err)
    }

    stream := cipher.NewCFBEncrypter(block, iv)
    stream.XORKeyStream(ciphertext[aes.BlockSize:], plaintext)

    return base64.URLEncoding.EncodeToString(ciphertext)
}

func desencriptar(datosEncriptados string, key []byte) string {

    ciphertext, _ := base64.URLEncoding.DecodeString(datosEncriptados)

    block, err := aes.NewCipher(key)
    if err != nil {
        panic(err)
    }

    if len(ciphertext) < aes.BlockSize {
        panic("ciphertext too short")
    }
    iv := ciphertext[:aes.BlockSize]
    ciphertext = ciphertext[aes.BlockSize:]

    stream := cipher.NewCFBDecrypter(block, iv)

    stream.XORKeyStream(ciphertext, ciphertext)

    return fmt.Sprintf("%s", ciphertext)
}

```

Como observamos pasamos el texto a cifrar o descifrar y la clave para realizar el trabajo. Para la clave usamos la clave que de el usuario y la completamos con 0 al final hasta llegar a 16 caracteres.

```
func obtenerkeyUsuario(contraseña string) []byte {
    var salida string
    salida = contraseña
    for {
        if len(salida) == 16 {
            return []byte(salida)
        } else {
            salida = salida + "0"
        }
    }
}
```

Usuarios y cuentas

Pasaremos a comentar el trabajo realizado para la gestion de usuarios en esta práctica, explicando esto en distintas secciones.

Usuarios

Los usuarios son una parte fundamental de la aplicación estos son los clientes de nuestra aplicacion, personas que quieren almacenar información sobre sus cuentas(login,contraseña y servicio).

Estructura

Para esto hemos definido una estructura tanto en cliente como en servidor:

cliente

```
type usuario struct {
    Name      string    `json:"nombre"`
    Correo    string    `json:"correo"`
    Cuentas   []cuenta `json:"cuentas"`
}
```

servidor

```
type usuario struct {
    Name      string    `json:"nombre"`
    Correo    string    `json:"correo"`
    Cuentas   []cuenta `json:"cuentas"`
}
```

Como podemos ver comparten estructura tanto en cliente como servidor, esto es debido a que a la hora de trabajar en ambos casos es necesario que la estructura sea común, como por ejemplo para el trabajo en la

conversion JSON.

JSON usuarios

Dado que en esta práctica el almacenamiento se ha realizado en ficheros JSON nos es necesario realizar una transformación de los datos a almacenar en JSON.

```
func usuarioToJSON(user usuario) []byte { //Crear el json
    resultado, _ := json.Marshal(user)
    return resultado
}
```

Esta función está en el lado del cliente dado que los datos que recibe el servidor estarán listos para ser escritos en el fichero para más seguridad del sistema.

```
func jsonToUsuario(user []byte) usuario { //desjson
    var usuarioDescifrado usuario
    json.Unmarshal(user, &usuarioDescifrado)
    return usuarioDescifrado
}
```

Esta función realiza lo contrario, transforma el mensaje del JSON a una variable del tipo usuario.

Cuentas

Otra parte a tener en cuenta es la estructura de las cuentas, las cuentas es el núcleo principal de la información del usuario, aquí es donde podemos encontrar toda la información sobre las cuentas(contraseña,login,servicio).

Estructura cuentas

```
type cuenta struct {
    Usuario    string `json:"usuario"`
    Contraseña string `json:"contraseña"`
    Servicio   string `json:"servicio"`
}
```

JSON cuentas

Al igual que con los usuarios podemos ver que se ha trabajado el JSON tanto de transformación de la estructura a JSON como del JSON a la estructura.

```
func cuentasToJSON(cuent []cuenta) []byte { //Crear el json
    resultado, _ := json.Marshal(cuent)
    return resultado
}
```

```
func jsonToCuentas(datos []byte) []cuenta {
    var listadeCuentas []cuenta
    json.Unmarshal(datos, &listadeCuentas)
    return listadeCuentas
}
```

Lógica trabajo usuario/cuentas

A continuación explicaremos la lógica que sigue la aplicación para el trabajo con cuentas de los usuarios.

En primer lugar cabe hablar de que la aplicación tiene en el lado del servidor un archivo con la información de los usuarios que existen en la aplicación, este archivo contiene una lista en JSON de los usuarios existentes, esto es debido a que cada usuario tiene su propio archivo de información de cuentas por ello es necesario controlar la existencia de usuarios por este archivo. El archivo en cuestión es "usuarios.json" todos los datos, nombres de estos usuarios están cifrados.

Creación usuarios

Los usuarios se crean mediante el cliente, en este cliente los usuarios deciden si crearse una cuenta y que nombre y contraseña tendrá, se le enviará una confirmación por correo al realizar acceso a sus datos más adelante.

cliente

```
func crearUsuario() {

    //Datos de usuario
    var nombre string
    var correo string
    var contraseñaUsuario string
    var contes []cuenta

    //Datos de cuenta
    var usuarioNombre string
    var contraseñaCuenta string
    var servicio string

    //Booleano de añadir cuenta
    var crear string
}
```

```

//Datos de Usuario
println("Nombre del usuario")
fmt.Scanf("%s\n", &nombre)

println("Contraseña")
fmt.Scanf("%s\n", &contraseñaUsuario)

println("Correo del usuario")
fmt.Scanf("%s\n", &correo)

//Añadir primera cuenta
println("¿Deseas añadir una cuenta?")
fmt.Scanf("%s\n", &crear)

//key del usuario
var key []byte
key = obtenerkeyUsuario(contraseñaUsuario)
keyuser = key
if crear == "si" {
    for crear != "no" {
        println("Usuario:")
        fmt.Scanf("%s\n", &usuarioNombre)
        println("Contraseña:")
        fmt.Scanf("%s\n", &contraseñaCuenta)
        println("Servicio:")
        fmt.Scanf("%s\n", &servicio)
        n := cuenta{encriptar([]byte(usuarioNombre), key), encriptar([]byte(contraseñaCuenta), key), encriptar([]byte(servicio), key)}
        contes = append(contes, n)
        println("¿Deseas añadir otra cuenta?")
        fmt.Scanf("%s\n", &crear)
    }
}

//
//HASH USUARIO CONTRASEÑA
//
password := []byte(nombre + contraseñaUsuario)
// Hashing the password with the default cost of 10
hashedPassword, _ := bcrypt.GenerateFromPassword(password, bcrypt.DefaultCost)

for strings.Contains(string(hashedPassword), "/") { //Lo realizamos para que no genere con / ya que a la hora de directorios da problemas
    hashedPassword, _ = bcrypt.GenerateFromPassword(password, bcrypt.DefaultCost)
}
//fmt.Println(string(hashedPassword))
nombre = string(hashedPassword)

```

```
//
//HASH USUARIO CONTRASEÑA
//

user := usuario{nombre, correo, contes}
pet := peticion{"crearUsuario", "null", user, nil, ""}
var peti = peticionToJSON(pet)
comunicacion(peti)
}
```

Como podemos ver en el código anterior se le pide al usuario que introduzca sus datos(nombre contraseña y correo) y se le pregunta si quiere añadir cuentas.Después se cifra esta información y se pasa a realizar la comunicación con el servidor.

Servidor

El servidor recibe la petición de creación y pasa a crear el usuario, introduciendo en el archivo usuarios.json a este nuevo usuario y creando el archivo propio del usuario con sus datos.

Tratamiento petición servidor

```
case "crearUsuario":
    if creacionUsuarioPorPetición(pet) {
        // "-----\nUsuario Creado\n-----"

        res := respuesta{"Correcto", getCookieUsuarios("").Oreo, "string", []byte("Usuario creado correctamente")} //falta meter la cookie
        resp = respuestaToJSON(res)

    } else {
        // "-----\nUsuario ya Existente\n-----"
        res := respuesta{"Incorrecto", getCookieUsuarios("").Oreo, "string", []byte("Usuario no creado, ya existe un usuario")}
        resp = respuestaToJSON(res)
    }
}
```

Creación usuario

```

func creacionUsuarioPorPeticion(pet peticion) bool {
    var correcto = false
    var usuarios = jSONtoUsuariosBD(leerArchivo("usuarios.json"))
    var usuarioNuevo usuarioBD
    usuarioNuevo.Name = pet.Usuario.Name
    usuarioNuevo.Correo = pet.Usuario.Correo
    //println("AQUI " + pet.Usuario.Name)

    if !comprobarExistenciaUSR(usuarios, usuarioNuevo) {
        //println("ENTRA")
        var nombre string

        nombre = pet.Usuario.Name

        deleteFile("usuarios.json")
        createFile("usuarios.json")
        usuarioNuevo.Name = nombre
        var nuevalista = append(usuarios, usuarioNuevo)
        if escribirArchivoClientes("usuarios.json", string(usuariosBDToJSON(nuevalista))) {
            createFile(nombre + ".json")
            if pet.Usuario.Cuentas != nil {
                if escribirArchivoClientes(nombre+".json", string(cuentasToJSON(pet.Usuario.Cuentas))) {
                    correcto = true
                } else {
                    correcto = false
                }
            } else {
                if escribirArchivoClientes(nombre+".json", "[ ]") {
                    correcto = true
                } else {
                    correcto = false
                }
            }
        }
        setCookie(tamCookie)
    }

    return correcto
}

```

Aquí podemos ver como en la creacion del usuario lo que se realiza es comprobar si existe este usuario previamente, en caso de que exista se le comunica al cliente que no se ha podido crear su usuario ya que ya existe. En caso de que no exista pasariamos a introducir este usuario en el archivo usuarios.json, el siguiente paso a realizar seria crear el documento con el nombre del usuario(encriptado) e introducimos en el las cuentas del usuario encriptadas.

Gestión de cuentas

Hablaremos aquí de como se le ofrece al usuario gestionar sus cuentas. Una vez creada una la cuenta en nuestra aplicación el usuario se puede loggear, cuando el usuario se ha logeado satisfactoriamente puede elegir gestionar sus cuentas pudiendo borrar, modificar y listar estas cuentas.

Listar Cuentas

Ofrecemos al usuario la posibilidad de listar sus cuentas disponibles obteniendo la información sobre todas sus cuentas. Podemos ver que si el usuario accede a la opción de listar las cuentas en el cliente se realiza una petición al servidor de la siguiente manera.

```
func listarCuentas() {  
  
    var listCuentasdisponbils []cuenta  
    UsuarioConectado.Cuentas = nil  
  
    pet := peticion{"getcuentas", sesionUsuario.Valor, UsuarioConectado, nil, ""}  
  
    var peti = peticionToJSON(pet)  
    var comunicacionDel = comunicacion(peti)  
    var respuesta = jSONtoRespuesta([]byte(comunicacionDel))  
    cuentasRespuesta := jSONtoCuentas(respuesta.Cuerpo)  
    for _, obj := range cuentasRespuesta {  
        listCuentasdisponbils = append(listCuentasdisponbils, cuenta{desencriptar(obj.Usuario, keyuser), desencriptar(obj.Contraseña, keyuser), desencriptar(obj.Servicio, keyuser)})  
    }  
    listaCuentas(listCuentasdisponbils)  
}
```

El servidor recibe esta petición por medio del menu de conexión.

```
case "getcuentas":  
  
    if comprobarCookieValida(pet) {  
  
        res := respuesta{"Correcto", getCookieUsuarios(obtenerUsuarioCookie(pet)).Oreo, "string", devolvercuentasUsuario(pet)}  
        resp = respuestaToJSON(res)  
    } else {  
        fmt.Print("sesion caudcada")  
    }  
}
```

y realiza la siguiente gestión para leer el archivo del cliente que al estar cifrado necesitará descifrarlo(sin descifrar sus datos) ya que tenemos un doble cifrado(de datos y del archivo completo).

```

func devolvercuentasUsuario(pet peticion) []byte {
    var listaUSR = JSONtoUsuariosBD(leerArchivo("usuarios.json"))

    for _, obj := range listaUSR {
        err := bcrypt.CompareHashAndPassword([]byte(obj.Name), []byte(pet.Usuario.
Name))

        if err == nil {

            return leerArchivo(obj.Name + ".json")
        }
    }
    return []byte("error al ler archivo")
}

```

El cliente será el encargado de descifrar los datos que recibe del servidor y mostrárselo al cliente.

Eliminar cuenta

En esta opcion el usuario podrá borrar la cuenta que decida y seleccione, en primer lugar esta opcion le muestra todas las cuentas disponibles(sin contraseña) y tendrá que introducir el nombre de la cuenta y el servicio al que pertenece esta cuenta, ya que podemos tener cuentas con el mismo nombre pero distinto servicio.

En primer lugar en el cliente gestionamos el borrado mediante el siguiente método:

```

func borrarCuentaServicio() {

    var cuentaname string
    var servicio string

    var nuevaListaCuentas []cuenta
    var listCuentasdisponbls []cuenta

    UsuarioConectado.Cuentas = nil
    pet := peticion{"delcuentas", sesionUsuario.Valor, UsuarioConectado, nil, ""}

    var peti = peticionToJSON(pet)
    var comunicacionDel = comunicacion(peti)
    var respuesta = jSONtoRespuesta([]byte(comunicacionDel))
    //println(string(respuesta.Cuerpo))
    cuentasRespuesta := jSONtoCuentas(respuesta.Cuerpo)
    for _, obj := range cuentasRespuesta {
        listCuentasdisponbls = append(listCuentasdisponbls, cuenta{desencriptar(obj.Usuario, keyuser), obj.Contraseña, desencriptar(obj.Servicio, keyuser)})
    }
    menuBorrado(listCuentasdisponbls)
    fmt.Print("Introduce cuenta a borrar: ")
    fmt.Scanf("%s\n", &cuentaname)
    fmt.Print("Introduce servicio a borrar: ")
    fmt.Scanf("%s\n", &servicio)

    for _, obj := range listCuentasdisponbls {
        if obj.Usuario != cuentaname || obj.Servicio != servicio {
            obj = cuenta{encriptar([]byte(obj.Usuario), keyuser), obj.Contraseña, encriptar([]byte(obj.Servicio), keyuser)}
            nuevaListaCuentas = append(nuevaListaCuentas, obj)
        }
    }
    UsuarioConectado.Cuentas = nuevaListaCuentas
    peticionActu := peticion{"delcuentas", sesionUsuario.Valor, UsuarioConectado, nuevaListaCuentas, ""}

    var petiActu = peticionToJSON(peticionActu)
    var comunicacionActu = comunicacion(petiActu)
    var respuestaActu = jSONtoRespuesta([]byte(comunicacionActu))

    if string(respuestaActu.Estado) == "Correcto" {
        println("Borrado realizado correctamente")
    } else if string(respuestaActu.Estado) == "Incorrecto" {
        println("Borrado no realizado")
    }
}

```


Como podemos ver este método envía una petición al servidor, el cual detecta que al ser una petición enviada sin cuentas en el cuerpo(es decir sin una modificación) se le está pidiendo que devuelva las cuentas que hay disponibles para borrar. Esto es debido a que la lógica seguida para el borrado es reutilizando la modificación por eso se envía previamente una petición sin cuentas en el cuerpo así detectando el servidor que no es la modificación si no la petición del cliente de las cuentas que se pueden borrar.

Podemos ver la gestión en el servidor:

```
case "delcuentas":
    if comprobarCookieValida(pet) {
        if pet.Usuario.Cuentas == nil {
            var stin = devolvercuentasUsuario(pet)
            res := respuesta{"Correcto", getCookieUsuarios(obtenerUsuarioCookie(pet)).Oreo, "string", stin}
            resp = respuestaToJSON(res)

        } else {

            var resul = actualizarcuentas(pet)
            if resul {
                res := respuesta{"Correcto", getCookieUsuarios("").Oreo, "string", []byte("Cuenta Borrada")}
                resp = respuestaToJSON(res)
            } else {
                res := respuesta{"Incorrecto", getCookieUsuarios("").Oreo, "string", []byte("Cuenta no borrada")}
                resp = respuestaToJSON(res)
            }
        }
    } else {
        fmt.Print("sesion caudcada")
    }
}
```

Como podemos ver lo que realiza es primero devolver las cuentas, si el cuerpo de la petición esta vacío, si no, realiza una actualización. Hablaremos en el siguiente punto de la actualización.

Actualizar cuentas

En esta parte al usuario se le ofrece la posibilidad de actualizar sus cuentas, su información, contraseña etc. Este es una funcionalidad importante ya que también se le ofrece al usuario la posibilidad de añadir cuentas nuevas introduciendo en los primeros campos de referencia hacia la cuenta a modificar el nombre y datos de la cuenta que desea crear.

En el cliente se le gestiona de la siguiente forma:

```
func modificarCuentas() {

    var cuentaname string
    var servicio string
    var password string
    var cuentaNNombre string
    var servicioNServi string
    var nuevoPassword string

    var nuevaListaCuentas []cuenta
    var listCuentasdisponbls []cuenta

    UsuarioConectado.Cuentas = nil
    pet := peticion{"actualizarCuenta", sesionUsuario.Valor, UsuarioConectado, nil
, ""}

    var peti = peticionToJSON(pet)
    var comunicacionDel = comunicacion(peti)
    var respuesta = jSONtoRespuesta([]byte(comunicacionDel))
    //println(string(respuesta.Cuerpo))
    cuentasRespuesta := jSONtoCuentas(respuesta.Cuerpo)
    for _, obj := range cuentasRespuesta {
        listCuentasdisponbls = append(listCuentasdisponbls, cuenta{desencriptar(obj.Usuario, keyuser), desencriptar(obj.Contraseña, keyuser), desencriptar(obj.Servicio, keyuser)})
    }
    listaCuentas(listCuentasdisponbls)
    fmt.Print("Introduce cuenta a modificar: ")
    fmt.Scanf("%s\n", &cuentaname)
    fmt.Print("Introduce servicio a modificar: ")
    fmt.Scanf("%s\n", &servicio)
    fmt.Print("Introduce contraseña del servicio a modificar: ")
    fmt.Scanf("%s\n", &password)

    fmt.Print("¿Quieres cambiar el nombre?")
    var cambiarNombre string
    fmt.Scanf("%s\n", &cambiarNombre)

    if cambiarNombre == "si" || cambiarNombre == "SI" {
        fmt.Print("Introduce nuevo nombre para la cuenta " + cuentaname + ": ")
        fmt.Scanf("%s\n", &cuentaNNombre)
    } else {
        cuentaNNombre = cuentaname
    }

    fmt.Print("¿Quieres cambiar el servicio?")
```

```

var cambiarServicio string
fmt.Scanf("%s\n", &cambiarServicio)

if cambiarServicio == "si" || cambiarServicio == "SI" {
    fmt.Print("Introduce nuevo servicio para la cuenta " + cuentaname + " del
servicio " + servicio + ": ")
    fmt.Scanf("%s\n", &servicioNServi)
} else {
    servicioNServi = servicio
}

fmt.Print("¿Quieres cambiar la contraseña?")
var cambiarPass string
fmt.Scanf("%s\n", &cambiarPass)

if cambiarPass == "si" || cambiarPass == "SI" {
    fmt.Print("Introduce nueva contraseña para la cuenta " + cuentaname + ": ")
)
    fmt.Scanf("%s\n", &nuevoPassword)
} else {
    nuevoPassword = password
}

var cuentaModificada = cuenta{encriptar([]byte(cuentaNNombre), keyuser), encri
ptar([]byte(nuevoPassword), keyuser), encriptar([]byte(servicioNServi), keyuser)}
for _, obj := range listCuentasdisponbls {
    if obj.Usuario != cuentaname || obj.Servicio != servicio {
        obj = cuenta{encriptar([]byte(obj.Usuario), keyuser), encriptar([]byte
(obj.Contraseña), keyuser), encriptar([]byte(obj.Servicio), keyuser)}
        nuevaListaCuentas = append(nuevaListaCuentas, obj)
    }
}

nuevaListaCuentas = append(nuevaListaCuentas, cuentaModificada)
UsuarioConectado.Cuentas = nuevaListaCuentas
peticionActu := peticion{"actualizarCuenta", sesionUsuario.Valor, UsuarioConec
tado, nuevaListaCuentas, ""}

var petiActu = peticionToJSON(peticionActu)
var comunicacionActu = comunicacion(petiActu)
var respuestaActu = jsonToRespuesta([]byte(comunicacionActu))

if string(respuestaActu.Estado) == "Correcto" {
    println("Actualizado realizado correctamente")
} else if string(respuestaActu.Estado) == "Incorrecto" {
    println("Actualizado no realizado")
}
}

```

El método al igual que borrar le ofrece al usuario todas las cuentas que tiene disponible para modificar, en caso de que no quiera modificar una cuenta si no añadir una nueva tendrá que añadir los datos y se creará una nueva cuenta con estos datos. Esta petición se le enviará al servidor el cual actualizará el fichero de datos del cliente con los nuevos datos recibidos.

Por parte del servidor se hará un comportamiento similar al borrado. La gestión principal en la comunicación es la siguiente:

```
case "actualizarCuenta":
    if comprobarCookieValida(pet) {
        if pet.Usuario.Cuentas == nil {
            var stin = devolvercuentasUsuario(pet)
            res := respuesta{"Correcto", getCookieUsuarios(obtenerUsuarioCookie(pet)).Oreo, "string", stin}
            resp = respuestaToJSON(res)
        } else {
            var resul = actualizarcuentas(pet)
            if resul {
                res := respuesta{"Correcto", getCookieUsuarios("").Oreo, "string", []byte("Cuenta Actualizada")}
                resp = respuestaToJSON(res)
            } else {
                res := respuesta{"Incorrecto", getCookieUsuarios("").Oreo, "string", []byte("Cuenta no Actualizada")}
                resp = respuestaToJSON(res)
            }
        }
    } else {
        fmt.Print("sesion caudcada")
    }
}
```

y el método actualizar cuentas actualizará el fichero de datos del cliente de la siguiente manera:

```

func actualizarcuentas(pet peticion) bool {
    var resultado = false
    var listaUSR = JSONtoUsuariosBD(leerArchivo("usuarios.json"))

    for _, obj := range listaUSR {
        err := bcrypt.CompareHashAndPassword([]byte(obj.Name), []byte(pet.Usuario.
Name))
        if err == nil {

            deleteFile(obj.Name + ".json")
            createFile(obj.Name + ".json")
            escribirArchivoClientes(obj.Name+".json", string(cuentasToJSON(pet.Usu
ario.Cuentas)))
            resultado = true

        }
    }

    return resultado
}

```

Recibe las cuentas, las cifra y crea un nuevo archivo borrando el anterior no sin antes de escribir el fichero cifrar todos sus datos.

Parte optativa

Doble autenticación con Correo

El correo lo hemos utilizado para una doble autenticación. En el servidor, usando una cuenta generica de GMAIL; `func email(correo string, mensaje string)` Le pasamos el correo y el mensaje.

```

func (s *SmtpServer) ServerName() string {
    return s.host + ":" + s.port
}

func (mail *Mail) BuildMessage() string {
    message := ""
    message += fmt.Sprintf("From: %s\r\n", mail.senderId)
    if len(mail.toIds) > 0 {
        message += fmt.Sprintf("To: %s\r\n", strings.Join(mail.toIds, ";"))
    }

    message += fmt.Sprintf("Subject: %s\r\n", mail.subject)
    message += "\r\n" + mail.body

    return message
}

```

```

}

func email(correo string, mensaje string) {

    mail := Mail{
        mail.senderId = "sdspoleo@gmail.com"
        mail.toIds = []string{correo}
        mail.subject = "Autenticacion"
        mail.body = mensaje

    messageBody := mail.BuildMessage()

    smtpServer := SmtpServer{host: "smtp.gmail.com", port: "465"}

    log.Println(smtpServer.host)
    //build an auth
    auth := smtp.PlainAuth("", mail.senderId, "qwertyqwerty", smtpServer.host)

    // Gmail will reject connection if it's not secure
    // TLS config
    tlsconfig := &tls.Config{
        InsecureSkipVerify: true,
        ServerName:         smtpServer.host,
    }

    conn, err := tls.Dial("tcp", smtpServer.ServerName(), tlsconfig)
    if err != nil {
        log.Panic(err)
    }

    client, err := smtp.NewClient(conn, smtpServer.host)
    if err != nil {
        log.Panic(err)
    }

    // step 1: Use Auth
    if err = client.Auth(auth); err != nil {
        log.Panic(err)
    }

    // step 2: add all from and to
    if err = client.Mail(mail.senderId); err != nil {
        log.Panic(err)
    }
    for _, k := range mail.toIds {
        if err = client.Rcpt(k); err != nil {
            log.Panic(err)
        }
    }
}

```

```

}

// Data
w, err := client.Data()
if err != nil {
    log.Panic(err)
}

_, err = w.Write([]byte(messageBody))
if err != nil {
    log.Panic(err)
}

err = w.Close()
if err != nil {
    log.Panic(err)
}

client.Quit()

log.Println("Mail sent successfully")
}

```

El mensaje lo generamos con una funcion aleatoria y lo almacenamos en memoria en el servidor en una estructura donde se almacena el correo-valor.

```

//Estructura
type correoValor struct {
    Correo string `json:"correo"`
    clave  string `json:"clave"`
}

//variable en memoria temporal con las relaciones clave-valor
var listaCorreoClave []correoValor

```

Lo almacenamos así:

```

key := mrand.Intn(9999)
valor := correoValor{obj.Correo, strconv.Itoa(key)}
listaCorreoClave = append(listaCorreoClave, valor)

```

Conocimiento 0

El servidor no tiene capacidad de encriptar el usuario ni su nombre ni nada del cuerpo solo tiene la capacidad de comprobar el bcrypt. Por otro lado, a la hora de leer y escribir lo hace usando por encima

AES.

la idea es que con este método atacando al servidor no tenga muchas posibilidades el atacante de poder obtener los datos de una forma satisfactoria.

Log de la aplicación Servidor

Guardamos un log de las operaciones que ha realizado el servidor por si tenemos algun problema poder observar los pasos dados.

```
func escribirLog(data string) bool {
    leerArchivo("log.txt")
    var log = false
    t := time.Now()
    var stringHora = string(t.Format("20060102150405"))
    var linea = stringHora + ": " + data
    log = escribirArchivoClientes("log.txt", linea)

    return log
}

//Aprovechado de otro funcion el metodo de escribir
func escribirArchivoClientes(file string, data string) bool {

    var escrito = false
    if file != "" {
        f, err := os.OpenFile(file, os.O_RDWR|os.O_APPEND, 0666)
        if err != nil {
            log.Fatal(err)
        } else {
            _, error := f.WriteString(data + "\n")
            if error != nil {
                log.Fatal(error)
            }

            escrito = true

            f.Sync()
            f.Close()
        }
    }

    return escrito
}
```

Para leerlo usamos el un metodo espfico:

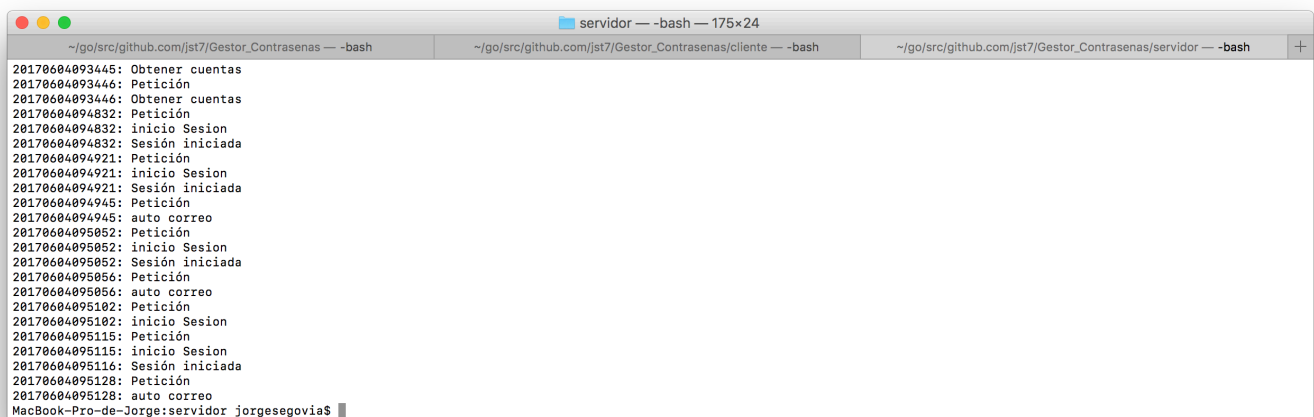

```
func leerArchivoLog(readfile string) string {

    var leer = true
    dat, err := ioutil.ReadFile(readfile)
    if err != nil {
        if readfile == "log.txt" {
            createFile("log.txt")
            leer = false
        } else {
            panic(err)
        }
    }
    var log = string(dat)
    var lineas = strings.Split(log, "\n")
    var res string

    for i := range lineas {
        if leer && lineas[i] != "" {
            res = res + "\n" + desenscriptar(lineas[i], keyEncripArch)
        }
    }

    return res
}
```

Nos proporciona la información que hayamos ido recogiendo. Resultado similar a esto:



```
servidor -- -bash -- 175x24
~/go/src/github.com/jst7/Gestor_Contrasenas -- -bash
~/go/src/github.com/jst7/Gestor_Contrasenas/cliente -- -bash
~/go/src/github.com/jst7/Gestor_Contrasenas/servidor -- -bash
20170604093445: Obtener cuentas
20170604093446: Petición
20170604093446: Obtener cuentas
20170604094832: Petición
20170604094832: inicio Sesión
20170604094832: Sesión iniciada
20170604094921: Petición
20170604094921: inicio Sesión
20170604094921: Sesión iniciada
20170604094945: Petición
20170604094945: auto correo
20170604095052: Petición
20170604095052: inicio Sesión
20170604095052: Sesión iniciada
20170604095056: Petición
20170604095056: auto correo
20170604095102: Petición
20170604095102: inicio Sesión
20170604095115: Petición
20170604095115: inicio Sesión
20170604095116: Sesión iniciada
20170604095128: Petición
20170604095128: auto correo
MacBook-Pro-de-Jorge:servidor jorgesegovia$
```

Información adicional Usuarios

Como parte adicional, se ha realizado guardar información adicional de los usuarios.

Como comentamos anteriormente la lógica de nuestra aplicación guarda a los usuarios disponibles en un archivo usuarios.json, este archivo almacena el usuario y la contraseña de este y lo utiliza para comprobar

si el login es correcto. Como parte opcional se ha añadido más información de este usuario en este caso hemos añadido el correo, este correo como se ha explicado en otros puntos se utiliza para comprobar el login del usuario. Toda esta información se guarda cifrada y el archivo se cifra también haciendo de esta información un doble cifrado.

Puesta en marcha

Para poner en marcha tenemos que tener los directorios:

- Cliente/
 - /cliente.go
- Servidor/
 - /server.crt
 - /server.key
 - /servidor.go
 - /usuarios.json
 - log.txt

Compilamos:

```
go build cliente.go go build servidor.go
```

Ejecutamos:

```
sudo ./cliente sudo ./servidor
```



A macOS terminal window titled "servidor — servidor ◀ sudo — 91x24". The address bar shows the path "~/go/src/github.com/jst7/Gestor_Contrasenas/servidor" and the current directory "servidor ◀ sudo". The terminal content shows the command "sudo ./servidor" being executed. The program outputs a menu with two options: "1. Comprobar log" and "2. Arrancar servidor". Option "2" is selected, followed by a separator line and the message "ARRANCADO EL SERVIDOR". A cursor is visible on the line following the message.

```
MacBook-Pro-de-Jorge:servidor jorgesegovia$ sudo ./servidor
1. Comprobar log
2. Arrancar servidor
2
-----
ARRANCADO EL SERVIDOR
-----
█
```



A terminal window titled "cliente — cliente ◀ sudo — 80x24" is shown. It contains two tabs: "...r_Contrasenas/servidor — servidor ◀ sudo" and "...stor_Contrasenas/cliente — cliente ◀ sudo". The terminal output shows the user running "go build cliente.go" and "sudo ./cliente". The program then displays a welcome message and a menu with three options: "1. Crear cuenta nueva", "2. Recuperar datos", and "3. Salir". A cursor is visible on the line following the menu.

```
cliente — cliente ◀ sudo — 80x24
...r_Contrasenas/servidor — servidor ◀ sudo  ...stor_Contrasenas/cliente — cliente ◀ sudo
[MacBook-Pro-de-Jorge:cliente jorgesegovia$ go build cliente.go
[MacBook-Pro-de-Jorge:cliente jorgesegovia$ sudo ./cliente

-----
Bienvenido a sus Gestor de Contraseñas
-----
1. Crear cuenta nueva
2. Recuperar datos
3. Salir
█
```

Prueba de ejecución real