Міністерство освіти і науки, молоді та спорту України
Національний технічний університет України
“Київський політехнічний інститут”
Фізико-Технічний інститут

Лабораторна робота з Методів реалізації криптографічних механізмів

Студент групи ФІ-32мн
Карловський Володимир Олександрович

Лабораторна робота № 1. Завдання 3А

Розробка технічних вимог (із вибором або бібліотеки реалізації арифметичних операцій або бібліотеки реалізації основних криптографічних примітивів) для різних варіантів реалізації IT-систем. Вибір бібліотеки для реалізації Web-сервісу електронного цифрового підпису.

Криптографічна система має складатися з веб сторінки та cli:

Функціонал cli:

1. Отриманням ключа для підпису файлів та текстових повідомлень
2. Підпис файлу за допомогою приватного ключа і отримання сигнатури.

Функціонал веб сторінки:

1. Сторінка для перевірки підписаних файлів

Технології та бібліотеки, які будуть використовуватись для побудови системи:

1. Мова Go https://go.dev/
2. Пакети зі стандартної бібліотеки crypto, crypto/rand, crypto/rsa, crypto/sha512
3. Функції для підпису `SignPSS, VerifyPSS`

   PSS (Probabilistic Signature Scheme) - це схема підпису для цифрового підпису, яка часто використовується разом із алгоритмом RSA для забезпечення безпеки в інформаційних системах. PSS була розроблена як один із методів покращення безпеки підпису, щоб уникнути певних проблем, які можуть виникнути при використанні стандартного підпису RSA (PKCS1v15).

Лабораторна робота № 2. Завдання 3А

Розробка технічних вимог (із вибором схеми генерації ПСП та схеми управління ключами) для різних варіантів реалізацій ІТ-систем. Вибір рішень для реалізації Web-сервісу електронного цифрового підпису.

Детальний опис роботи системи:

Генерація ключа

1. Користувач, дає команду на генерацію, отримує у відповідь 2 ключі публічний і приватний
2. Генерується ключ довжини 4096 бітів, використовується джерело випадковості rand.Reader - обгортка мови над /dev/urandom - що є безпечним джерелом випадковості
3. Користувачу віддається файл публічного та приватного ключа.

Підпис

1. Користувач передає в команду на генерацію файл/текст додає приватний ключ
2. Сервер хешує контен за допомогою `sha512`
3. Підписує хеш `SignPSS`
4. Віддає файл сигнатури користувичу

Перевірка

1. Користувач завантажує файл/текст додає публічний ключ і сигнатуру
2. Сервер перевіряє підпис

Лабораторна робота № 3. Завдання 3А

Завдання виконано в коді

Лабораторна робота № 4. Завдання 3А

Стандарт	Опис	Чому не актуально
PKCS #2	**Не активний**	**Очевидно, бо стандарт не використовується**
PKCS #3	***Діффі-Хелман***	**Проект не використовує схему *Діффі-Хелмана***
PKCS #4	**Не активний**	**Очевидно, бо стандарт не використовується**
PKCS #5	**Стандарт шифрування на основі паролю**	**Проект не використовує шифрування на основі паролю**
PKCS #6	**Розширений сертифікат**	**В проекті нема розширеного сертифікату**
PKCS #7	**Повідомлення**	**В проекті немає обміну по повідомленнями**
PKCS #8	**Про синтаксис закритого ключа**	**Тема актуальна для роботи, але припускається, що проект є в мережі центрів сертифікації**
PKCS #9	**Про додаткові атрибути**	**У підпису немає додаткових атрибутів**
PKCS #10	**Запит на сертифікацію**	**Проект не є центром сертифікації**
PKCS #12	**Стандарт синтаксису обміну персональною інформацією**	**Тема актуальна для роботи, але припускається, що проект є в мережі центрів сертифікації**
PKCS #13	**Еліптичні криві**	**В проекті нема еліптичних кривих**
PKCS #14	**Генерація псевдовипадкових чисел**	**Тема актуальна для роботи, але в PKCS #1 вже є стандартизований алгоритм**
PKCS #15	**Стандартний формат інформації для окремих криптографічних пристроїв**	**Проект не програмує окремі криптографічні пристрої**

Тому актуальним стандартом для проекту може бути **PKCS #1 - стандарт RSA.** Тому далі розберемо актуальний стандарт https://datatracker.ietf.org/doc/html/rfc3447

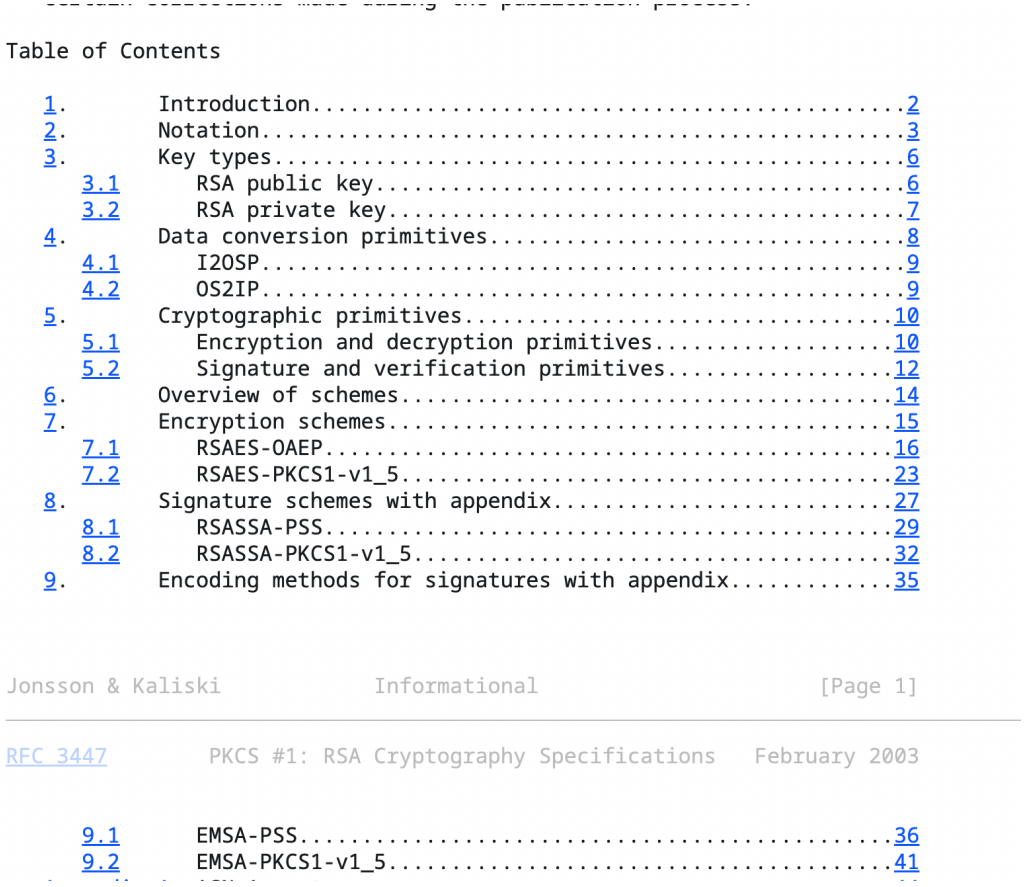

Table of Contents

В стандарті є 2 основні частини: 1 - алгоритми, 2 - схеми застосування

1 - алгоритми

В проекті використовується бібліотеці https://pkg.go.dev/crypto/rsa і відповідно до документації, алгоритм імплементує стандарт

> Package rsa implements RSA encryption as specified in PKCS #1 and RFC 8017.

2 - схеми застосування

RSASSA-PSS is recommended for eventual adoption in new applications.

З стандарту:

Наведені тут схеми підпису з додатком відповідають загальній моделі, подібній до тієї, яка використовується в IEEE Std 1363-2000, поєднуючи примітиви підпису та перевірки з методом кодування для підписів. Операції генерації підпису застосовують операцію кодування повідомлення до повідомлення, щоб створити закодоване повідомлення, яке потім перетворюється на цілочисельний представник повідомлення. Для створення підпису до представника повідомлення застосовується примітив підпису. Змінюючи це, операції перевірки підпису застосовують примітив перевірки підпису до підпису для відновлення представника повідомлення, який потім перетворюється на повідомлення, закодоване рядком октетів. Операція перевірки застосовується до повідомлення та закодованого повідомлення, щоб визначити їх узгодженість.

```
rsa.VerifyPSS(key, crypto.SHA512, contentHashSum, signature,  opts: nil)
```

Додатково, підтвердження того, що sha512 - можна використовувати згідно зі стандартом

```
a discussion of supported hash functions, see Appendix B.1.

   DigestAlgorithm ::=
       AlgorithmIdentifier { {PKCS1-v1-5DigestAlgorithms} }

   PKCS1-v1-5DigestAlgorithms    ALGORITHM-IDENTIFIER ::= {
       { OID id-md2 PARAMETERS NULL    }|
       { OID id-md5 PARAMETERS NULL    }|
       { OID id-sha1 PARAMETERS NULL   }|
       { OID id-sha256 PARAMETERS NULL }|
       { OID id-sha384 PARAMETERS NULL }|
       { OID id-sha512 PARAMETERS NULL }
   }
```

PKCS #11

https://datatracker.ietf.org/doc/html/rfc7512#page-2

https://pkg.go.dev/github.com/miekg/pkcs11

Слід зазначити що, цей стандарт є великим і стандартизує багато криптографічних сервісів, наприклад центр сертифікації і т.д.

В нашому проекті наявний лише підпис тому далі я буду розписувати вимоги відповідно до нашого проекту. Попередньо я розбив вимоги на 3 групи:

1. Sign Flow (Схема взаємодії)
2. Форматування даних
3. Кодування даних

Sign Flow (Схема взаємодії)

В стандарті всі операції починаються з виклику функції OpenSession()

```
func (c *Ctx) OpenSession(slotID uint, flags uint) (SessionHandle, error)
```

Для проекту з підпису, це має створити для нас контекст по якому ми далі будемо взаємодіяти.

Також слід зазначити, что процес генерації ключів який в нас є також не ок відповідно до формату, ця схема має працювати наступним чином: людині видає підпис якийсь центр сертифікації (наприклад податкова), і ми можемо просто валідувати підпис, який ставить людина в себе локально. Ми це можемо робити завдяки загальному стандарту форматування та кодування.

Форматування даних

В проекти формат api, це вигаданий формат, який був найзручнішим для виконання завдання (JSON). В стандарті формати повідомлення

задаються типізовано завняки ASN1 (це формат опису контракту взаємодії, схоже за сутью на protobuf для gRPC). Слід також зазначити що це не PKCS#11, а посилання з нього.

Тому щоб проект відповідав стандарту треба перейти на використання

Table 1: Mapping between URI Path Component Attributes
and PKCS #11 Specification Names

The following table presents mapping between the "type" attribute values and corresponding PKCS #11 object classes.

Attribute value	PKCS #11 object class
cert	CKO_CERTIFICATE
data	CKO_DATA
private	CKO_PRIVATE_KEY
public	CKO_PUBLIC_KEY
secret-key	CKO_SECRET_KEY

Table 2: Mapping between the "type" Attribute
and PKCS #11 Object Classes

Формат url-ів також задається форматом

```
pk11-URI             = "pkcs11:" pk11-path [ "?" pk11-query ]
; Path component and its attributes.  Path may be empty.
pk11-path            = [ pk11-pattr *(";" pk11-pattr) ]
pk11-pattr           = pk11-token / pk11-manuf / pk11-serial /
                       pk11-model / pk11-lib-manuf /
                       pk11-lib-ver / pk11-lib-desc /
                       pk11-object / pk11-type / pk11-id /
                       pk11-slot-desc / pk11-slot-manuf /
                       pk11-slot-id / pk11-v-pattr
; Query component and its attributes.  Query may be empty.
pk11-qattr           = pk11-pin-source / pk11-pin-value /
                       pk11-module-name / pk11-module-path /
                       pk11-v-qattr
pk11-query           = [ pk11-qattr *("&" pk11-qattr) ]
; Section 2.2 of [RFC3986] mandates all potentially reserved characters
; that do not conflict with actual delimiters of the URI do not have
; to be percent-encoded.
pk11-res-avail       = ":" / "[" / "]" / "@" / "!" / "$" /
                       "'" / "(" / ")" / "*" / "+" / "," / "="
pk11-path-res-avail  = pk11-res-avail / "&"
; "/" and "?" in the query component MAY be unencoded but "&" MUST
; be encoded since it functions as a delimiter within the component.
pk11-query-res-avail = pk11-res-avail / "/" / "?" / "|"
pk11-pchar           = unreserved / pk11-path-res-avail / pct-encoded
pk11-qchar           = unreserved / pk11-query-res-avail / pct-encoded
pk11-token           = "token" "=" *pk11-pchar
pk11-manuf           = "manufacturer" "=" *pk11-pchar
pk11-serial          = "serial" "=" *pk11-pchar
pk11-model           = "model" "=" *pk11-pchar
pk11-lib-manuf       = "library-manufacturer" "=" *pk11-pchar
pk11-lib-desc        = "library-description" "=" *pk11-pchar
pk11-lib-ver         = "library-version" "=" 1*DIGIT [ "." 1*DIGIT ]
pk11-object          = "object" "=" *pk11-pchar
pk11-type            = "type" "=" ( "public" / "private" / "cert" /
                       "secret-key" / "data" )
pk11-id              = "id" "=" *pk11-pchar
pk11-slot-manuf      = "slot-manufacturer" "=" *pk11-pchar
pk11-slot-desc       = "slot-description" "=" *pk11-pchar
pk11-slot-id         = "slot-id" "=" 1*DIGIT
pk11-pin-source      = "pin-source" "=" *pk11-qchar
pk11-pin-value       = "pin-value" "=" *pk11-qchar
pk11-module-name     = "module-name" "=" *pk11-qchar
pk11-module-path     = "module-path" "=" *pk11-qchar
pk11-v-attr-nm-char  = ALPHA / DIGIT / "-" / "_"
; The permitted value of a vendor-specific attribute is based on
; whether the attribute is used in the path or in the query.
```

```
pkcs11:token=The%20Software%20PKCS%2311%20Softtoken;
       manufacturer=Snake%20Oil,%20Inc.;
       model=1.0;
       object=my-certificate;
       type=cert;
       id=%69%95%3E%5C%F4%BD%EC%91;
       serial=
       ?pin-source=file:/etc/token_pin
```

Кодування даних

Стандарт потребує використання UTF-8

> composed entirely of textual data and therefore SHOULD all first be encoded as octets according to the UTF-8 character encoding

Також всі поля (priv key, pub key etc) мають кодуватись відповідно до формату, це описується не в PKCS#11, там наводяться лише посилання