Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 4

з дисципліни «Безпека та захист програм і даних» на тему:

«КРИПТОГРАФІЯ З ВІДКРИТИМ КЛЮЧЕМ. ФУНКЦІЯ ҐЕШУВАННЯ»

Виконав:

ст. гр. КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:

доцент Тетяна ЗАЙКО

2025

1 Мета роботи

Ознайомитись з найпростішою функцією ґешування та схемами електронного цифрового підпису RSA та El Gamal.

2 Завдання до лабораторної роботи

2.1 Створити геш-образ наступного повідомлення:

«ВИДАТИ_ТРИ_МЛН_ГРН_ГОЛОВН_БУХГ_СТРІЖКО=». При створенні геш-образу використовувати кодову таблицю, кінцевий символ «=», початкове значення геш-функції H0=9 та геш-функцію, яка задана наступною формулою:

$$Hi = (Hi-1 + Mi-1)^2 \bmod 33$$

2.2 Використати публічний e=7, приватний d=3 ключі та модуль n=33 алгоритму RSA. Сформувати та перевірити цифровий підпис для повідомлення:

«ВИДАТИ_ТРИ_МЛН_ГРН_ГОЛОВН_БУХГ_СТРІЖКО=». Геш-образ необхідно отримати із завдання 2.1. Цифровий підпис надати у двійковому вигляді.

2.3 Сформувати та перевірити цифровий підпис для отриманого із завдання 4.2.1 геш-образу, але за допомогою схеми ElGamal.

2.4 Для заданого повідомлення у двійковому вигляді:

```
00101011 10001000 11100001 10111010 00000000
01101000 01000000 11000100 10101101 01101001
01110011 01000000 01001000 00110100 00000101
01001100 00001100 10101001 00111010 00011001
```

- відокремити повідомлення та цифровий підпис, який був створений за допомогою алгоритму RSA (5 біт);
- повідомлення та цифровий підпис надати у десятковому вигляді;
- за допомогою публічного ключа e=7 та модуля n=33 розшифрувати цифровий підпис;
- сформувати геш-образ відокремленого повідомлення (див. завдання 4.2.1);
- перевірити коректність діючого цифрового підпису, порівнявши результати двох попередніх пунктів.

3 Відповіді на контрольні питання

Поняття криптографії з відкритим ключем

Криптографія з відкритим ключем, або ж двохключова криптографія – це коли у нас є два математично пов’язані ключі, при цьому ми можемо зашифрувати повідомлення одним із, а розшифрувати тільки іншим

Один з пари ключів зберігається таємно та називається приватним, а інший стає публічним

Поняття геш-функції. Її властивості

Геш-функція це перетворення H, що переводить повідомлення M довільної довжини у повідомлення H(M) фіксованої довжини

По результату геш-функції не повинно бути можливо за розумний час отримати початкове повідомлення. Геш-функція також повинна завжди переводити однакові вхідні данні в однаковий хеш

У геш-функції повинна бути захищеність від колізій, коли можна знайти Mi та Mj такі, що H(Mi) = H(Mj) з ймовірністю не вище ніж завідомо задане значення

Модель взаємної недовіри

Кожен користувач генерує ключі сам собі. Генерується два ключі: відкритий та закритий. Відкритий надсилається всім учасникам, з якими буде відбуватись діалог. Закритий, або ж приватний, зберігається в секретному місці

В такій схемі хто завгодно може зашифрувати повідомлення через публічний ключ, але тільки той, хто володіє приватним може його розшифрувати

Загальна схема формування та перевірки цифрового підпису

Для початку нам потрібно мати два ключі: приватний та публічний (закритий/відкритий). Приватний ми зберігаємо у себе нікому не показуючи, а публічний можемо вільно передавати всім, хто буде перевіряти наші підписи, або ж можна виставити його в public та зробити доступним всім

Далі якась інформація підписується з використанням приватного ключа. Береться хеш інформації та через спеціальну функцію отримуємо підпис. Він може бути як числами, так і рядком

Після цього інформація разом з підписом передаються отримувачам відкритими каналами

При отриманні повідомлення отримувач також робить операцію хешування інформації, а далі через спеціальний алгоритм порівнює підпис з хешем інформації

Загальна схема спрямованого шифрування

При спрямованому шифруванні повідомлення зашифровується через відкритий ключ отримувача, а розшифрувати його може тільки сам отримувач за допомогою секретного ключа

Схема цифрового підпису RSA

Для створення цифрового підпису ми виконуємо формулу

$$C = h^d \ mod \ N$$

Тут d – приватний ключ, h – хеш, а N – модуль перетворення

Для перевірки замість приватного ключа використовується публічний:

$$h = C^e \ mod \ N$$

Тут e – публічний ключ

Якщо хеш дорівнює правій частині формули, то підпис правильний

Доказати математичну коректність алгоритму RSA

RSA базується на тому, що з поточними технологіями ми не можемо знайти прості числа при перемноженні яких отримуємо N, знаючи тільки N

При цьому нам абсолютно точно потрібно знати ці числа P та Q, які при перемноженні дають N, щоб обчислити функцію Ейлера та вивести наш приватний ключ

Наш приватний ключ базується на тому, щоб при перемноженні з публічним на полі (P-1)(Q-1) дає 1 mod (P-1)(Q-1).

Число N є дуже великим, 2000 – 4000 біт, або навіть більше. Тому, якщо P та Q обрані гарно, наприклад, вони не є однаковими та знаходяться достатньо далеко, то знайти ці два числа займе величезну кількість часу

Схема цифрового підпису ElGamal

Схема ElGamal – це криптосистема з відкритим ключем, яку засновано на складності обчислення логарифмів у скінченному полі

Система включає в себе алгоритм шифрування та алгоритм цифрового підпису

Для роботи в режимі підпису використовується хеш-функція, значення якої лежить в діапазоні (1, p-1), де p це випадкове просте число

При підписі створюється пара (r, s). Ця пара, разом з публічними полями ElGamal передається на верифікцію

При верифікації підпису виконується математична формула, яка базується на значенні хеша, r, s та інших публічних полях алгоритму

Доказати математичну коректність алгоритму ElGamal

Вся безпека цього алгоритму зводиться до того, що ми не можемо просто вирахувати значення x, секретного ключа, з формули отримання відкритого

$$y = a^x \ mod \ p$$

y – відкритий ключ

a – первісний корінь за модулем p

p – велике просте число, модуль перетворення

x – секретний ключ

З цієї формули нам досить складно знайти правильне значення x, це назиється дискретним логарифмом

Що стосується математичної коректності, то можна розглянути формули та вивести формулу для перевірки підпису

У нас є наступні формули

$$1 < k < p - 1$$
$$h = H(M)$$
$$r = a^k \ mod \ p$$
$$s \equiv (h - xr) \ k^{-1} \quad (mod \ p - 1)$$

З цього

$$sk \equiv h - xr \quad (mod \ p - 1)$$
$$h \equiv xr + ks \quad (mod \ p - 1)$$

Тепер ми можемо піднести a до степеня обох частин за модулем p використовуючи малу теорему Ферма

$$a^h \equiv a^{xr+ks} \pmod{p}$$
$$a^h \equiv (a^x)^r \cdot (a^k)^s \pmod{p}$$

І отримуємо

$$a^h \equiv y^r \cdot r^s \pmod{p}$$

Що відповідає формулі для верифікації підпису

Від чого захищає цифровий підпис? Яким чином?

Цифровий підпис захищає від зміни даних під час передачі, тобто забезпечує їх цілісність. Також цифровий підпис гарантує, що данні були підписані конкретним відправником

Гарним прикладом використання цифрового підпису є JWT, які використовуються для авторизації, наприклад, через Google. Google підписує JWT, дає його нам, далі ми його перевіряємо через публічні ключі та можемо сказати, чи це точно підписали Google, а також чи були зміненні дані в процесі.

При цьому підході, якщо інформація додатково не шифрується, не можна передавати приватну інформацію. Зазвичай інформацію передають в відкритому вигляді без додаткового шифрування

4 Текст програми

```
package lb4

import (
	"fmt"
	"io"
	"log"
	"math"
	"math/big"
)

const DEBUG = true

var lookUpTable map[rune]int = map[rune]int{
	'А': 1, 'Б': 2, 'В': 3, 'Г': 4, 'Д': 5,
	'Е': 6, 'Є': 7, 'Ж': 8, 'З': 9, 'И': 10,
	'І': 11, 'Ї': 12, 'Й': 13, 'К': 14, 'Л': 15,
	'М': 16, 'Н': 17, 'О': 18, 'П': 19, 'Р': 20,
```

```
	'С': 21, 'Т': 22, 'У': 23, 'Ф': 24, 'Х': 25,
	'Ц': 26, 'Ч': 27, 'Ш': 28, 'Щ': 29, 'Ь': 30,
	'Ю': 31, 'Я': 32, '=': 63, '_': 0,
}

// Golang math library works only with float64 :(
func powInt(x, y int) int {
	return int(math.Pow(float64(x), float64(y)))
}

func hashMessage(inputMessage string) int {
	message := []rune(inputMessage)
	hash := 9 // H0
	MOD := 33

	log.Printf("Index\tNum\tChar\tSum\tPower\tHash\n")

	for i, char := range message {
		symbol, ok := lookUpTable[char]

		if !ok {
			panic(fmt.Errorf("Symbol not found in lookUpTable! %d", symbol))
		}

		sum := (hash + symbol)
		power := sum * sum
		hash = power % MOD

		log.Printf("%d\t%d\t%s\t%d\t%d\t%d\n", i, symbol, string(char), sum, power, hash)
	}

	return hash
}

func hashMessageWithoutTable(message []int) int {
	hash := 9 // H0
	MOD := 33

	log.Printf("Index\tNum\tSum\tPower\tHash\n")

	for i, value := range message {
		sum := (hash + value)
		power := sum * sum
		hash = power % MOD
		hash = (hash + value) * power % MOD

		log.Printf("%d\t%d\t%d\t%d\t%d\n", i, value, sum, power, hash)
	}

	return hash
}

func signingRSA(h int, N, d int) int {
	S := powInt(h, d) % N

	return S
}

func verifyRSA(h int, s int, N, e int) bool {
```

```
	log.Println(powInt(s, e)%N, h)

	return powInt(s, e)%N == h
}

func openKeyElGamal(p, a, x int) int {
	return powInt(a, x) % p
}

func signingElGamal(hash int, p, a, x int) (int, int) {
	// k should be generated, but it's ok for lab work
	k := 13

	r := powInt(a, k) % p

	// Usage of math/big here because of "Euclidean modulus" in big.Mod
	pBig := big.NewInt(int64(p - 1))
	s := int(
		new(big.Int).Mod(
			new(big.Int).Mul(
				big.NewInt(int64(hash-x*r)),
				new(big.Int).ModInverse(
					big.NewInt(int64(k)),
					pBig,
				),
			),
			pBig,
		).Int64(),
	)

	if s == 0 {
		// Reference: https://en.wikipedia.org/wiki/ElGamal_signature_scheme
		panic("signingElGamal have S equal to 0, try different k")
	}

	return r, s
}

func verifyElGamal(hash int, r, s int, p, a, y int) bool {
	hBig := big.NewInt(int64(hash))
	rBig := big.NewInt(int64(r))
	sBig := big.NewInt(int64(s))
	pBig := big.NewInt(int64(p))
	aBig := big.NewInt(int64(a))
	yBig := big.NewInt(int64(y))

	first := new(big.Int).Exp(aBig, hBig, pBig)
	second := new(big.Int).Mod(
		new(big.Int).Mul(
			new(big.Int).Exp(yBig, rBig, nil),
			new(big.Int).Exp(rBig, sBig, nil),
		),
		pBig,
	)

	log.Println("Verify ElGamal", first, second)

	return first.Cmp(second) == 0
}
```

```
func Run() {
	if DEBUG {
		log.SetFlags(log.Lmsgprefix)
		log.SetPrefix("[DEBUG]: ")
	} else {
		log.SetOutput(io.Discard)
	}

	fmt.Println("Lb 4:")

	message := "ВИДАТИ_ТРИ_МЛН_ГРН_ГОЛОВН_БУХГ_СТРІЖКО="
	fmt.Println("Input message:", message)

	fmt.Println()
	hash := hashMessage(message)
	fmt.Printf("Hash is %d, in binary is %06b\n", hash, hash)

	type RSA struct {
		N int
		e int
		d int
	}

	rsa := RSA{N: 33, e: 7, d: 3}

	fmt.Println()
	signedRSA := signingRSA(hash, rsa.N, rsa.d)
	verifiedRSA := verifyRSA(hash, signedRSA, rsa.N, rsa.e)
	fmt.Printf("Signed RSA: %d. Verification is %t\n", signedRSA, verifiedRSA)

	type ElGamal struct {
		p int
		a int
		x int
		y int
	}

	elGamal := ElGamal{
		p: 41,
		a: 6,
		x: 10,
	}
	elGamal.y = openKeyElGamal(elGamal.p, elGamal.a, elGamal.x)

	fmt.Println()
	r, s := signingElGamal(hash, elGamal.p, elGamal.a, elGamal.x)
	verifiedElGamal := verifyElGamal(hash, r, s, elGamal.p, elGamal.a, elGamal.y)
	fmt.Printf("Signed ElGamal: r = %d, s = %d. Verification is %t\n", r, s, verifiedElGamal)

	fmt.Println()
	secondMessage := []int{
		0b00101011,
		0b01101000,
		0b01110011,
		0b01001100,

		0b10001000,
		0b01000000,
```

```
		0b01000000,
		0b00001100,

		0b11100001,
		0b11000100,
		0b01001000,
		0b10101001,

		0b10111010,
		0b10101101,
		0b00110100,
		0b00111010,

		0b00000000,
		0b01101001,
		0b00000101,
		0b00011001,
	}

	secondRSASignature := secondMessage[len(secondMessage)-1]
	secondMessage = secondMessage[:len(secondMessage)-1]
	fmt.Printf("Message: %+v, Signature %d\n", secondMessage, secondRSASignature)

	secondRSA := RSA{
		e: 7,
		N: 33,
	}

	isSecondVerified := verifyRSA(
		hashMessageWithoutTable(secondMessage),
		secondRSASignature,
		secondRSA.N,
		secondRSA.e,
	)

	fmt.Printf("Is second verified: %t\n", isSecondVerified)
}
```

5 Результати роботи програми

Для виконання роботи було написано програму на Golang

Спочатку виконується розрахунок хеш-образу. Результат інформації з інформацією про внутрішні процеси показаний на рисунку 1

```
C:\home\university-4\security>go run .
Lb 4:
Input message: ВИДАТИ_ТРИ_МЛН_ГРН_ГОЛОВН_БУХГ_СТРІЖКО=

[DEBUG]: Index  Num     Char    Sum     Power   Hash
[DEBUG]: 0      3       В       12      144     12
[DEBUG]: 1      10      И       22      484     22
[DEBUG]: 2      5       Д       27      729     3
[DEBUG]: 3      1       А       4       16      16
[DEBUG]: 4      22      Т       38      1444    25
[DEBUG]: 5      10      И       35      1225    4
[DEBUG]: 6      0       _       4       16      16
[DEBUG]: 7      22      Т       38      1444    25
[DEBUG]: 8      20      Р       45      2025    12
[DEBUG]: 9      10      И       22      484     22
[DEBUG]: 10     0       _       22      484     22
[DEBUG]: 11     16      М       38      1444    25
[DEBUG]: 12     15      Л       40      1600    16
[DEBUG]: 13     17      Н       33      1089    0
[DEBUG]: 14     0       _       0       0       0
[DEBUG]: 15     4       Г       4       16      16
[DEBUG]: 16     20      Р       36      1296    9
[DEBUG]: 17     17      Н       26      676     16
[DEBUG]: 18     0       _       16      256     25
[DEBUG]: 19     4       Г       29      841     16
[DEBUG]: 20     18      О       34      1156    1
[DEBUG]: 21     15      Л       16      256     25
[DEBUG]: 22     18      О       43      1849    1
[DEBUG]: 23     3       В       4       16      16
[DEBUG]: 24     17      Н       33      1089    0
[DEBUG]: 25     0       _       0       0       0
[DEBUG]: 26     2       Б       2       4       4
[DEBUG]: 27     23      У       27      729     3
[DEBUG]: 28     25      Х       28      784     25
[DEBUG]: 29     4       Г       29      841     16
[DEBUG]: 30     0       _       16      256     25
[DEBUG]: 31     21      С       46      2116    4
[DEBUG]: 32     22      Т       26      676     16
[DEBUG]: 33     20      Р       36      1296    9
[DEBUG]: 34     11      І       20      400     4
[DEBUG]: 35     8       Ж       12      144     12
[DEBUG]: 36     14      К       26      676     16
[DEBUG]: 37     18      О       34      1156    1
[DEBUG]: 38     63      =       64      4096    4
Hash is 4, in binary is 000100
```

Рисунок 1 – Знаходження хешу

Далі було створено цифровий підпис через RSA, з параметрами відповідно до завдання, та виконано його верифікацію. Це показано на рисунку 2

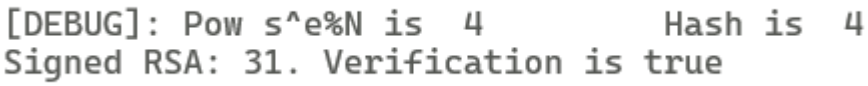

Рисунок 2 – Підпис через RSA

Наступним кроком було створено цифровий підпис використовуючи схему ElGamal. Було обрано наступні значення p = 41, a = 6, x = 10, y – розраховується в процесі, k = 13. Це показано на рисунку 3

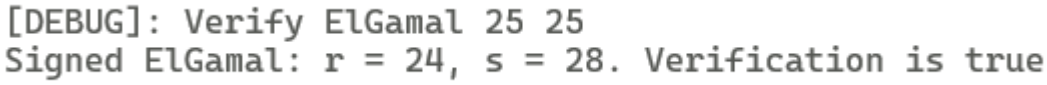

Рисунок 3 – Підпис через ElGamal

І останнім кроком було прочитано повідомлення з таблиці по стовпцях, показано в десятковому вигляді, в тому числі з цифровим підписом. Розраховано хеш-образ, але без використання таблиці, бо вона тут не підходить під значення та виконана перевірка підпису. Згідно перевірки підпис не є правильним. Це показано на рисунку 4

```
Message: [43 104 115 76 136 64 64 12 225 196 72 169 186 173 52 58 0 105 5], Signature 25
[DEBUG]: Index  Num     Sum     Power   Hash
[DEBUG]: 0      43      52      2704    17
[DEBUG]: 1      104     121     14641   0
[DEBUG]: 2      115     115     13225   2
[DEBUG]: 3      76      78      6084    0
[DEBUG]: 4      136     136     18496   23
[DEBUG]: 5      64      87      7569    21
[DEBUG]: 6      64      85      7225    8
[DEBUG]: 7      12      20      400     31
[DEBUG]: 8      225     256     65536   16
[DEBUG]: 9      196     212     44944   8
[DEBUG]: 10     72      80      6400    25
[DEBUG]: 11     169     194     37636   23
[DEBUG]: 12     186     209     43681   22
[DEBUG]: 13     173     195     38025   21
[DEBUG]: 14     52      73      5329    32
[DEBUG]: 15     58      90      8100    6
[DEBUG]: 16     0       6       36      9
[DEBUG]: 17     105     114     12996   0
[DEBUG]: 18     5       5       25      24
[DEBUG]: Pow s^e%N is  31        Hash is  24
Is second verified: false
```

Рисунок 4 – Перевірка підпису з повідомлення

Результат виконання програми без додаткової інформації показаний на рисунку 5

```
C:\home\university-4\security>go run .
Lb 4:
Input message: ВИДАТИ_ТРИ_МЛН_ГРН_ГОЛОВН_БУХГ_СТРІЖКО=

Hash is 4, in binary is 000100

Signed RSA: 31. Verification is true

Signed ElGamal: r = 24, s = 28. Verification is true

Message: [43 104 115 76 136 64 64 12 225 196 72 169 186 173 52 58 0 105 5], Signature 25
Is second verified: false
```

Рисунок 5 – Виконання програми без додаткової інформації

6 Висновки

Я ознайомився з найпростішою функцією ґешування та схемами електронного цифрового підпису RSA та El Gamal