**Toteutusdokumentti:**

Tämä RSA-salauksen totetuttava ohjelma koostuu kahdesta osasta, käyttöliittymästä (main.py) sekä toiminnallisuudesta (utilities.py). Tämän lisäksi on testitiedosto utilities_test.py.

Ohjelman funktiot ovat seuraavanlaiset:

**sieve_of_eratosthenes(limit)**: Sieve of Eratosthenes (Eratostheneen seula) on tapa generoida lista, jossa on annettu määrä (limit) pienimpiä alkulukuja. Alkuluvut generoidaan kirjoittamalla lista kaikista luonnollisista luvuista alkaen kakkosesta ja päättyen johonkin valittuun suurimpaan lukuun. Seula toimii niin että ensin poistetaan listasta kaikki luvun 2 monikerrat (4, 6, 8 jne.), jolloin listan seuraava jäljellä oleva luku on alkuluku. Sen jälkeen poistetaan listasta kaikki ne luvut, jotka ovat sekä edellisessä vaiheessa löydettyä alkulukua suurempia että sen monikertoja. Toistetaan edellisiä vaiheita, kunnes listan seuraava jäljellä oleva luku on suurempi kuin listan suurimman luvun neliöjuuri. Silloin listassa on jäljellä vain alkulukuja.

**def miller_rabin(n, k=8):** Miller-Rabinin alkulukutarkistus toimii seuraavanlaisesti:

Kun testataan, onko luku n alkuluku:

- Kirjoitetaan n-1 muodossa 2^r*d, eli jaetaan n-1 kahdella niin monta kertaa kun mahdollista.
- Esimerkkinä jos n = 561, n-1 = 560
- 560/2=280 -> 280/2=140 -> 140/2=70 -> 70/2=35
- n jaettiin kahteen 4 kertaa, eli r=4, d=35 (jakojäännös)
- Valitaan satunnaisesti jokin luku a väliltä [2,n−2]
- Lasketaan x= a^d mod n (esimerkissä x=2^35 mod 561 = 263)
- Jos tulos on x=1 tai x=n-1, luku on todennäköisesti alkuluku.
- Muuten toistetaan seuraavasti:
    - Neliöidään x mod n korkeintaan r-1 kertaa
    - Jos jokin tuloksista on n-1, luku on todennäköisesti alkuluku.
    - Muuten n ei ole alkuluku.

**def generate_prime(keysize):** Luo annetun avainkoon mukaisen avaimen (tässä projektissa 2048 bittiä).

Ensin kutsutaan **sieve_of_eratosthenes()** ja luodaan 500 pienimmän alkuluvun lista. Generoidaan randomilla vaaditun pituinen numero, joka tarkistetaan luotua listaa vasten. Jos generoitu numero ei ole jaollinen milläkään listan numeroilla, tarkistetaan luku vielä ajamalla **miller_rabin()**. Jos tämäkin tarkistus menee läpi, hyväksytään numero avaimen käyttöön.

**def generate_keys(keysize):** Luo public- ja private-avaimet. Avaimen kokoon suhteutettuna luodaan kaksi alkulukua, jotka ovat puolet avaimen koosta. Näistä alkuluvuista p ja q lasketaan pq=n, jota käytetään modulona sekä public- että private-avaimessa.

Lasketaan phi, joka on pienin yhteinen jaettava -> phi = lcm(prime1-1,prime2-1)

Käytetään e = 65537 (2^16+1), joka on yleisin käytetty e:n arvo. Tämä arvo julkaistaan osana julkista avainta.

Lasketaan Euleriin totient-funktio ϕ(n): phi = (prime1 - 1) * (prime2 - 1)

Lasketaan yksityinen avain d, joka on e:n inverse modulo ϕ(n):n suhteen: secret = mod_inverse(public, phi)

Lopulta palautetaan avainpari ((public, n), (secret, n)).

**mod_inverse(a, m):** Laskee luvun a modulo-inversin modulo m käyttäen laajennettua Eukleideen algoritmia. Käytetään RSA:n yksityisen avaimen laskemiseen. Tähän olisi voinut käyttää Sumpy:a, mutta tein tästä apufunktion jotta vertaisarvioijat voisivat ajaa ohjelman suoraan.

**string_to_int(message: str) -> int**: Muuntaa merkkijonon kokonaisluvuksi (str -> bytes → int). Tarvitaan ennen salauksen suorittamista.

**int_to_string(message_int: int) -> str:** Muuntaa kokonaisluvun takaisin merkkijonoksi (int → bytes → str). Tarvitaan salauksen purkamisen jälkeen.

**encrypt(message, public_key):** RSA-salaus: muuntaa viestin merkkijonosta kokonaisluvuksi string_to_int()-funktiolla ja salaa sen julkisella avaimella e.

return pow(message_int, e, n) -> ciphertext = message_int^e mod n

**decrypt(ciphertext, private_key):** RSA-purku: purkaa salatun viestin yksityisellä avaimella d ja muuttaa kokonaisluvun takaisin merkkijonoksi int_to_string()-funktiolla.

message_int = pow(ciphertext, d, n) -> message_int = ciphertext^d mod n

## Aika- ja tilavaativuudet:

Projektin aika- ja tilavaativuudet voi laskea arviolta encrypt- ja decrypt-funktioiden pow() lausekkeesta.

**Aikavaativuus:**

sieve_of_eratosthenes(): koska listan koko on tässä projektissa aina vakio, funktio toimii vakioajassa.

miller_rabin(): aikavaativuus = O($k\ n^3$) jossa k on kierrosten määrä ja n on annetun luvun pituus, mutta koska tässä projektissa molemmat asetetaan vakioiksi, funktio toimii vakioajassa.

prime_generation(): Alkulukulauseella voi laskea vaadittavien iteraatioiden aikavaativuuden O(n*ln2), jossa n on annetun luvun pituus. Tämän projektin kontekstissa alkulukujen generointi toimii kuitenkin vakioajassa, koska sieve_of_eratosthenes() ja miller_rabin() toimivat myös tämän projektin kontekstissa vakioajassa ja näin ollen iteraatioiden määrä on keskimäärin samanlainen aina alkulukua generoidessa.

generate_keys(): toimii muuten vakioajassa paitsi mod_inverse():n osalta, jonka aikavaativuuden laskeminen ei ole aivan triviaalia, koska se riippuu enemmän funktion matemaattisista ominaisuuksista kun annetuista syötteistä.

string_to_int(): toimii vakioajassa.

int_to_string(): toimii vakioajassa.

encrypt(): pow(message_int, e, n):n aikavaativuus skaalautuu sen mukaan, miten suuri viesti m on ja kuinka suuri kantaluku e on. Modulon laskeminen toimii vakioajassa. Aikavaativuus on siis O(m*e).

decrypt(): pow(ciphertext, d, n):n aikavaativuus on sama kun encryptauksen.

**Tilavaativuus:**

Merkkijonon/viestin muuttaminen string -> int tapahtuu funktiolla string_to_int. Jos viestissä on B tavua, sen integer-vastineessa on 8*B bittiä. Tässä on siis tilavaatimus O(B).

Viestin salaaminen pow(message_int, e, n) käyttää samanaikaisesti muutamaa 2048-bittisiä lukuja, joten tämän tilavaatimus on O(k) jossa k on avaimen pituus (eli 2048).

Tilavaativuus on siis yhteensä O(B+k).

**Työn mahdolliset puutteet ja parannusehdotukset:**

Projektia olisi parantanut se, että olisin luonut generoiduista avaimista tiedostot kuten oikeassa RSA-salauksessa tehdään. Jätän kuitenkin projektin tälle tasolle ajan puutteen vuoksi, kun avainten käyttö kuitenkin onnistuu toivotusti – hieman kömpelömmin vain kun miten jatkokehityksellä voisi.

Jatkokehitystä voisi olla myös graafisen käyttöliittymän luonti, jolla voitaisiin importata tiedosto ja avain, sekä salata tai purkaa tiedoston sisältö.

**Laajojen kielimallien (ChatGPT yms.) käyttö.**

Käytin projektissa ChatGPT:tä muutamien apufunktioiden luomiseen, jotka ovat kirjattu viikkoraportteihin. Pyrin luomaan tekoälyllä vain sellaisia funktioita jotka ovat aidosti apufunktioita, eikä varsinaisesti algoritmin toteutusta.

Funktiot: string_to_int(), int_to_string(), mod_inverse()

**Lähteet, joita olet käyttänyt:**


- https://www.geeksforgeeks.org/dsa/primality-test-set-3-miller-rabin/
- https://en.wikipedia.org/wiki/RSA_cryptosystem
- https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
- https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test