**ЛАБОРАТОРНА РОБОТА № 4**

Припустимо, що деякий веб-сайт містить великий відео файл F, який можна завантажувати. Браузери, які завантажують цей файл повинні бути переконані, що файл є автентичним (цілісним) перед тим як його демонструвати користувачеві. Один підхід до вирішення цього питання є використання стійкої до колізій хеш функції. Значення хеш функції h=H(F), підписане електронним підписом, є доступним усім користувачам. Після того як браузер завантажив увесь відео файл, обчислюється значення хешу завантаженого файлу і порівнюється з h. Якщо хеші однакові, то файл F є автентичним і може бути демонстрований користувачеві.

На жаль, такий підхід передбачає, що демонструвати файл можна лише тоді, коли він увесь завантажений. Мета даного завдання побудувати систему автентифікації файлу частинами, без необхідності завантаження цілого файлу. Це дає можливість демонструвати завантажені частини без очікування поки завантажиться решта.

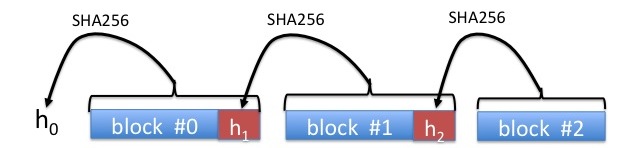
Реалізується така система автентифікації наступним чином. Веб-сайт розбиває файл на блоки 1KB (1024 байт), обчислює хеш останнього блоку, додає його до передостаннього блоку, обчислює хеш передостаннього блоку разом з хешем останнього блоку і так далі до першого блоку (див. схему).   


Рисунок 1 – Схема розбиття

Значення h0 підписується і вважається хешем цілого файлу. Саме підписане значення h0 доступне усім користувачам.

Тепер браузер завантажує перший блок разом з хешем h1, хешує його і порівнює отримане значення з h0. Якщо вони співпадають, то перший блок є автентичним і його можна демонструвати. В цей час завантажується другий блок з хешем h2. Після їх завантаження і хешування, порівнюємо обчислений хеш з h1. У випадку співпадіння другий блок також автентичний і його можна демонструвати. Цей процес продовжується поки не вичерпаються усі блоки.

В даному завданні ми будемо використовувати хеш функцію SHA256. Її слід знайти в будь-якій криптографічній бібліотеці програмної системи, якою ви користуєтесь (наприклад, PyCrypto (Python), Crypto++ (C++), тощо).

Коли додаєте хеш до блоку, додавайте його як бінарні дані, тобто 32 довільні байти. Якщо довжина файлу не кратна 1KB, то самий останній блок буде коротший, ніж 1KB, але усі інші блоки будуть в точності 1KB. Ваша задача написати програму, яка буде обчислювати h0 для заданих файлів. Для перевірки слід обчислити h0 для файлу test.mp4 і переслати його викладачу.

Для тестування вашого коду можете скористатися значенням h0 для файлу test.mp4.

h0: 03c08f4ee0b576fe319338139c045c89c3e8e9409633bea29442e21425006ea8

**ВИХІДНИЙ КОД**

import (

"os"

"log"

"fmt"

"crypto/sha256"

"container/list"

)

func OpenFile(path string) \*os.File {

file, err := os.Open(path) // For read access.

if err != nil {

log.Fatal(err)

}

return file

}

func ReadFilePart(file \*os.File, part rune) ([]byte, bool) {

data := make([]byte, part)

count, err := file.Read(data)

if err == nil {

return data[:count], false

} else {

return nil, true

}

}

func ElementAt (l \*list.List, index int) \*list.Element {

var listIndex int = 0

for e := l.Front(); e != nil; e = e.Next() {

if listIndex == index {

return e

}

listIndex++

// do something with e.Value

}

return nil

}

func GenerateBlocks(file \*os.File, partSize rune) \*list.List {

blocksList := list.New()

for {

part, isEof := ReadFilePart(file, partSize)

if !isEof {

blocksList.PushBack(part)

} else {

blocksList.PushBack([]byte{})

break

}

}

return blocksList

}

func GetSHA256(block []byte) []byte {

sha\_256 := sha256.New()

sha\_256.Write(block)

return sha\_256.Sum(nil)

}

func Solve(blocks \*list.List) []byte {

blockIndex := blocks.Len() - 3

resultBlock := ElementAt(blocks, blockIndex + 1).Value.([]byte)

lastBlockIndex := -1

for lastBlockIndex != blockIndex {

shBlock := GetSHA256(resultBlock)

mainBlock := ElementAt(blocks, blockIndex).Value.([]byte)

resultBlock = append(mainBlock, shBlock...)

blockIndex -= 1

}

return GetSHA256(resultBlock)

}

func main() {

var file \*os.File = OpenFile("d://test.mp4")

defer file.Close()

var blocks \*list.List = GenerateBlocks(file, 1024)

var result []byte = Solve(blocks)

fmt.Printf("h0:\t%x\n", result)

}

**РЕЗУЛЬТАТ ВИКОНАННЯ**

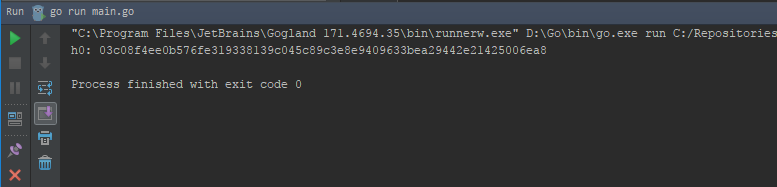


Рисунок 2 – Результат виконання