Міністерство освіти і науки України
Національний Технічний Університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

Лабораторна работа №3
з дисципліни
«Методи реалізації криптографічних
механізмів»
Тема:
« Реалізація Web-сервісу електронного цифрового підпису »

Виконав
Студент групи ФІ-22мн
Русєв Денис
Перевірив
Кудін А.М.

Київ-2023

Для реалізації Web-сервісу ЕЦП було обрано мову програмування GO та пакет Echo, який предоставляє усі необхідні функції та інструменти для комфортного написання API.

Перше с чого необхідно почати це створити “router”(класс, що автомотично обраблює усі вхідні та вихідні HTTP запити) та описати контроллери, які ми будемо використовувати. В нашому випадке це лиша два контроллера.

```go
func NewApi() *Api {
    s := &Api{echo.New()}
    s.configureRouter()
    return s
}

func (s *Api) configureRouter() {
    s.Use(middleware.CORS())
    s.POST("/sign", s.SignData)
    s.POST("/verify", s.VerifyData)
}
```

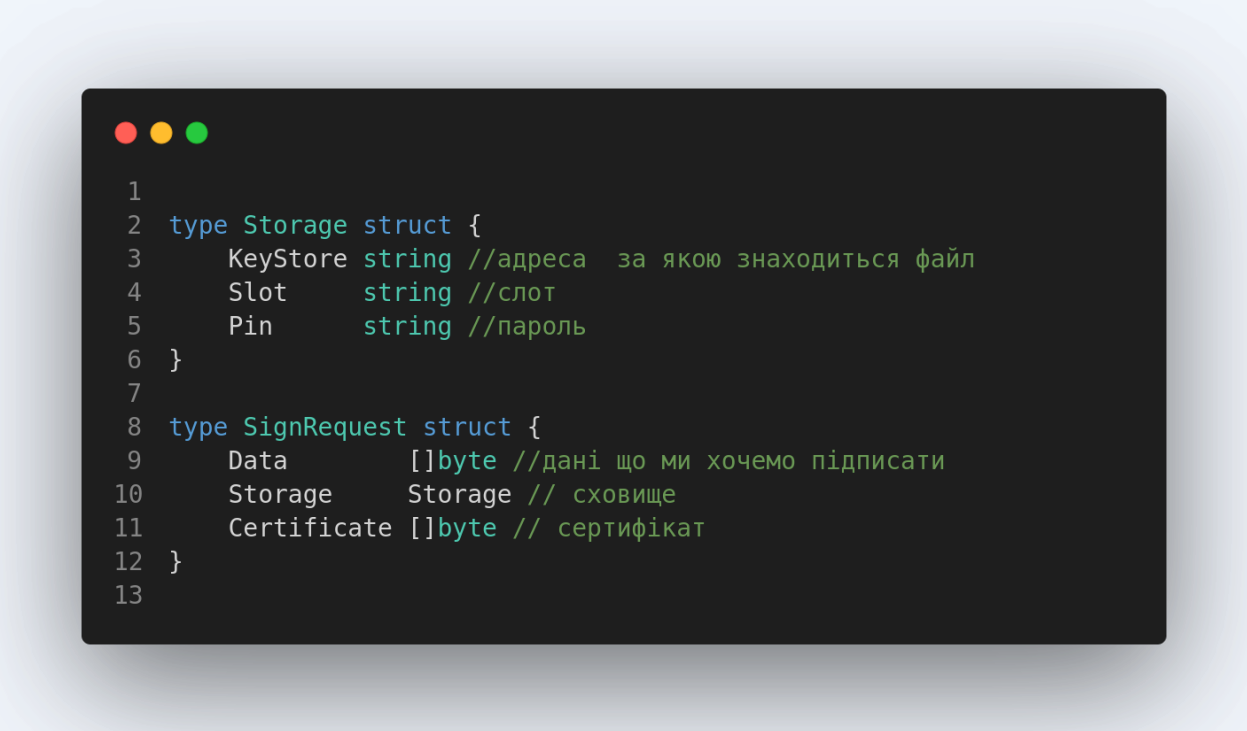
API запускається за допомогою функції Start(addr string), с аргументом addr, що позначаю за якою адресою можно звернутися до API.

```go
NewApi().Start("localhost:9999")
```

Опис та реалізація контролерів

SignData

Метод контролеру POST, адреса /sign.Призначенням цього контролеру є підпис даних. Для підпису даних необідно мати приватний ключ, сертифікат та данні, які необхідно підписати. Частіше за все приватні ключи зберігаються на захищених носіях і мною було обрано, що мій сервіс ЕЦП буде підтримувати сховища, що підтримують стандарт PKCS#12. Для того щоб отримати дані такого сховища необхідно знати мати 3 речі саме сховище у вигляді файлу, пароль до нього та слот на якому знаходиться необхідний приватний ключ. Виходячі з того, що необхідно - було створенно такий запит:



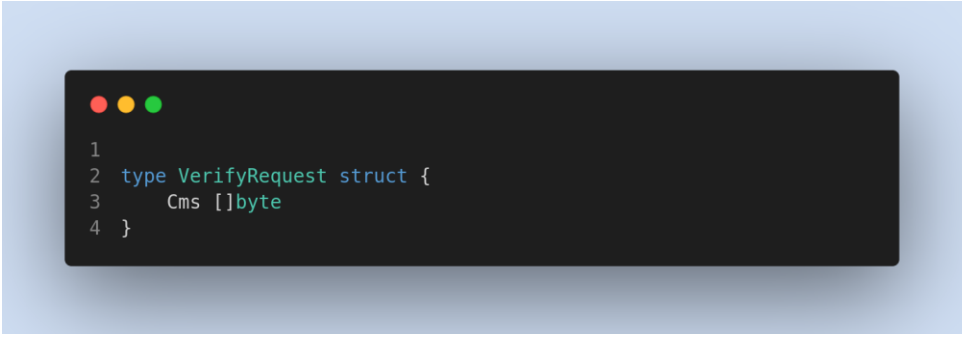
```
1
2  type Storage struct {
3      KeyStore string //адреса  за якою знаходиться файл
4      Slot     string //слот
5      Pin      string //пароль
6  }
7
8  type SignRequest struct {
9      Data        []byte //дані що ми хочемо підписати
10     Storage     Storage // сховище
11     Certificate []byte // сертифікат
12 }
13
```

Отримавши запит, починається процес підпису. В-першу чергу ми дістаємо приватний ключ за сховища, далі ми задаємо геш-функцію у нашому випадку це буде SHA-256(2.16.840.1.101.3.4.2.1), створюємо CMS, прикладаємо до нього данні, які ми будемо підписувати, інформацію про публічний ключ та про алгоритм гешування. Тепер переходимо до другого кроку, ми обчислюємо геш на дані та підписуємо геш значення, отриманий результат ми додаємо до CMS. В кінці ми отримуємо CMS, який містить у собі підпис, дані, що були підписані, алгоритм гешування, інформацію про публічний ключ, тобто все необхідне для наступною перевірки.

```go
func Sign(cert []byte, data []byte, st Storage) ([]byte, error
) {
    fmt.Println("Getting Private key")
    pk, err := GetPrivatekeyFromCert(st, cert)
    if err != nil {
        return nil, err
    }
    fmt.Println("set digest alg")
    err = pk.SetDigestAlg("2.16.840.1.101.3.4.2.1", nil)
    if err != nil {
        return nil, err
    }
    fmt.Println("create cms")
    algs, _ := algorithms.NewInternationalAlgFactory()
    cms, err := cryptolib.NewCmsAdvanced_AttachedAlgs(true
, algs)
    if err != nil {
        return nil, err
    }
    fmt.Println("new cert")
    c, err := certificate.NewCertificate_Certblob(cert)
    if err != nil {
        return nil, err
    }
    fmt.Println("Sign Init")
    _, err = cms.AddSigner_CertPrivatekey(c, pk)
    if err != nil {
        return nil, err
    }
    fmt.Println("Sign")
    err = cms.Update(data, len(data))
    if err != nil {
        return nil, err
    }
    fmt.Println("Sign Finished")
    if err := cms.EnsureSigned(); err != nil {
        return nil, err
    }
    fmt.Println("cms encoded")
    return cms.GetEncoded()
}
```

VerifyData

Метод контролеру POST, адреса /verify.Призначенням цього контролеру є перевірка підпису. В нашому випадку для перевірки підпису нам необхідно лише CMS, яка вже містить у собі всю необхідну інформацію. Тому запит на цей контролер виглядає так:



```go

type VerifyRequest struct {
    Cms []byte
}
```

Отримавши CMS ми достаємо дані, які були підписані, обчислюму геш за домогою SHA-256, отримане значення перевіряється на відповідність підпису за допомогою відкритого ключа підписувача. Якщо перевірка підпису пройшло без будь-яких помилок, то ми отримаємо дані, що були підписані, в іншому — ми отримуємо код помилки.

```go
func Verify(cmsenc []byte) ([]byte, error) {
    algs, _ := algorithms.NewInternationalAlgFactory()

    cms, err := cryptolib.NewCmsAdvanced_DerdataSizeAlgs
(cmsenc, len(cmsenc), algs)
    if err != nil {
        return nil, err
    }

    data, err := cms.GetContent()
    if err != nil {
        return nil, err
    }

    err = cms.VerifyBegin(&certificates.CertificateFinder{})
    if err != nil {
        return nil, err
    }

    n, err := cms.GetSignerCount()
    if err != nil {
        return nil, err
    }

    if n > 1 {
        return nil, errors.New("more than 1 signer")
    }

    info, err := cms.VerifySigner(0)
    if err != nil {
        return nil, err
    }
    if info == nil {
        si, _ := cms.GetSigner(0)
        code, _ := si.GetVerificationStatus()

        return nil, errors.New("error code = " + strconv.Itoa
(code))
    }

    return data, nil
}
```