ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ
ՀԱՄԱԼՍԱՐԱՆ

Կուրսային Աշխատանք

Թեմա՝ RSA, RLE, Affine cipher

Խումբ՝ *SS019-1*

Ուսանող՝ Սերգեյ Կոռյան

Երևան 2023

Բովանդակություն

Ներածություն

Տեղեկատվական անվտանգությունը վերաբերում է թվային տեղեկատվության և համակարգերի պաշտպանությունը չարտոնված մուտքից, օգտագործումից, բացահայտումից, խափանումից, փոփոխումից կամ ոչնչացումից: Թե՛ անձնական, թե՛ մասնագիտական միջավայրում տեխնոլոգիաների նկատմամբ անընդհատ աճող վստահության պայմաններում տեղեկատվական անվտանգության անհրաժեշտությունը դարձել է ավելի կարևոր, քան երբևէ: Տեղեկատվական անվտանգությունը ներառում է միջոցառումների լայն շրջանակ, ինչպիսիք են անվտանգության քաղաքականության, ընթացակարգերի և տեխնոլոգիաների իրականացումը, որոնք ապահովում են թվային տեղեկատվության գաղտնիությունը, ամբողջականությունը և հասանելիությունը: Այս ոլորտը կարևոր է դարձել անհատների, կազմակերպությունների և կառավարությունների համար ամբողջ աշխարհում, քանի որ նրանք ձգտում են պաշտպանել իրենց զգայուն տվյալները կիբեր սպառնալիքներից և հարձակումներից: Այս համատեքստում տեղեկատվական անվտանգության մասնագետների դերն ավելի կարևոր է դարձել, քանի որ նրանք աշխատում են պաշտպանել թվային ակտիվները անվտանգության մի շարք հնարավոր ռիսկերից:

Գաղտնագրություն

Գաղտնագրությունը պարզ տեքստի կամ տեղեկատվության փոխակերպման գործընթաց է կոդի կամ ծածկագրի, որն անընթեռնելի է որևէ մեկի համար, բացառությամբ այն վերծանելու բանալին ունեցողների: Այն կարևոր գործիք է ժամանակակից թվային դարաշրջանում, որտեղ զգայուն տվյալներն ու անձնական տեղեկությունները փոխանցվում և պահվում են էլեկտրոնային եղանակով: Գաղտնագրությունը կարող է օգտագործվել անձնական հաղորդակցությունները, ֆինանսական գործարքները և այլ գաղտնի տվյալները պաշտպանելու համար գաղտնալսումից և չարտոնված մուտքից: Կիբեր հարձակումների և տվյալների խախտումների աճող տարածվածության պայմաններում գաղտնագրումը դարձել է տվյալների անվտանգության կարևոր բաղադրիչ: Այս համատեքստում այն կենսական դեր է խաղում անձնական գաղտնիությունը պաշտպանելու և համացանցում անվտանգ հաղորդակցության ապահովման գործում:

RSA Ալգորիթմ

RSA ալգորիթմը լայնորեն օգտագործվող գաղտնագրման և գաղտնազերծման տեխնիկա է, որը մշակվել է Ռոն Ռիվեստի, Ադի Շամիրի և Լեոնարդ Ադլեմանի կողմից 1977 թվականին: Այն անվանվել է նրանց ազգանուններով և համարվում է բաց բանալիների ամենաապահով կրիպտոհամակարգերից մեկը:

RSA ալգորիթմը հիմնված է այն փաստի վրա, որ շատ դժվար է գործոնավորել երկու մեծ պարզ թվերի արտադրյալը: Գաղտնագրման գործընթացը ներառում է երկու բանալի՝ հանրային և մասնավոր բանալի: Հանրային բանալին օգտագործվում է հաղորդագրությունը գաղտնագրելու համար, մինչդեռ մասնավոր բանալին օգտագործվում է այն ապակոդավորելու համար:

Ալգորիթմի նկարագրությունը

1. Բանալիների պատրաստում. գեներացնում ենք հանրային եւ մասնավոր բանալիները: Ընտրում ենք երկու պարզ թիվ՝ p-ն եւ q-ն: Հաշվարկում ենք p-ի եւ q-ի արտադրյալը. n = p × q:
2. Հաշվում ենք էյլերի ֆունկցիան՝ Φ = (p-1) × (q-1):
3. Ընտրում ենք e թիվը (բաց էքսպոնենտը), որը համապատասխանում է հետեւյալ չափանիշներին.

   1) այն պետք է լինի պարզ,

   2) պետք է լինի Φ -ից փոքր,

   3)պետք է լինի փոխադարձ պարզ Φ –ի հետ:

{e, n} թվերի զույգը ներկայանում է որպես հանրային բանալի: A կողմն ուղարկում է այդ զույգը B կողմին, որպեսզի B կողմը գաղտնագրի իր հաղորդագրությունը:

4. Պետք է հաշվել նաեւ d թիվը, որը պետք բավարարի հետեւյալ պայմանին՝ d × e արտադրյալը Φ –ի բաժանելիս ստացված մնացորդը պետք է հավասար լինի 1-ի՝

   (d × e) % φ = 1:

{d, n} զույգը մասնավոր բանալին է: d-ի հաշվարկը կատարվում է ըստ Էվկլիդեսի ընդլայնած ալգորիթմի: Մասնավոր բանալին չպետք է փոխանցվի ինչ-որ մեկին:

Գաղտնագրում

Այժմ հաղորդագրությունը գաղտնագրելու հերթն է: Գաղտնագրվող հաղորդագրությունը նշանակվում է P-ով, այն չպետք է լինի ավելի մեծ, քան n-ը: Գաղտնագրումը կատարվում է ըստ հետեւյալ քայլերի՝ հաշվարկվում է՝

E = Pe (mod n):

E-ն գաղտնագրված տվյալն է: Այն B կողմն ուղարկում է A կողմին:

Վերծանում

A կողմը, ստանալով գաղտնագրված հաղորդագրությունը եւ ունենալով մասնավոր բանալին՝ {d, n}-ը, կատարում է վերծանման գործընթացը՝ հաշվարկում է

L=Ed(mod n),

եւ ստացված L-ը սկզբնական հաղորդագրությունն է:

Ալգորիթմի իրականացումը Python լեզվով

```python
import random


def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)


def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True


def generate_key_pair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randint(1, phi - 1)
    while gcd(e, phi) != 1:
        e = random.randint(1, phi - 1)
    d = pow(e, -1, phi)
    return (n, e), (n, d)


def encrypt(message, public_key):
    n, e = public_key
    ciphertext = [chr(pow(ord(char), e, n)) for char in message]
    return ''.join(ciphertext)


def decrypt(ciphertext, private_key):
    n, d = private_key
    plaintext = [chr(pow(ord(char), d, n)) for char in ciphertext]
    return ''.join(plaintext)
```

RLE Ալգորիթմ

Կան բազմաթիվ գործնական, առանց կորուստների սեղմման տեխնիկաներ, որոնք աշխատում են տարբեր արդյունավետությամբ, տարբեր տեսակի և ծավալների տվյալների հետ:
Դրանցից ամենահայտնի պարզ մոտեցումը շրջելի եղանակով տեղեկատվությունը սեղմելու համար Run Length Encoding (RLE) ալգորիթմն է: Այս մոտեցման էությունը տողերի կամ կրկնվող բայթերի շարքերի կամ դրանց հաջորդականությունների փոխարինումն է մեկ կոդավորման բայթով և դրանց կրկնությունների քանակի հաշվիչով:

Օրինակ այս հաջորդականությունը "aaaabaaaac" կարող է սեղմվել "4a1b4a1c" և այդպիսով ավելի քիչ ծավալ զբաղեցնել: Այս մեթոդը սովորաբար բավականին արդյունավետ է գրաֆիկական պատկերները սեղմելու համար:

Ալգորիթմի նկարագրությունը

- Նախ, պետք է սկանավորել մուտքային տվյալները՝ միաժամանակ որոնելով հաջորդական կրկնվող նիշերը
- Այնուհետև, երբ գտնենք անընդմեջ կրկնվող նիշերի հաջորդականություն, այն կփոխարինենք նիշով, որին հաջորդում է նրա հայտնվելու քանակը:
- Վերջում, սեղմված ելքային հաջորդականությունը փոփոխված մուտքային տվյալն է՝ փոխարինված կրկնվող նիշերով

Ալգորիթմի իրականացումը

RLE-ն սեղմում է մուտքագրված նիշերի կրկնվող հաջորդականությունները ավելի կարճ, ավելի հակիրճ հաջորդականության մեջ: Արդյունքում, ելքը սեղմված տող է, որը պարունակում է նիշերը և իրենց հայտնվելու քանակությունը:

Ալգորիթմի իրականացումը Python լեզվով

```
import itertools
import re


def rle2_compress(s):
    encoded = []
    for char, group in itertools.groupby(s):
        count = len(list(group))
        encoded.append(str(count) + " " + str(ord(char)) + "
")  return ''.join(encoded)


def rle2_decompress(s):
    decoded = []
    for group in re.findall("(\d+\s\d+\s)", s):
        value = group.split(" ")
        count, char = value[0], value[1]
        char = chr(int(char))
        decoded.append(char * int(count))
    return ''.join(decoded)
```

Affine cipher

Affine ծածկագիրը փոխարինող ծածկագրի տեսակ է, որտեղ պարզ տեքստի յուրաքանչյուր տառ փոխարինվում է տառով, որը այբուբենի ներքևում գտնվող դիրքերի ֆիքսված քանակ է: Գաղտնագրումը հիմնված է մաթեմատիկական բանաձևի վրա (ax + b) mod m, որտեղ a-ն և b-ն բանալիներն են, x-ը պարզ տեքստն է, և m-ը այբուբենի չափն է: Աֆինային ծածկագիրը կարող է դիտվել որպես Կեսարի ծածկագրի ընդլայնում, որը պարզ փոխարինող ծածկագիր է, որը տառերը տեղափոխում է ֆիքսված թվով դիրքերով:

Աֆինային ծածկագիրը ունի երկու բանալի՝ a և b, որտեղ a-ն պետք է ընտրվի այնպես, որ a-ն և այբուբենի չափը լինեն համակցված: Սա ապահովում է, որ գաղտնագրումը մեկ առ մեկ է և շրջելի: b ստեղնը կարող է լինել ցանկացած ամբողջ թիվ 0-ի և m-1-ի միջև, որտեղ m-ը այբուբենի չափն է:

Հաղորդագրությունը գաղտնագրելու համար, օգտագործելով affine ծածկագիրը, պարզ տեքստի յուրաքանչյուր տառ նախ փոխարկվում է իր թվային համարժեքին, որն այնուհետև բազմապատկվում է a-ով և ավելացվում b մոդուլի m-ին: Ստացված թիվը այնուհետև վերածվում է տառի՝ օգտագործելով այբուբենը: Օրինակ, եթե այբուբենը «ABCDEFGHIJKLMNOPQRSTUVWXYZ» է, իսկ ստեղները՝ a = 5 և b = 8, «A» տառը կգաղտնագրվի «I»՝ օգտագործելով (5*0 + 8) mod 26 = 8 բանաձևը, որտեղ 0-ը ներկայացնում է այբուբենի «Ա»-ի դիրքը:

Affine ծածկագիրը ապահովում է հաղորդագրությունների գաղտնագրման պարզ և արդյունավետ միջոց, սակայն այն այնքան էլ ապահով չէ ժամանակակից գաղտնագրային հարձակումներից: Աֆինային ծածկագրի թույլ կողմերից մեկն այն է, որ այն խոցելի է հաճախականության վերլուծության համար, որտեղ հարձակվողը վերլուծում է գաղտնագրման

տառերի հաճախականությունը՝ ստեղները պարզելու համար: Մեկ այլ թույլ կողմն այն է, որ նույն բանալին կարող է օգտագործվել հաղորդագրությունների գաղտնագրման և գաղտնազերծման համար, ինչը նշանակում է, որ եթե բանալին վտանգված է, այդ բանալիով գաղտնագրված բոլոր հաղորդագրությունները կարող են վերծանվել:

Չնայած իր թույլ կողմերին, affine ծածկագիրը մնում է օգտակար գործիք կրթական նպատակներով և պարզ գաղտնագրման առաջադրանքների համար, որտեղ անվտանգությունը մեծ խնդիր չէ: Այն նաև օգտագործվում է որպես շինանյութ ավելի բարդ գաղտնագրային համակարգերում, ինչպիսիք են Hill ծածկագիրը և RSA ալգորիթմը:

Ալգորիթմի իրականացումը Python լեզվով

```python
def affine_encrypt(text, a, b):
    result = ''
    for char in text:
        if char.isalpha():
            char_index = ord(char.lower()) - 97
            encrypted_index = (a * char_index + b) % 26
            result += chr(encrypted_index + 97)
        else:
            result += char
    return result


def affine_decrypt(text, a, b):
    result = ''
    a_inverse = pow(a, -1, 26)
    for char in text:
        if char.isalpha():
            char_index = ord(char.lower()) - 97
            decrypted_index = (a_inverse * (char_index - b)) % 26
            result += chr(decrypted_index + 97)
        else:
            result += char
    return result
```

Ծրագրի նկարագրությունը

Այս կոդը ցուցադրում է Python-ում ներդրված client-server ճարտարապետություն: Հաճախորդը կապ է հաստատում սերվերի հետ՝ օգտագործելով socket-ներ: Սերվերը ստեղծում է public-private բանալիների զույգ RSA ալգորիթմի միջոցով, որն այնուհետև կիսվում է հաճախորդի հետ: Հաճախորդը ընտրում է մականուն և ուղարկում այն սերվերին:

Կոդն օգտագործում է գաղտնագրման երեք ալգորիթմ՝ Affine Cipher, Run-Length Encoding (RLE) և RSA: Երբ հաճախորդը հաղորդագրություն է ուղարկում, այն ենթարկվում է կոդավորման՝ օգտագործելով Affine Cipher և RLE ալգորիթմները: Կոդավորված հաղորդագրությունը հետագայում գաղտնագրվում է RSA-ի կողմից ստեղծված սերվերի հանրային բանալին օգտագործելով: Սերվերը ստանում է գաղտնագրված հաղորդագրությունը, վերծանում է այն՝ օգտագործելով իր անձնական բանալին և հեռարձակում վերծանված հաղորդագրությունը բոլոր միացված հաճախորդներին:

Սերվերը միաժամանակ մշակում է բազմաթիվ հաճախորդի միացումներ՝ օգտագործելով թրեդներ: Այն պահպանում է միացված հաճախորդների ցուցակը, նրանց հետ կապված մականունները և դրանց համապատասխան հանրային բանալիները: Սերվերը ստանում է հաղորդագրություններ հաճախորդներից, վերծանում է դրանք և հեռարձակում բոլոր հաճախորդներին, բացառությամբ ուղարկողի:

Ընդհանուր առմամբ, այս կոդը ցույց է տալիս ապահով հաղորդակցությունը հաճախորդի և սերվերի միջև՝ օգտագործելով գաղտնագրման ալգորիթմները և client-server ճարտարապետությունը:

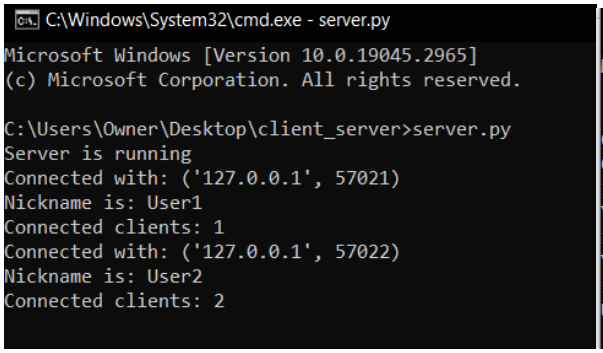
```python
import socket
import threading

import custom_rsa as rsa

host = "127.0.0.1"
port = 55555
FORMAT = "utf-8"


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
print("Server is running")

p, q = 13, 5
public_key, private_key = rsa.generate_key_pair(p, q)


credentials = {}
nicknames = {}
clients = []


def broadcast(message, sender):
    for client, client_public_key in credentials.items():
        if client != sender:
            encrypted_message = rsa.encrypt(message, client_public_key)
            client.send(encrypted_message.encode(FORMAT))
            client.send(nicknames[sender].encode(FORMAT))


def handle(client):
    while True:
        try:
            m = client.recv(2048).decode(FORMAT)
            message = rsa.decrypt(m, private_key)
            broadcast(message, client)
        except:
            clients.remove(client)
            client.close()
            del nicknames[client]
            del credentials[client]
            break


def receive():
    while True:
        client, address = server.accept()
        clients.append(client)
        print(f"Connected with: {str(address)}")

        client.send(f"{public_key}".encode(FORMAT))
        client_public_key = eval(client.recv(2048).decode(FORMAT))

        nickname = client.recv(2048).decode(FORMAT)
        print(f"Nickname is: {nickname}")

        print(f"Connected clients: {len(clients)}")
        nicknames[client] = nickname
        credentials[client] = client_public_key

        thread = threading.Thread(target=handle, args=(client,))
        thread.start()


receive_thread = threading.Thread(target=receive)
receive_thread.start()
```

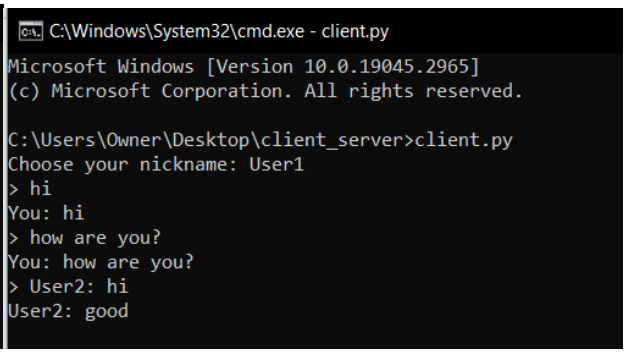
server.py

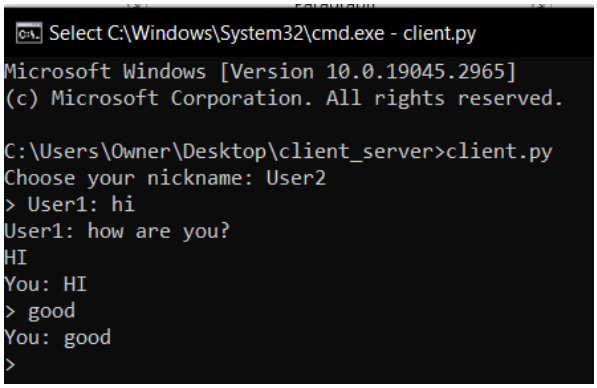
```python
import socket
import threading

import custom_rsa as rsa
from affine_cipher import affine_encrypt, affine_decrypt
from rle import rle2_compress, rle2_decompress

FORMAT = "utf-8"

p, q = 19, 11
public_key, private_key = rsa.generate_key_pair(p, q)


client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("127.0.0.1", 55555))

SERVER_PUBLIC_KEY = eval(client.recv(2048).decode(FORMAT))
client.send(f"{public_key}".encode(FORMAT))

nickname = input("Choose your nickname: ")
client.send(nickname.encode(FORMAT))


def receive():
    while True:
        message = client.recv(2048).decode(FORMAT)
        if message:
            decrypted_message = rsa.decrypt(message, private_key)
            rle_message_decrypt = rle2_decompress(decrypted_message)
            affine_message_decrypt = affine_decrypt(rle_message_decrypt, 5,
8)
            print(f"{client.recv(2048).decode(FORMAT)}:
{affine_message_decrypt}")


def write():
    while True:
        message = input("> ")
        if not message:
            break
        affine_message_encrypt = affine_encrypt(message, 5, 8)
        rle_message_encrypt = rle2_compress(affine_message_encrypt)
        encrypted_message = rsa.encrypt(rle_message_encrypt,
SERVER_PUBLIC_KEY)
        client.send(encrypted_message.encode(FORMAT))
        print(f"You: {message}")


receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write)
write_thread.start()
```

client.py

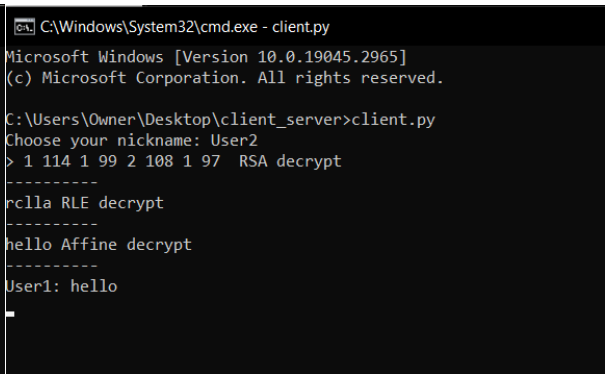
Ծրագրի աշխատանքը







Ալգորիթմների կատարումը



Եզրակացություն

Ընդհանուր առմամբ, այս նախագիծը ընդգծում է գաղտնագրման ամուր տեխնիկայի և հաճախորդ-սերվեր ճարտարապետության ներդրման նշանակությունը՝ անվտանգ հաղորդակցություն ապահովելու համար: Այն ցույց է տալիս գաղտնագրման ալգորիթմների կիրառման կարևորությունը՝ պաշտպանելու զգայուն տեղեկատվությունը, միաժամանակ հաճախորդների և սերվերների միջև տվյալների արդյունավետ փոխանակման հնարավորություն: Այս տարրերը համադրելով՝ կոդը հիմք է տալիս անվտանգ և հուսալի կապի համակարգեր կառուցելու համար: