Shorov algoritam za kriptiranje

Vojvodić, Eduard

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Eduard Vojvodić

SHOROV ALGORITAM ZA KRIPTIRANJE

ZAVRŠNI RAD

Zagreb, 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
ODBOR ZA ZAVRŠNI RAD

Zagreb, 10. travnja 2019.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Baze podataka**

ZAVRŠNI ZADATAK br. 5099

Pristupnik: **Eduard Vojvodić (0135243187)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Shorov algoritam za kriptiranje**

Opis zadatka:

U radu je potrebno opisati i usporediti rad klasičnih i kvantnih računala. Objasniti što je to kriptografija i RSA algoritam. Potrebno je opisati Shorov algoritam na kvantnim računalima i njegovu primjenu u području kriptografije. Provesti i objasniti analizu slučaja jednostavne primjene Shorovog algoritma.

Mentor:

prof. dr. sc. Hrvoje Gold

Predsjednik povjerenstva za završni ispit:

Sveučilište u Zagrebu

Fakultet prometnih znanosti

ZAVRŠNI RAD

SHOROV ALGORITAM ZA KRIPTIRANJE

SHOR'S ENCRYPTION ALGORITHM

Mentor: prof. dr. sc. Hrvoje Gold

Student: Eduard Vojvodić

Neposredni voditelj/komentor:

JMBAG: 0135243187

Tomislav Erdelić, mag. ing. el. techn. inf.

Zagreb, rujan 2019.

SAŽETAK

Kriptografija je znanost o zaštiti podataka prilikom prijenosa i pohrane informacija što ju danas čini jednom od najznačajnijih djelatnosti u komunikacijskim sustavima i sustavima za obradu podataka. U svijetu „surfanja“ internetom, pohranjivanja osobnih informacija na društvenim mrežama, korištenjem pogodnosti poput e bankarstva, sigurnost podataka je najvažnije pitanje. Danas se sigurnost podataka na internetu osigurava kriptografskom metodom poznatom pod imenom RSA (Rivest–Shamir–Adleman). Ova kriptografska metoda koristi asimetrične ključeve kako bi zaključala podatke i osigurala njihovu sigurnost. No s razvojem razumijevanja pojava iz domene kvantne mehanike kao što su qubiti, superpozicija ili kvantna Fourierova transformacija razvila se i ideja o izradi kvantnog računala. Ono bi moglo obavljati određenu vrstu procesa neusporedivo brže nego današnja klasična računala. Jedan od takvih procesa je i traženje prostih faktora nekog broja poznat pod imenom Shorov algoritam. To dovodi u potencijalnu opasnost kriptografske algoritme koji su zasnovani na predodžbi da je taj proces težak i dugotrajan čak i za jaka klasična računala. RSA je jedna od metoda koje se baziraju na takvoj predodžbi.

KLJUČNE RIJEČI: kriptografija; RSA; klasična računala; kvantna računala; prosti brojevi; Shorov algoritam

SUMMARY

Cryptography is the science of data protection in the transmission and storage of information, which makes it today one of the most important activities in communications and data processing systems. In the world of “surfing” the internet, storing personal information on social media, and using amenities like e-banking, data security is the most important issue. Today, internet data security is ensured using a cryptographic method known as RSA (Rivest-Shamir-Adleman). This cryptographic method uses asymmetric keys to lock the data and ensure it’s security. But with the development of certain aspects in the field of quantum mechanics, such as qubits, superposition and quantum Fourier transformation, the idea of creating a quantum computer has evolved. Such computers would be able to perform a certain type of process at a much faster rate than today's classic computers. One such process is the search for a number's prime factor known as the Shor algorithm. This puts at risk the cryptography algorithms that are based on the premise that this process is difficult and time-consuming, even for strong classic computers. RSA is one of the methods based on such premise.

KEYWORDS: cryptography; RSA; classical computers; quantum computers; prime numbers; Shor's algorithm

Sadržaj

1. Uvod

Želja za bržom i kvalitetnijom obradom podataka pobudila je razvoj novih tehnologija kao što su kvantna računala. Iako su ta računala danas komercijalno nedostupna, moguće ih je koristiti u istraživačke svrhe. S njihovim razvojem razviti će se i njihova sposobnost obrade podataka što će dalje biti korisno za daljnji razvoj tehnologije i njezine primjene. Jedno od područja gdje je bitno imati i brzinu i dobru metodu obrade podataka je u razbijanju kriptografskih šifri(kodova) kako bi razbijanje bilo učinkovito. Iako razbijanje šifri danas, s obzirom na povećani broj slučajeva krađe identiteta, osobnih podataka i slično, za sobom vuče loše konotacije, razvoj kriptografskih algoritama za razbijanje kodiranih poruka država može (smije li i kada je drugo pitanje) upotrebljavati u svrhu zaštite ljudi od izvanrednih opasnosti kao što su teroristički napadi. Kroz povijest je razvijeno više kriptografskih metoda s ciljem kriptiranja određenih informacija, a s razvojem klasičnih računala pojavio se i jedan od najsigurnijih oblika kriptografije danas, takozvani RSA (Rivest–Shamir–Adleman) algoritam.

Cilj i svrha ovog završnog rada je opisati Shorov algoritam, njegov način rada te opisati popratne pojmove vezane za kvantna računala potrebne da se shvati korištena terminologija.

Rad se sastoji od 6 poglavlja:

1. Uvod
2. Računala
3. Kriptografija
4. Shorov algoritam
5. Analiza slučaja
6. Zaključak

U drugom poglavlju opisan je se osnovni koncept rada klasičnih i kvantnih računala. Svrha ovog poglavlja je postaviti temelje za razumijevanje terminologije četvrtog poglavlja.

Treće poglavlje opisuje pojam kriptografije. Prolazi se kroz kriptografiju od prošlosti do danas te se opisuje rad najpoznatijih i najbitnijih kriptografskih metoda.

Četvrto poglavlje je ujedno i najvažnije poglavlje. U njemu će se detaljno opisati rad algoritma za kvantna računala, Shorov algoritam.

Kroz peto poglavlje teorija Shorovog algoritma potkrijepljena je primjerom implementiranim u simuliranom kvantnom okruženju na klasičnom računalu. Programski jezik koji će se koristiti bit će Python.

2. Računala

Računala klasične arhitekture razvijaju se više od 80 godina. Rad Alana Turinga, Harolda Keena, Johna von Neumanna, Stevea Jobsa, i mnogih drugih te kompanija poput IBM-a (International Business Machines) i Apple-a kroz godine rezultirao je time da su računala moćni alat za obradu podataka, informacija i znanja. Dok su prva računala bila takvih veličina da su trebala vlastite sobe, a bila su sporija i manjih kapaciteta od modernih kalkulatora, moderna klasična računala sadrže komponente koje su veličine 14 nanometara (nm). Pošto je dijametar jednog atoma silikona nešto manji od 0,225nm nema mnogo prostora za smanjivanje veličine tranzistora. Osim ako se ne spusti na kvantnu razinu, a tada se počinje govoriti o kvantnim računalima.

Klasično računalo vrši radnje nad nizovima nula i jedinica, poput 110010111011000. Svaka se vrijednost u takvom nizu naziva bit, a on je u stanju 0 ili 1. Da bi predstavljao takve zbirke bitova, računalo mora sadržavati odgovarajuću zbirku fizičkih sustava, od kojih svaki može postojati u dva nedvojbeno različita fizička stanja, povezana s vrijednošću (0 ili 1) apstraktnog bita koji fizički sustav predstavlja. Takav fizički sustav može biti, na primjer, prekidač koji može biti otvoren (0) ili zatvoren (1), ili magnet čija se polarizacija može usmjeriti u dva različita smjera, "gore" (0) ili "dolje" (1)). Budući da cijelo računalo radi isključivo s ta dva stanja, odnosno radi na binarnom sustavu, kaže se da je to strojni jezik. Svi podaci koji se nalaze na računalu, neovisno o vrsti podatka, na najnižoj razini na kojoj stroj radi zapisani su u nizovima nula i jedinica [1].

Da bi računalo bilo kvantno računalo, fizički sustavi koji kodiraju pojedine bitove moraju biti u potpunosti izolirani od bilo kakvih vanjskih utjecaja. Sve druge interakcije, ma koliko bile nevažne u običnom računalu, koje će se nazivati „klasičnim“ kada se želi suprotstaviti kvantnom računalu, uvode potencijalno katastrofalne poremećaje u rad kvantnog računala. Takve katastrofalne interakcije mogu uključivati interakcije s vanjskim okruženjem kao što su molekule zraka koje se odbijaju od fizičkih sustava koji predstavljaju bitove. Mogu čak biti poremećene interakcije između računski relevantnih stupnjeva slobode fizičkih sustava koji predstavljaju bitove s drugim stupnjevima slobode tih istih sustava, povezanih s računalno nebitnim značajkama njihove unutarnje strukture. Sve takve interakcije rezultiraju

"dekoherencijom", što je pogubno za kvantno računanje. Da bi se izbjegla dekoherencija, pojedinačni se bitovi ne mogu kodirati u fizičke sustave makroskopske veličine, jer se takvi sustavi ne mogu izolirati od vlastitih nebitnih unutarnjih stupnjeva slobode. Bitovi se moraju kodirati u vrlo malom broju stanja sustava veličine atoma, gdje dodatni unutarnji stupnjevi slobode ne dolaze u obzir zato što ne postoje ili zato što su im potrebne nedostupno velike količine energije. Takvi se sustavi s atomskom skalom također moraju odvojiti od svoje okoline, osim u potpunosti kontroliranim interakcijama koje su povezane sa samim procesom računanja. Dvije stvari sprečavaju situaciju da postane beznadna:

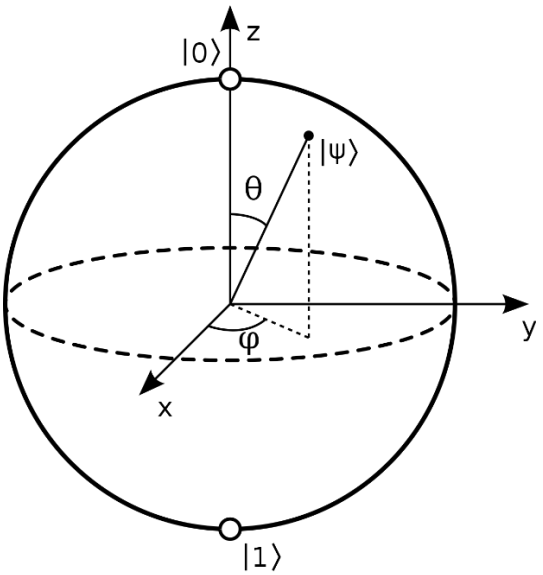
1) budući da razdvajanje između diskretnih energetskih razina sustava na atomskoj ljestvici može biti znatno veće od razdvajanja između nivoa velikog sustava, lakše je postići dinamičku izolaciju atomskog sustava jer može primiti značajan udarac bez da se atom izbaci iz njegova osnovnog stanja,
2) otkriće da se pogreške izazvane izvan serijskim interakcijama mogu ispraviti pod uvjetom da se događaju s dovoljno niskom brzinom.

Iako je ispravljanje pogrešaka rutinsko za bitove predstavljene klasičnim sustavima, kvantna korekcija pogreške ograničena je velikim zahtjevom da se izvrši bez ikakvog saznanja o tome što bi izvorno ili oštećeno stanje bita moglo biti. No, iako situacija nije beznadna, praktične poteškoće u načinu postizanja korisnog kvantnog računanja su ogromne. [1]

Kako klasična računala rade s bitovima tako kvantna računala rade s kvantnim bitovima (engl. quantum bits), tako zvanim qubitima. Baš kao i njegov klasični rođak, i qubit može imati vrijednost od 0 ili 1. Fizički, qubiti se mogu predstaviti kao bilo koji dvo-stupanjski kvantni sustav poput:

- zakretanja čestice u magnetskom polju gdje gore označava vrijednost 0, a dolje označava vrijednost 1 ili,
- polarizacija jednog fotona gdje vodoravna polarizacija označava vrijednost 1, a vertikalna polarizacija označava vrijednost 0.

U oba slučaja 0 i 1 su jedina moguća stanja. Geometrijski se qubiti mogu vizualizirati pomoću oblika koji se zove Bloch-ova sfera, prikazana na slici 1 , instrumenta nazvanog po švicarskom fizičaru Felixu Blochu.



Slika 1. Bloch-ova sfera [2]

Formalno, Blochova sfera je geometrijski prikaz u trodimenzionalnom Hilbertovom prostoru čistog stanja dvo-stupanjskog kvantnog sustava ili qubita. Sjeverni i južni pol sfere predstavljaju standardne osnovne vektore |0> i |1>, oni zauzvrat odgovaraju vrtnji-gore (engl. spin – up) i vrtnji-dolje (engl. spin – down) elektrona. Pored osnovnih vektora, sfera može biti u još jednom stanju koje se označava s |ψ>, a može se nalaziti pod nekim kutom Θ u odnosu na z-os, nekim kutom φ u odnosu na x-os ili u kombinaciji ta dva kuta. Ta pozicija se zove superpozicija i u osnovi je vjerojatnost za 0 ili 1. Trik je u tome što se ne može predvidjeti kakva će biti, osim u trenutku promatranja kada se vjerojatnost uruši u definitivno stanje [2].

Iako se na prvi pogled čini da su kvantna računala brža i bolja od klasičnih računala, ona za velik broj računalnih operacija to nisu. Zašto se onda masivno radi na razvoju kvantnih računala? Zato što postoje operacije koje mogu izvršiti brže od klasičnih računala, a izvršavaju ih milijun puta brže od klasičnih računala. I dok to nije bitno ako klasičnom računalu trebaju milisekunde, sekunde ili minute, bitno je kada mu treba nekoliko desetaka tisuća godina. Čak i ako mu treba milijun godina, kvantnom računalu će trebati samo jedna. Dakle budućnost tehnologije neće počivati na isključivo kvantnoj infrastrukturi već će se koristiti i klasična i kvantna računala kako bi se vrijeme obrade informacija svelo na minimum.

3. Kriptografija

Kriptografija (grč. kriptós skriven i gráfo pisati) je pisanje tajnim znakovima, šifriranje [3].
Bitno je istaknuti da je upotrijebljena riječ šifriranje, a ne kodiranje. Odnosno šifriranje i kodiranje nisu istoznačnice.

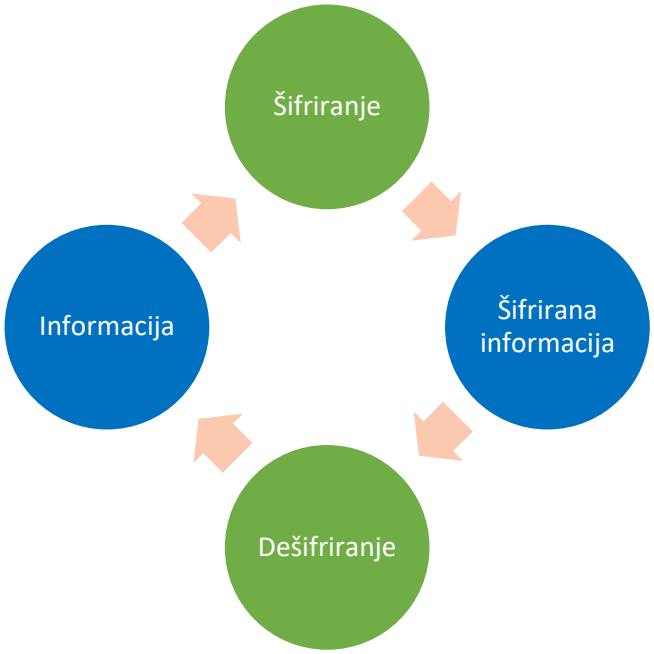
Svrha kodiranja je transformirati podatke tako da ih drugačija vrsta sustava može ispravno i sigurno konzumirati. Primjer su binarni podaci koji se šalju putem e-pošte ili pregledavanje posebnih znakova na internetskim stranicama. Cilj kodiranja nije držati podatke u tajnosti, već osigurati da se oni mogu pravilno konzumirati. Kodiranje pretvara podatke u drugi format pomoću shema koje su javno dostupne kako bi se kodirane informacije smogle lako konvertirati u izvorni oblik. Za dekodiranje nije potreban ključ, već je potreban algoritam koji se koristio za kodiranje [4].

Svrha šifriranja je transformirati podatke da bi se sačuvala tajnost podataka. Primjerice, slanje tajnog pisma koje će biti razumljivo isključivo namijenjenom primatelju. Cilj šifriranja je osigurati da podatci ili informacija budu konzumirani isključivo od točno određenog primatelja. Šifriranje pretvara podatke u drugi format na takav način da samo određeni pojedinci mogu poništiti transformaciju podataka, odnosno vratiti ju u prvobitno stanje u svrhu konzumacije sadržaja. Da bi se ovo postiglo koristi se ključ koji se čuva u tajnosti, zajedno sa šifriranom informacijom i algoritmom s kojim se vršilo šifriranje [4].

Kriptografija se stoga može podijeliti na dvije metode:

1) Metodu šifriranja koja služi za zaštitu podataka i informacija upotrebom tajnog algoritma kako ona (podatak ili informacija) ne bi postala javno dostupna, odnosno namijenjena je za točno određenog primatelja, a pročitati se može isključivo ako se posjeduju sva tri potrebna elementa; ključ, algoritam i šifrirana informacija.
2) Metoda dešifriranja koja upotrebom identičnog algoritma i ključa iz postupka šifriranja vraća sadržaj šifrirane informacije u prvobitno stanje pogodan za korištenje.

Dijagramom šifriranja i dešifriranja (slika 2) prikazuju se metode kriptografije (zeleno) i njihovi produkti (plavo). [5]



Slika 2. Dijagram kriptografije [5]

3.1. Kriptografija kroz povijest

Kriptografija je kroz povijest iznimno napredovala. Od mezopotamskih pločica koje su znale sadržavati šifrirane recepte za proizvodnju stvari, preko Cezarove šifre do Enigme. Svi primjeri kroz povijest imaju jednu zajedničku stvar, sakriti određenu informaciju koja se smatra bitnom ili privatnom kako ju nitko ne bi mogao zloupotrijebiti protiv vlasnika. U sljedećim poglavljima ukratko će se opisati metode kriptografije koje su bitno utjecale na izgled današnjeg svijeta.

3.2. Cezarova šifra

Cezarova šifra jedan je od najpoznatijih i najjednostavnijih oblika kriptografije. Temelji se na supstituciji svakog slova u poruci za određeni fiksni razmak između slova u abecedi. To znači da ukoliko je fazni pomak na primjer 9, slovo 'A' (prvo slovo u abecedi) u poruci koja se nastoji šifrirati postat će slovo 'F' (deseto slovo u abecedi). Ova tvrdnja se može izraziti i preko izraza (1).

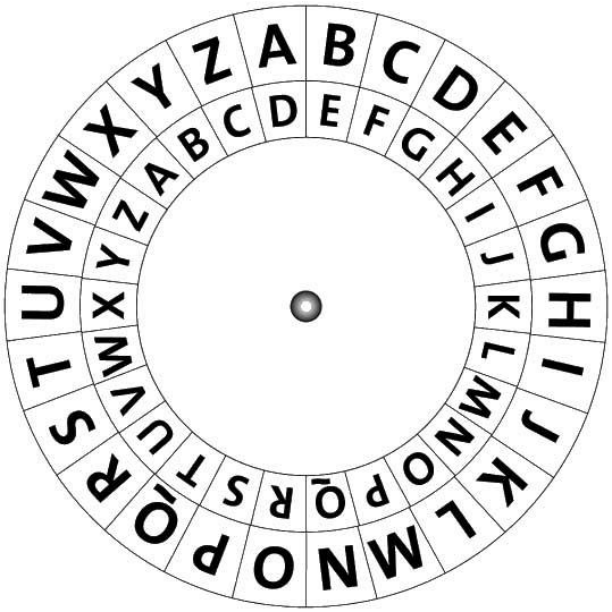
$$E_k(n) = n + k \; mod \; q \tag{1}$$

Oznake u izrazu (1) imaju sljedeće značenje:

- '$E_k(n)$' – šifrirani znak,

- 'n' – ne šifrirani znak,
- 'k' – fazni pomak od ne šifriranog znaka,
- 'q' – maksimalan broj znakova koji se koristi.

Navedeni izraz opisuje proces šifriranja. Šifriranje po Cezarovoj šifri se također može prikazati pomoću dva koncentrična prstena kao što je prikazano na slici 3. Tada se međusobnom rotacijom prstena stvara određeni fazni pomak između istog slova abecede koji će se koristiti za šifriranje poruke [6].



Slika 3. Cezarov disk s engleskom abecedom i faznim pomakom k = 23 (preuzeto iz [7])

S obzirom na to da je navedeni primjer objašnjen na temelju hrvatske abecede, potrebno je naglasiti da se za slučajeve u kojim su slova složena od više znakova poput slova Dž, Lj i Nj, navedena slova definiraju se kao zasebna cjelina, a ne kao kombinacija dvaju slova, odnosno vrijednost q iznosi 30, a ne 27 [6].

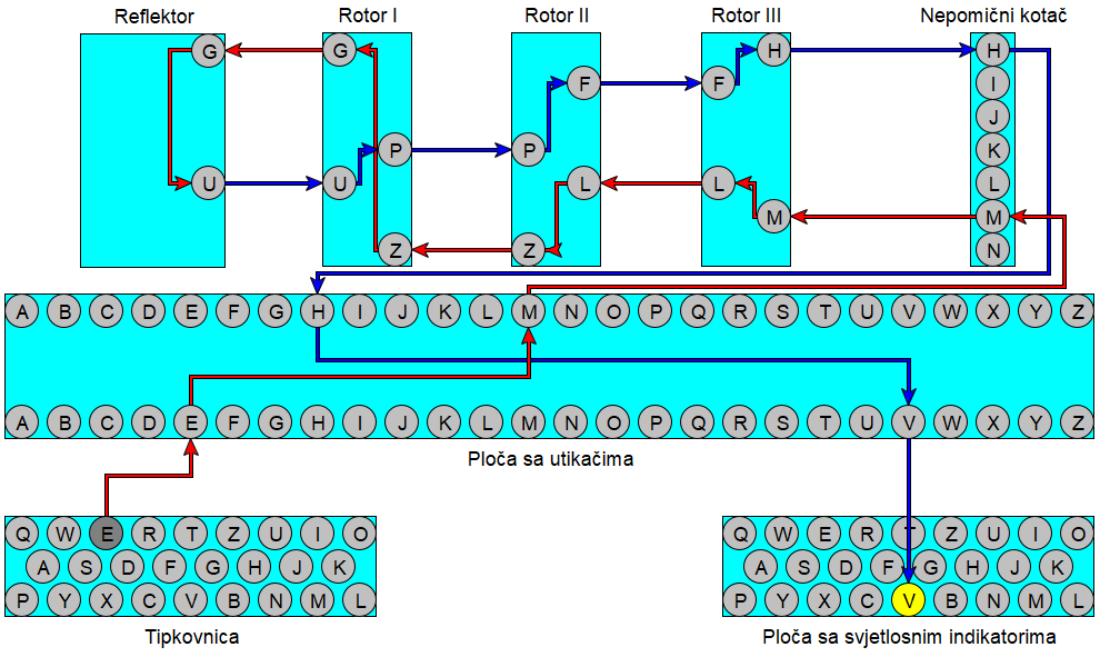
Nakon što je šifrirana poruka došla do odredišta, potrebno je provesti postupak dešifriranja kako bi se ponovno dobila odaslana informacija. Sam postupak je obrnut od postupka šifriranja, a vrši se po formuli prikazanoj izrazom (2). Oznake su iste kao i za izraz (1).

$$n = E_k(n) - k \; mod \; q \tag{2}$$

Ukoliko se želi dešifrirati poruka šifrirana Cezarovom šifrom ključ bi bio 'k', a algoritam formula za dešifriranje koja se može izvući iz formule za šifriranje ukoliko su poznata značenja pojedinih članova izraza [6].

3.3. Stroj Enigma

Enigma je stroj koji je izumio njemački inženjer dr. Arthur Scherbius 1918 godine. Prvotno se koristio komercijalno, no s nadolaskom rata primjenu je pronašao i unutar vojske. Koristile su ga Sile Osovine, no najsloženiji model je koristila njemačka vojska [8]. Sam uređaj temelji se, poput Cezarove šifre, na supstituciji slova, no izvedba je puno složenija. Stroj se sastoji od rotirajućih diskova, reflektora, ploče s utikačima, tipkovnice i ploče sa svjetlosnim indikatorima, a sama supstitucija se izvršava više puta. Shema šifriranja slova E je prikazana na slici 4.



Slika 4. Postupak šifriranja unutar Enigme [9]

Prilikom pritiska tipke 'E' na tipkovnici, šalje se električni signal prema ploči s utikačima. Na ploči s utikačima nalaze se utičnice označene slovima. Ukoliko je spojena utičnica jednog slova s utičnicom drugog slova ta slova će se međusobno zamijeniti prilikom ulaska u daljnji dio sustava. Na primjer, ako su međusobno spojena slova 'E' i 'M', kao što je to prikazano slikom 4 te se pritisne slovo 'E', sustav će unijeti slovo 'M', i obrnuto. Ukoliko neko slovo nije

upareno s drugim slovom na ploči s utikačima, neće se izvršiti supstitucija slova već će upravo pritisnuto slovo na tipkovnici ići dalje u sustav. Signal se zatim šalje dalje preko nepomičnog kotača, koji služi samo kako bi prenio signal na rotor s kojim je povezan. Rotor na koji signal dolazi zove se rotor III na slici 4. Većina modela Enigmi je dolazila s pet rotora od kojih su se birala tri prilikom uporabe stroja. Svaki rotor sadrži u sebi unutarnji prsten kontakata i vanjski prsten kontakata, a njihova je svrha šifriranje signala. Vanjski kontakti prstena spajaju svaki rotor na sljedeći rotor, nepomični rotor ili reflektor kao i na njegov unutarnji prsten. Unutarnji kontakti prstena mogu se zakretati u odnosu na vanjski prsten. Također sam se rotor može zakrenuti u odnosu na nepomični rotor, tako da izlaz 'M' nepomičnog rotora nije povezan s ulazom 'M' na rotirajućem rotoru. Nakon što se upiše jedno slovo, rotori se okreću za jedan položaj tako da ista slova u jednoj šifriranoj poruci vrlo rijetko označavaju isto slovo u ne šifriranoj poruci. Dakle, ako se pritisne na primjer slovo 'K', Enigma će ga šifrirati u neko drugo slovo, na primjer slovo 'B'. Ukoliko se odmah nakon toga ponovno pritisne slovo 'K', Enigma će ga šifrirati u potpuno neko drugo slovo, ali ono zasigurno neće ponovno biti 'B' . Da bi se dodatno zakompliciralo, svaki rotor ima zareze na sebi, a različiti rotori imaju zarez u različitim položajima, te kada su oni dosegnuti potaknut će se jedan dodatan korak na lijevom rotoru od rotora na kojem je zarez dosegnut. Ovaj mehanizam se naziva i mehanizmom dvostrukog koraka.

Reflektor uzima ulaz i reflektira električni signal za povratno putovanje kroz rotore. Važno je da se signal šifrira kada se reflektira, zbog načina na koji je dizajnirana Enigma. Dakle, ako je izlaz reflektora isto slovo kao i njegov ulaz kad signal prođe natrag kroz rotore, oni će jednostavno dešifrirati ono što je već šifrirano i dobit će se ponovno izvorno nešifrirano pismo. Nakon toga reflektirani signal prolazi natrag kroz rotore. Dakle, slovo 'U' prolazi kroz rotor I i postaje 'P', a zatim prolazi kroz rotor II i postaje 'F', a zatim prolazi kroz rotor III i postaje 'H'. Signal ostaje nepromijenjen dok ponovno prolazi kroz nepomični rotor. Prije nego što prođe kroz ploču signal ponovno ostaje ne promijenjen ako nema spojenog utikača ili se promijeni ako je slovo 'H' priključeno na drugo slovo. U primjeru sa slike 4, 'H' se priključuje na slovo 'V', pa je signal koji pristiže na ploču sa svjetlosnim indikatorima 'V'. Operater Enigme tada zabilježi slovo te može unijeti novo slovo za šifriranje ili dešifriranje [9].

3.4. Kriptografija danas

Unutar područja računalnih znanosti, kriptografija označava sigurne informacijske i telekomunikacijske tehnike izvedene iz matematičkih koncepata te na skupu točno određenih kalkulacija koje se nazivaju algoritmi kako bi se prenijele poruke na načine koje je teško dešifrirati. Ovi deterministički algoritmi se upotrebljavaju za generiranje kriptografskih ključeva te za digitalne potpise i verificiranje u svrhu zaštite privatnosti podataka ili toka podataka.

RSA (Rivest–Shamir–Adleman) jedan je od prvih kripto sistema s javnim ključem i u širokoj je upotrebi kada se priča o sigurnom prijenosu podataka. RSA šifriranjem, poruke se kriptiraju kôdom koji se zove javni ključ. Zbog određenih matematičkih svojstava RSA algoritma, nakon što je poruka šifrirana javnim ključem ona se može dešifrirati samo drugim ključem, poznatim pod imenom privatni ključ. Svaki RSA korisnik ima par ključeva koji se sastoji od njihovih javnih i privatnih ključeva. Kao što ime sugerira, privatni ključ mora biti čuvan u tajnosti dok se javni ključ smije dijeliti.

Sheme šifriranja javnih ključeva razlikuju se od šifriranja simetričnih ključeva, gdje i šifriranje i dešifriranje koriste isti privatni ključ. Te razlike čine šifriranje javnih ključeva poput RSA korisnim za komunikaciju u situacijama kada prethodno nije bilo mogućnosti provesti sigurnu distribuciju ključeva. Algoritmi simetričnih ključeva imaju vlastite aplikacije, poput šifriranja podataka za osobnu upotrebu ili kada postoje sigurni kanali preko kojih se privatni ključevi mogu dijeliti. RSA šifriranje često se koristi u kombinaciji s drugim shemama šifriranja ili za digitalni potpis kojim se može dokazati autentičnost i cjelovitost poruke. Obično se ne koristi za šifriranje cijelih poruka ili datoteka jer je manje učinkovita i teška za resurse šifriranja simetričnog ključa.

Da bi se stvar učinila učinkovitijom, podaci će općenito biti šifrirani algoritmom simetričnog ključa, a potom će se simetrični ključ kodirati RSA šifriranjem. U ovom će procesu samo entitet koji ima pristup privatnom ključu RSA moći dešifrirati simetrični ključ kojim se mogu otključati podatci. RSA šifriranje djeluje pod pretpostavkom da je algoritam lako izračunati u jednom smjeru, ali gotovo nemoguće u drugom. Na primjer, puno je teže pronaći dva prosta faktora koji daju rezultat 701111 nego izračunati umnožak faktora 907 i 773. Još jedan zanimljiv aspekt ove jednadžbe je da je lako utvrditi jedan od prostih brojeva ako je već

poznat drugi, kao i rezultat. Ako je rečeno da je 701111 rezultat prostog broja 907 pomnožen s drugim prostim brojem, lako je za utvrditi da je drugi prosti broj 773. Budući da je odnos između tih brojeva jednostavno izračunati u jednom smjeru, ali nevjerojatno teško u drugom smjeru, funkcija je poznata i pod nazivom funkcija zamke. Bitno je za napomenuti da brojeve ovog reda veličine današnja računala obrađuju u trivijalnom vremenu. Zbog toga RSA koristi mnogo veće brojeve. Red veličine prostih brojeva u stvarnoj RSA implementaciji varira, no u 2048-bitnom RSA-u oni bi se sastavili kako bi napravili ključeve duge 617 znamenki, odnosno 10^{617} – 1 brojeva.

Spomenute funkcije zamke čine osnovu za funkcioniranje shema šifriranja javnih i privatnih ključeva. Njihova svojstva omogućuju dijeljenje javnih ključeva bez ugrožavanja poruke ili otkrivanja privatnog ključa.

Prvi korak šifriranja poruke s RSA algoritmom je generiranje ključeva. Da bi se to učinilo, potrebna su dva prosta broja koja će biti označena s 'p' i 'q', a odabrani su testom primarnosti. Test primarnosti je algoritam koji učinkovito pronalazi proste brojeve, kao što je test primalnosti Rabin-Miller. Prosti brojevi u RSA moraju biti vrlo veliki, a također i relativno udaljeni. Brojevi koji su mali ili bliže jedan drugom mnogo je lakše probiti. Unatoč tome, primjer će upotrijebiti manje brojeve kako bi stvari bilo lakše pratiti i računati. Pretpostavit će se da test primarnosti daje proste brojeve koji su se koristili prethodno, 907 i 773. Sljedeći je korak otkrivanje modula (n) koristeći izraz (3). Iz toga slijedi da je n = 701111.

$$n = p \times q. \tag{3}$$

Oznake u izrazu (3) imaju sljedeće značenje:

- 'n' – javni ključ kojim su se zaključali podaci,
- 'p' – prvi prosti faktor javnog ključa,
- 'q' – drugi prosti faktor javnog ključa.

Kad je definiran n, koristit će se Carmichaelova totientna funkcija dana izrazom (4).

$$\lambda\,(n)\;=\;LCM\,(p\;-\;1, q\;-\;1) \tag{4}$$

Oznake u izrazu (4) imaju sljedeće značenje:

- '$\lambda\,(n)$ ' – Carmichaelov totient za n,

- 'LCM' – operator za najniži zajednički višekratnik

Ostale oznake su iste kao i za izraz (3). Ukoliko se svi dosadašnji brojevi uvrste u izraz (4) dobije se zapis:

$$\lambda (701111) = LCM (907 - 1, 773 - 1)$$

$$\lambda (701111) = LCM (906, 772)$$

Rješenje tada iznosi:

$$\lambda (701111) = 349716$$

S dobivenim Carmichael-ovim najčešćim brojem izvedenim iz prostih brojeva, može se odrediti javni ključ. Kod RSA, javni ključevi sastoje se od prostog broja e, kao i n. Broj e može biti bilo koji broj između 1 i vrijednosti za λ (n), a λ (n) u primjeru iznosi 349716. Iz razloga jer se javni ključ dijeli otvoreno, nije toliko važno da e bude slučajni broj. U praksi se e obično postavlja na 65,537. U slučaju kada se nasumično odabere broj veći od 65,537, šifriranje je manje učinkovito. Za primjer će biti zadržani mali brojevi kako bi proračuni bili učinkoviti. Vrijednost prostog broja e može iznositi na primjer 11. Konačni se šifrirani podaci nazivaju šifriranim tekstom, a označavat će se slovom c. Šifrirani podaci se izvode iz otvorene poruke koja se označava slovom m te primjenom javnog ključa pomoću izraza (5).

$$c = m^e \bmod n \tag{5}$$

Oznake u izrazu (5) imaju sljedeće značenje:

- 'c' – vrijednost šifrirane poruke,
- 'm' – vrijednost originalne poruke,
- 'e' – slučajno odabrani prosti broj,
- 'n' – modul.

Vrijednost e i n su dostupne, a jedina je nepoznanica mod koji se odnosi na operaciju modulo, što je zapravo ostatak nakon dijeljenja djelitelja i djeljenika. Na primjer 10 mod 3 iznosi 1. To je zato što 3 ulazi u 10 tri puta, a ostatak iznosi 1. Pojednostavljeno, ako se pretpostavi da je poruka m koja se šifrira i čuva u tajnosti samo jedan broj, na primjer 4. Tada se dosadašnje vrijednosti mogu uvrstiti u izraz (5), čime se dobiva:

$$c = 4^{11} \bmod 701111$$

$$c = 4194304 \bmod 701111.$$

Što znači da vrijednost šifrirane poruke iznosi:

$$c = 688749$$

Stoga, korištenjem RSA za šifriranje poruke 4, javnim ključem, dobiva se šifrirani tekst 688749.

Cilj je bio zadržati poruku 4 u tajnosti. Na nju je primijenjen javni ključ, što omogućava šifrirani rezultat od 688749. Sada kada je šifriran, slanje broja 688749 vlasniku para ključeva je sigurno. Vlasnik para ključeva jedina je osoba koja će to moći dešifrirati svojim privatnim ključem. Nakon dešifriranja, vidjet će poruku originalno poslanu poruku koja glasi „4“.

U RSA šifriranju, nakon što se podaci ili poruka pretvore u šifriran tekst javnim ključem, on se može dešifrirati samo privatnim ključem iz istog para ključeva. Privatni se ključevi sastoje od d i n. Poznat je n, a za pronalaženje d koristi se izraz (6).

$$ed \equiv 1 \pmod{\lambda(n)} \tag{6}$$

Oznake u izrazu (6) imaju sljedeće značenje:

- 'd' – komponenta privatnog ključa,

a ostale oznake imaju isto značenje kao u izrazima (4) i (5). U izrazu (6) javlja se operator mod u kombinaciji s oznakom„≡“ koja simbolizira modularni inverz.

To u suštini znači da se umjesto izvođenja standardne operacije modula, provodi obrnuti postupak. Ukoliko se postupak provodi na klasičnom računalu, obično se koristi prošireni Euklidov algoritmom [10]. Uvrštavanjem dosadašnjih podataka u izraz (6) dobije se:

$$11d \equiv 1 \bmod 349716$$

Daljnjim raspisivanjem dobiva se čitljiviji oblik izraza (6) s uvrštenim brojevima koji glasi:

$$(11d - 1) \bmod 349716 == 0$$

Vrijednost „d“ za koju će to vrijediti iznosi :

$$d = 254339$$

Sa dobivenom vrijednosti d, mogu se dešifrirati poruke koje su šifrirane javnim ključem pomoću izraza (7).

$$m = c^d \bmod n \tag{7}$$

Oznake imaju isto značenje kao u dosadašnjim izrazima.

Nakon što se šifrirala poruka javnim ključem, ona je dala vrijednost za c koja je iznosila 688749. Prethodno su poznate vrijednosti d koja je jednaka 254339, kao i da je n jednak 701111. Time se dobiva izraz $m = 688749^{254339} \bmod 701111$.

Kao što se može primijetiti, potenciranje bilo kojeg, a pogotovo velikih brojeva s brojem reda veličine kao što je vrijednost d neće se moći izvršiti na prosječnom kalkulatoru stoga se preporučuje korištenje RSA kalkulatora za dešifriranje šifre zaključane RSA algoritmom.

Nakon što se unesu potrebni podaci za dešifriranje, a to su javni ključ n, ključ za šifriranje e, ključ za dešifriranje d te naravno i šifrirana poruka c. Dobit će se originalna poruka. Ako se sve ispravno napravilo, dobiti će se odgovor da je šifrirana poruka glasila 4, a upravo je ona bila originalna poruka koja se šifrirala javnim ključem.

U prethodno navedenim koracima pokazano je kako dva entiteta mogu sigurno komunicirati bez prethodnog dijeljenja kôda. Prvo, svi trebaju uspostaviti vlastite parove ključeva i dijeliti javni ključ jedan s drugim. Dva entiteta moraju čuvati svoje privatne ključeve u tajnosti kako bi komunikacija ostala sigurna. Jednom kada pošiljatelj ima javni ključ svog primatelja, on ih može upotrijebiti za šifriranje podataka koje želi zaštititi. Jednom kada je šifriran javnim ključem, on se može dešifrirati samo privatnim ključem iz istog para ključeva. Čak se isti javni ključ ne može upotrijebiti za dešifriranje podataka. To je zbog svojstava funkcija zamka vrata koja su spomenuta prethodno.

Kad primatelj primi šifriranu poruku, koristi svoj privatni ključ za pristup podacima. Ako on želi odgovoriti na siguran način, tada može svoju poruku šifrirati javnim ključem stranke s kojom komunicira. Ponovno, nakon što je šifriran javnim ključem, jedini način na koji se može pristupiti informacijama je putem odgovarajućeg privatnog ključa. Na taj način, nepoznate strane mogu RSA šifriranje koristiti za sigurno slanje podataka među sobom. Na

ovim temeljima izgrađeni su značajni dijelovi komunikacijskih kanala koji se koriste u današnjim mrežama.

U obrađenom primjeru stvari su mnogo pojednostavljene kako bi ih se lakše razumjelo, zbog čega se šifrirala jednostavna poruka "4". Mogućnost šifriranja broja 4 ne čini se osobito korisnom, pa se postavlja pitanje kako se može šifrirati složeniji skup podataka, poput simetričnog ključa ili čak poruke? Realnost je da su sve informacije koje računala obrađuju pohranjene binarnim zapisom (1 i 0) i koriste se standardi kodiranja poput ASCII ili Unicode da bi ih se predstavilo na način koji ljudi mogu razumjeti (slova).

To znači da ključevi poput "n38cb29fkbjh138g7fqijnf3kaj84f8b9f ..." i poruke poput "Eduard piše završni rad" već postoje kao brojevi, koji se lako mogu izračunati u RSA algoritmu. Brojevi kojima su oni predstavljeni mnogo su veći i teže je njima upravljati. Ako se želi šifrirati dulji ključ sesije ili složenija poruka s RSA-om, to bi jednostavno uključivalo puno veći broj [11].

4. Shorov algoritam

Peter Shor je profesor primijenjene matematike od 2003. godine, a predsjednik Odbora za primijenjenu matematiku od 2015. Završio je B.A. matematiku iz Caltech-a 1981., a doktorat u primijenjenoj matematici s MITa (Massachusetts Institute of Technology) 1985., pod vodstvom Toma Leightona. Nakon postdoktorske stipendije u MSRI (Mathematical Sciences Research Institute), pridružio se AT&T-u. Istraživačka zanimanja profesora Shora su područja iz teorijske informatike: algoritmi, kvantno računanje, računska geometrija i kombinatorika. Peter Shor također je dobitnik brojnih nagrada poput međunarodne nagrade za kvantno komuniciranje, Dickson-ove nagrade za znanost, Gödlove nagrade ACMa (Association for Computing Machinery). Član je Nacionalne akademije znanosti (2002) i suradnik Američke akademije umjetnosti i znanosti (2011). Nadalje, profesor Shor je 2017. godine primio Dirac medalju Međunarodnog centra za teorijsku fiziku te je 2018. dobio nagradu IEEE (Institute of Electrical and Electronics Engineers) Eric E. Sumner za izvanredni doprinos komunikacijskoj tehnologiji. Također je primio nagradu Micius Quantum 2019. godine. Algoritam za kvantna računala koji se obrađuje u ovom završnom radu osmislio je 1994. godine i služi za određivanje prostih faktora za određeni broj N [12].

Pojam algoritam se definira kao postupak ili formula za rješenje problema, temeljen na provođenju slijeda određenih radnji. Računalni program može se promatrati kao složeni algoritam. U matematici i računarskoj znanosti algoritam obično znači mali postupak koji rješava ponavljajući problem. Algoritmi se široko koriste u svim područjima informatičke tehnologije. Na primjer, algoritam tražilice kao niz podataka uzima nizove ključnih riječi i operatera u pretraživačkoj mreži, traži povezanu bazu podataka za relevantne web stranice i vraća rezultate. Drugi primjer upotrebe algoritama je za šifriranje gdje on pretvara podatke prema unaprijed definiranim koracima radi zaštite. Algoritam tajnih ključeva, kao što je, na primjer „Standard za šifriranje podataka Ministarstva obrane Sjedinjenih Država“ znan pod kraticom DES, koristi isti ključ za šifriranje i dešifriranje podataka što je u redu sve dok je algoritam dovoljno sofisticiran jer su tada šanse da netko tko nema ključ dešifrira podatke izuzetno male [13].

4.1. Temeljna zamisao algoritma

Temeljna zamisao Shorovog algoritma je upotrijebiti svojstva i način rada kvantnih računala poput superpozicije, zapletenosti i kvantne Fourierove transformacije za pronalazak prostih faktora nekog cijelog broja N.

Za objašnjavanje superpozicije može se zamisliti situacija kada bi se mogao baciti novčić koji neće pasti samo u položaj glave već i u položaj pisma i to istovremeno. Ipak postoji kvaka; onog trenutka kada se pogleda taj kvantni novčić, prisiljen je nasumično uzeti stanje ili glave ili pisma (vjerojatnost za oba stanja je 50% jer se bira jedno stanje od moguća dva) pri čemu promatrač nikad neće znati u kojem se položaju novčić nalazio. To je jedan od razloga zbog čega se mora biti oprezan pri mjerenju qubita, jer se mijenjaju čim ih se promatra. Sve u svemu, superpozicija mijenja „igru“ na posve novi način. Prednosti ove pojave mogu se prikazati i na sljedeći način:

- klasično računalo s jednim bitom može biti (ili pohraniti) u 1 od 2 stanja odjednom: 0 ili 1. Kvantno računalo s 1 qubitom može biti (ili pohraniti) u 2 stanja odjednom. To je $2^1 = 2$,
- klasično računalo s dva bita može pohraniti samo 1 od $2^2 = 4$ moguće kombinacije. Kvantno računalo s 2 qubita može pohraniti $2^2 = 4$ moguće vrijednosti istodobno.

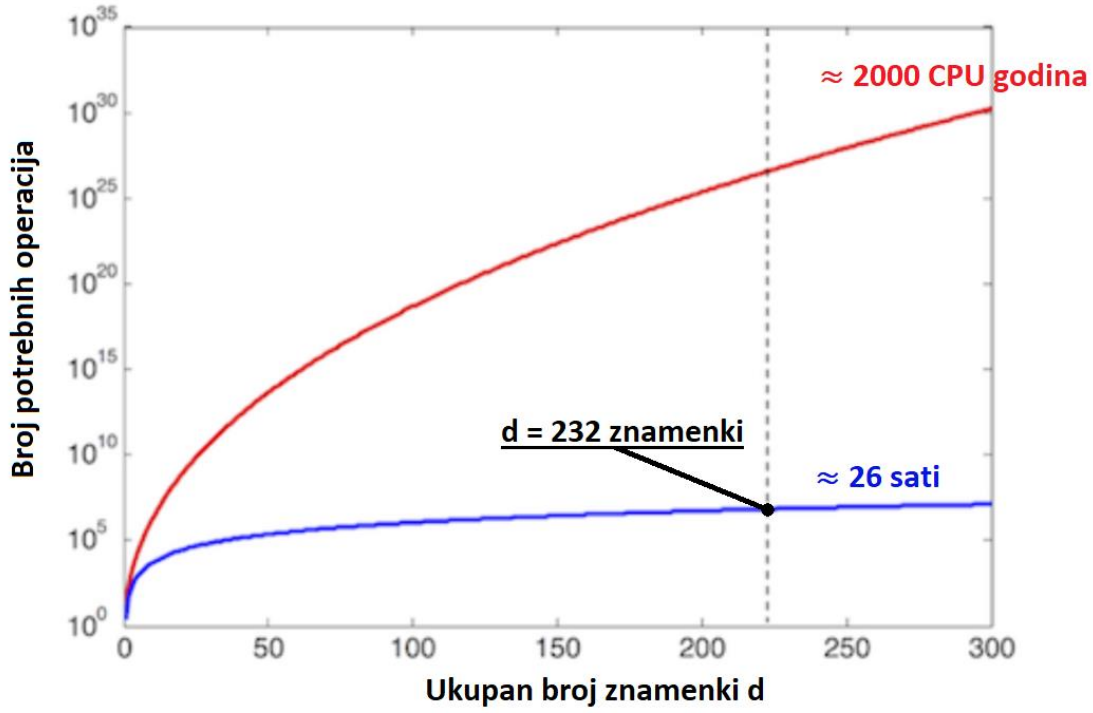
Pod pretpostavkom da je bajt (8 bita) osnovna jedinica koja se koristi za spremanje informacija u bilo kojem sustavu, tada bi broj vrijednosti koji se mogu istovremeno pohraniti u kvantno računalo bio 2^n, gdje je n broj qubita. Usporedi li se to s kapacitetom za pohranu klasičnog računala može se primijetiti zašto su qubiti moćni.

Količina podataka koja se može istovremeno pohraniti u kvantno računalo je zapanjujuća, toliko da je vani iskrsnuo novi pojam: kvantna nadmoć. To je točka u kojoj će kvantno računalo moći riješiti sve probleme koje klasično računalo ne može. Više o ovoj temi bit će prikazano u daljnjem odjeljku ovog poglavlja. A sada se prelazi na sljedeće neobično svojstvo qubita: zapletenost.

Ako se skup qubita zaplete, tada će svaki reagirati na promjenu drugog trenutno, bez obzira koliko su udaljeni. Ovo je korisno jer ako se izmjere svojstva u jednom qubitu, tada se mogu izvući svojstva njegovog partnera bez potrebe za ponovnim pregledom partnera.

Nadalje, zapletenost (engl. entanglement) se može mjeriti bez gledanja kroz proces koji se naziva kvantna tomografija. Kvantna tomografija nastoji odrediti stanje isprepletenog skupa prije mjerenja postupkom mjerenja sustava koji dolaze iz izvora. Drugim riječima, izračunava vjerojatnost mjerenja svakog mogućeg stanja sustava. Zapletenost je eksperimentalno dokazao francuski fizičar Alain Aspect 1982. godine. On je potvrdio kako efekt u jednoj od dvije korelirane čestice putuje brže od brzine svjetlosti. Zapletenost je jedan aspekt manipulacije qubitom. Postoji još jedna manipulacija qubitom putem kvantnih vrata, no ona izlazi iz teme ovoga rada.

Postoji više različitih algoritama koji pouzdano i učinkovito traže proste faktore za određeni broj te savršeno rade na današnjim računalima. Jedan od takvih algoritama je i algoritam sita polja brojeva (engl. number field sieve). Prednost Shorovog algoritma je upravo upotreba brzine proračuna podataka koje omogućuju kvantna računala. Na slici 5 prikazan je graf odnosa brzine traženja faktora između Shorovog algoritma, prikazan plavom bojom, i sita polja brojeva, prikazan crvenom bojom, za neki broj s dužinom od 232 znamenke [2].



Slika 5. Grafički prikaz utrošenog vremena za provođenje operacija za pronalazak prostih brojeva za brojeve duge d = 232 znamenke (preuzeto iz [14])

4.2. Način rada Shorovog algoritma

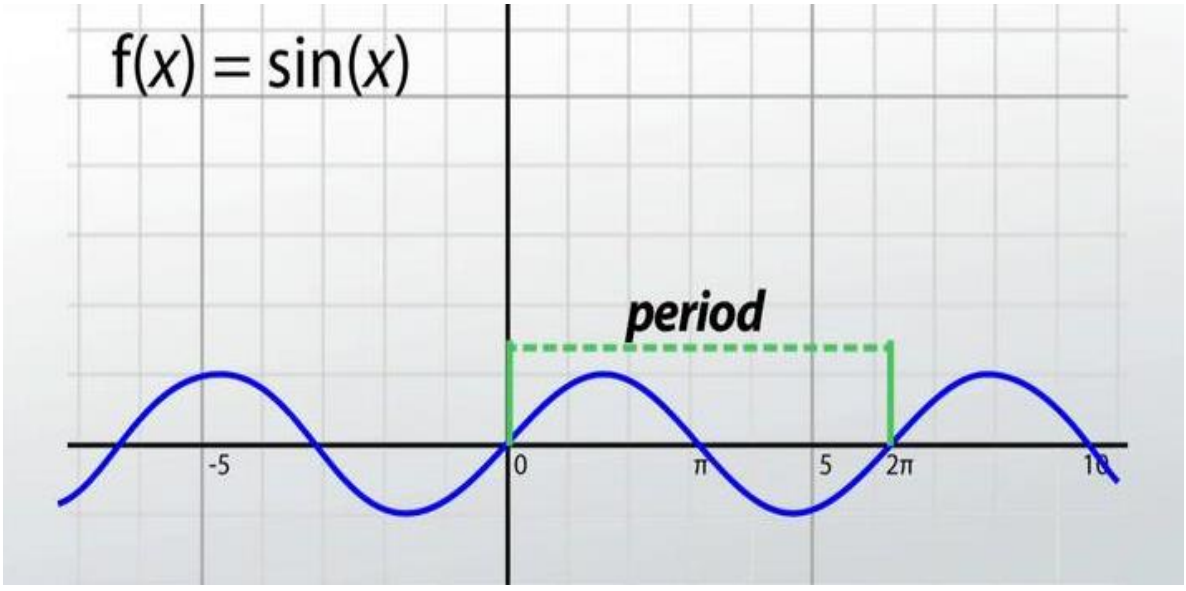
Shorov algoritam se zasniva na izrazu (8) iz teorije brojeva [15].

$$F(a) = x^a \ mod \ n, \tag{8}$$

Gdje oznake imaju sljedeće značenje:

- '$F(a)$' – periodična funkcija u ovisnosti o a,
- 'x' – cijeli ko-prosti broj u odnosu na n; kada su dva broja ko-prosti, to znači da je njihov najveći zajednički djelitelj 1,
- 'a' – cijeli broj koji se povećava za jedan sa svakim pokušajem izračuna vrijednosti funkcije
- n – broj koji se želi rastaviti na faktore.

Periodična funkcija je ona funkcija kojoj se neki period ponavlja točno svakih k intervala vremena. Primjer periodične funkcije je sinus, a prikazan je na slici 6 sa prikazanim periodom.



Slika 6. Prikaz izgleda funkcije sinus(preuzeto iz [16])

Izračunavanje ove funkcije se naziva i modularno eksponenciranje, a za eksponencijalnu količinu brojeva „a“, klasičnom računalu bi trebalo eksponencijalno vrijeme. Shorov algoritam koristi kvantni paralelizam za izvođenje eksponencijalnog broja operacija u jednom koraku.

Kvantni paralelizam proizlazi iz sposobnosti registra kvantne memorije da postoji u superpoziciji baznih stanja. Svaka komponenta ove superpozicije može se smatrati jedinstvenim argumentom neke funkcije. Funkcija izvedena u jednom registru u superpoziciji stanja izvodi se na svakoj od komponenti superpozicije. Budući da je broj mogućih stanja 2^n gdje je n broj qubita u kvantnom registru, u jednoj operaciji na kvantnom računalu može se izvesti ono što bi trebalo eksponencijalni broj operacija na klasičnom računalu. To je fantastično, ali što je više superponiranih stanja koja postoje u registru, to je manja vjerojatnost da će se mjeriti točno određeno stanje.

Kao primjer može se pretpostaviti da se koristi kvantno računalo za izračunavanje funkcije F, gdje je „x" vrijednost superpozicije unutar skupa $x \in \{0, 1, 2, 3, 4, 5, 6, 7\}$. Kvantni registar bi bio u podjednakom ponderiranom stanju stanja skupa {0, 1, 2, 3, 4, 5, 6, 7}. Tada bi se moglo jednom izvršiti operaciju 2x mod 7, a registar bi sadržavao jednako ponderiranu superpoziciju 0,2,4,6,1,3,5,0 stanja, a to su izlazi funkcije 2x mod 7 za ulaze iz skupa {0, 1, 2, 3, 4, 5, 6, 7}. Prilikom mjerenja kvantnog registra šanse za mjerenje 0 iznosile bi 2/8, dok bi šanse za mjerenje bilo kojeg drugog izlaza iznosile 1/8. Čini se da takva vrsta paralelizma nije korisna, jer što se više koristi paralelizam to je manja vjerojatnost da će se mjeriti vrijednost funkcije za određeni ulaz. Peter Shor je osmislio algoritam koji uspijeva koristiti kvantni paralelizam na funkciji u kojoj postoji interes nekih svojstava svih ulaza, a ne samo pojedinih.

Ova vrsta paralelizma vrlo je privlačna za simulaciju na paralelnom računalu. N bitni kvantni registar sadrži superpoziciju svakog od njegovih 2^n mogućih baznih stanja, a to se predstavlja nizom 2^n složenih brojeva koji su vjerojatni za mjerenje kvantnog registra kao odgovarajuće bazno stanje. Da bi se izvela operacija na kvantnom registru, jednostavno se modificira svaku od lokacija 2^n polja [17].

Razlog zašto je izraz (8) koristan u faktoriranju velikog broja naveden je u nastavku.

Pošto je F(a) periodična funkcija, ima neki period r pa se može zapisati:

$$x^0 \bmod n = 1 \rightarrow x^r \bmod n = 1 \rightarrow x^{2r} \bmod n = 1.$$

Ukoliko se uzme taj zapis te se matematički raspiše, dolazi se do izvoda koji glasi:

$$(x^{\frac{r}{2}})^2 - 1 \equiv 0 \; mod \; n$$

A ako je r paran broj:

$$\left(x^{\frac{r}{2}} - 1\right)\left(x^{\frac{r}{2}} + 1\right) \equiv 0 \; mod \; n.$$

Može se vidjeti da je produkt $\left(x^{\frac{r}{2}} - 1\right)\left(x^{\frac{r}{2}} + 1\right)$ cijeli broj koji je veći od n, a to je broj koji treba faktorirati. Sve dok je $x^{\frac{r}{2}}$ različit od +1 ili -1, tada bar jedan od $\left(x^{\frac{r}{2}} - 1\right), \left(x^{\frac{r}{2}} + 1\right)$ mora imati netrivijalni faktor zajednički s n. Tako će se računanjem najvećeg zajedničkog djelitelja(GCD) od izraza $\mathrm{GCD}\left(x^{\frac{r}{2}} - 1, n\right)$ i $\mathrm{GCD}\left(x^{\frac{r}{2}} + 1, n\right)$ dobiti faktor n.

Shorov algoritam pokušava pronaći r, period od x^a mod n, gdje je n broj koji treba faktorirati po Shorovom algoritmu, a x je cijeli broj ko-prost na n. Da bi se to postiglo, Shorov algoritam kreira registre kvantne memorije s dva dijela. U prvom dijelu algoritam postavlja superpoziciju cijelih brojeva koji trebaju poprimiti vrijednost „a“ u izrazu (8).

Zatim je potrebno odabrati q kao najmanju potenciju broja 2 tako da vrijedi $n^2 \leq q < 2n^2$. Tada algoritam izračunava x^a mod n, gdje je a superpozicija stanja, a rezultat stavlja u drugi dio registra kvantne memorije.

Zatim algoritam mjeri stanje drugog registra, onog koji sadrži superpoziciju svih mogućih rezultata za x^a mod n. Mjerenje ovog registra ima za posljedicu kolaps stanja u neku promatranu vrijednost, na primjer k. Također ima nuspojavu projiciranje prvog dijela kvantnog registra u stanje koje je u skladu s vrijednošću izmjerenom u drugom dijelu. Ako je kvantni registar podijeljen na dva dijela, mjerenje drugog dijela urušava taj dio u točno jednu vrijednost, dok se druga particija urušava u stanje u skladu s promatranom vrijednošću u drugom dijelu registra. Još je moguće da ne izmjereni dio registra postoji u superpoziciji baznih stanja, sve dok su svaka od tih baznih stanja u skladu s izmjerenom vrijednošću u drugom dijelu registra. To znači u ovom slučaju da nakon mjerenja drugi dio registra sadrži vrijednost k, a prvi dio registra sadrži superpoziciju baznih stanja koja se uključe u x^a mod n kako bi se proizveo k. Pošto je x^a mod n periodična funkcija, prvi dio registra sadržavat će vrijednosti c, c+r, c+2r…, gdje je c najniži cijeli broj za koji vrijedi x^c mod n = k.

Sljedeći je korak izvedba diskretne Fourierove transformacije na sadržaju prvog dijela registra (onaj koji sadrži sve cjelobrojne brojeve tako da $x^a \bmod n = k$ i da se rezultat vrati u registar [15].

Kvantna Fourierova transformacija ili Fourierovo uzorkovanje proces je manipulacije podacima koji imaju sljedeća svojstva:

- omogućuje promjenu ulaza bez promjene izlazne distribucije,
- periodičnu superpoziciju u kojoj su nulte amplitude višestruke vrijednosti razdoblja [2].

Poput klasične Fourierove transformacije, kvantna Fourierova transformacija (QFT) preuzima podatke od izvornog predstavljanja signala do reprezentacije u frekvencijskoj domeni. QFT se razlikuje od klasične Fourierove transformacije po tome što djeluje u stanju superpozicije i proizvodi drugačije stanje superpozicije kao izlaz. QFT radi pomoću smetnji koje čine signal jačim ili slabijim. Komponente interferiraju bilo konstruktivno ili destruktivno, ovisno o njihovoj amplitudi i fazi [18].

Ova Fourierova transformacija preslikava funkcije u vremenskoj domeni u funkcije u frekvencijskoj domeni. Primjena diskretne Fourierove transformacije ima učinak maksimalizacije vjerojatnosti amplituda prvog dijela registra pri cijelim brojevima veličine q / r. Mjerenje prvog dijela kvantnog registra donijet će cijeli broj koji je faktor inverznog perioda. Nakon što se ovaj broj izvadi iz registra kvantne memorije, klasično računalo može izvršiti analizu ovog broja, nagađati stvarnu vrijednost r i iz toga izračunati moguće faktore n. Ova će obrada biti detaljnije obrađena kasnije.

Postupak Shorovog algoritma može se opisati kroz jedanaest koraka.

1) Prvi korak je utvrditi je li broj n prost, paran, ili cjelobrojna potencija prostog broja. Ako je to slučaj, ne koristi se Shorov algoritam. Postoje učinkovite klasične metode za utvrđivanje pripada li cijeli broj nekoj od prethodno navedenih. Ovaj se korak izvodi na klasičnom računalu.
2) Drugi korak je odabir cijelog broja q koji je potencija broja 2 tako da zadovoljava uvjet $n^2 \leq q < 2n^2$. Ovaj se korak obavlja na klasičnom računalu.

3) Treći korak je odabir nasumičnog broja x koji je ko-prosti u n. Kada su dva broja ko-prosta to znači da je njihov najveći zajednički djelitelj 1. Postoje učinkovite klasične metode za pronalazak takvoga broja x. Ovaj se korak također izvodi na klasičnom računalu.
4) Četvrti korak je napraviti kvantni registar i podijeliti ga u dva skupa, registar 1 i registar 2. Stoga stanje kvantnog računala može dati vrijednost unutar skupa: | reg1, reg2>. Registar 1 mora imati dovoljno qubita da predstavlja cijele brojeve velike koliko i q - 1. Registar 2 mora imati dovoljno qubita da predstavlja cijele brojeve veličine n - 1. Izračunavanje koliko qubita je potrebno obaviti na klasičnom računalu.
5) Peti korak je učitavanje registra 1 s jednako ponderiranom superpozicijom svih cjelobrojnih brojeva od 0 do q - 1. Učitati registar 2 s nulama. Ovu bi operaciju izvodilo kvantno računalo. Ukupno stanje registra kvantne memorije u ovom trenutku se izražava izrazom (9).

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\,, 0\rangle \tag{9}$$

6) U šestom koraku se primjenjuje transformacija x^a mod n na svaki broj pohranjen u registru 1, te se rezultat pohranjuje u registar 2. Zbog kvantnog paralelizma za to će biti potreban samo jedan korak, jer će kvantno računalo samo izračunati $x^{|a\rangle}$ mod n, gdje je $|a\rangle$ superpozicija stanja stvorena u petom koraku. Ovaj se korak izvodi na kvantnom računalu. Stanje registra kvantne memorije u ovom trenutku izračunava se izrazom (10).

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\,, \mathrm{x}^a \bmod n\rangle \tag{10}$$

7) Sedmi korak je mjerenje drugog registara i promatranje neke vrijednosti k. To ima nuspojavu kolapsa prvog registara u jednaku superpoziciju za svaku vrijednost „a“ između 0 i q-1 (izraz 11).

$$x^a \bmod n = k. \tag{11}$$

Ovu operaciju izvodi kvantno računalo. Stanje registra kvantne memorije nakon ovog koraka je izraženo izrazom (12).

$$\frac{1}{\sqrt{\|A\|}} \sum_{a'=a' \in A} |a', \mathrm{k}> \tag{12}$$

Gdje je:

- 'A' skup svih 'a' za koje vrijedi da je x^a mod n = k,
- '|| A ||' predstavlja broj elemenata u tom skupu.

8) U osmom koraku prelazi se na izračunavanje diskretne Fourierove transformacije nad registrom jedan. Kada se diskretan Fourierov preobražaj primijeni na stanje $|a\rangle$ mijenja ga na sljedeći način čime se dobiva izraz (13).

$$|a\rangle = \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} |c\rangle * e^{\frac{2\pi i a c}{q}} \tag{13}$$

Ovaj korak kvantno računalo izvodi u jednom koraku kroz kvantni paralelizam. Nakon diskretne Fourierove transformacije stanje registara se može opisati sa izrazom (14):

$$\frac{1}{\sqrt{\|A\|}} \sum_{a' \in A} \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} |c, k\rangle * e^{\frac{2\pi i a' c}{q}} \tag{14}$$

9) Deveti korak mjeri stanje jednog registra. Ova vrijednost će biti označena slovom m. Taj cijeli broj m ima vrlo veliku vjerojatnost da će biti višekratnik q / r, gdje je r željeni period. Ovaj korak izvodi kvantno računalo.
10) Deseti korak uzima vrijednost m, a na klasičnom računalu se vrši naknadna obrada koja izračunava r na temelju poznavanja vrijednosti m i q. Mnogo je načina za obradu ove tvrdnje, a ta se obrada vrši na klasičnom računalu.
11) Jedanaesti korak jednom kada se pronađe r, faktor n može se odrediti uzimajući $\mathrm{GCD}\left(x^{\frac{r}{2}} + 1, n\right)$ i $\mathrm{GCD}\left(x^{\frac{r}{2}} - 1, n\right)$. Ako se pronašao faktor od n, algoritam završava, ako ne vraća se na četvrti korak. Ovaj posljednji korak vrši se na klasičnom računalu.

Jedanaesti korak sadrži odredbu što učiniti ako Shorov algoritam ne uspije proizvesti n faktora. Postoji nekoliko razloga zbog kojih Shorov algoritam nije uspio pronaći proste faktore, primjerice, diskretna Fourierova transformacija može se mjeriti kao 0 u koraku 9, što onemogućuje naknadnu obradu u koraku 10. Algoritam će ponekad naći faktore 1 i n, što nije korisno. Iz tih razloga korak 11 mora biti u mogućnosti skočiti se na korak četiri kako bi algoritam započeo ispočetka [15].

4.3. Opasnosti implementacije Shorovog algoritma

Ako je razvijen određeni algoritam za računala koja još ne postoje, zašto bi onda taj algoritam trebao biti bitan? Kakvu štetu i prijetnju može uzrokovati? Na prvi pogled, moglo bi se reći, to je samo program koji traži faktore nekog broja.

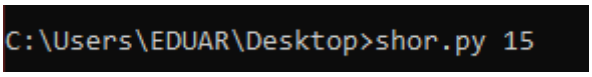
Problem nije u tome koliko je kompleksan Shorov algoritam, već u tome koliko je teško za razbiti RSA algoritam koji je opisan u poglavlju „Kriptografija“ pod potpoglavljem „Kriptografija danas“. RSA je osmišljen i implementiran uzimajući u obzir činjenicu da je teško pronaći proste faktore nekog velikog broja u relativno kratkom vremenskom intervalu. Taj isti RSA algoritam „stoji“ iza https-a (Hypertext transfer protocol secure) čime se svakom korisniku koji pristupa stranici koja je implementirala taj protokol osigurava sigurno pregledavanje stranice, kao i osiguranje da uneseni podaci na stranici ne budu dostupni nikome osim fizičkim ili pravnim osobama koje stoje iza nje. Kada bi se pojavila osoba sa jakim kvantnim računalom, sve što se bazira na RSA algoritmu našlo bi se u opasnosti od krađe bilo koje vrste podataka. Stoga se postavlja pitanje, koji je sljedeći korak u zaštiti podataka kako bi komunikacija kakva postoji danas opstala i u budućnosti?

4.4. Shorov algoritam na klasičnim računalima

Shorov algoritam ne može raditi na klasičnim računalima jer ona ne rade na temelju osnovnih svojstva kvantnih računala poput superpozicije i Fourierovog kvantnog ubrzanja. Shorov algoritam je osmišljen upravo s namjerom da se iskoriste ta kvantna svojstva. Unatoč tome, ljudi su se dosjetili napraviti takozvane kvantne simulatore unutar programa poput Microsoft Visual Studio Community-a za programske jezike kao što je programski jezik C# ili Python. U njima se omogućuje simuliranje kvantnog računala koje može pokretati kvantne algoritme kao da se radi o pravom kvantnom računalu.

5. Analiza slučaja

U ovome poglavlju analizirat će se rad Shorovog algoritma. Algoritam po kojem će se program izvoditi dostupan je na internetskoj lokaciji navedenoj u literaturi [19]. Cjelokupni algoritma koji je malo promijenjen za potrebe ovog rada nalazi se u Dodatku. Navedeni algoritam napravljen je za programski jezik Python, a pokreće se iz CMD, kratice od Command Prompt, aplikacije pozivajući se na lokaciju i ime datoteke nakon čega se mora unijeti vrijednost N, odnosno vrijednost broja koji se želi faktorirati. Primjer je prikazan na slici 7 sa unesenim brojem N = 15.



Slika 7. Primjer pokretanja kôda Shorovog algoritma pod nazivom Shor.py

Unutar kôda koji se pokreće nalaze se mnoge funkcije koje se međusobno pozivaju dok se ne dođe do cilja, dva prosta faktora. Neke od tih funkcija objašnjene su preko pseudokoda. Pseudokod je neformalan način opisivanja programa koji ne zahtijeva strogu sintaksu programskog jezika ili temeljna tehnološka znanja. Koristi se za izradu grubog nacrta programa [20].

5.1. Implementacija kôda

Prilikom pokretanja programa glavna funkcija prvo poziva funkciju „parseArgs“. Unutar programskog jezika Python služi za spremanje vrijednosti različitih varijabli na jedno mjesto u svrhu njihovog kasnijeg lakšeg pozivanja. Funkcija parseArgs tako deklarira parametre i varijable pomoću kojih se upravlja tokom programa. Ti parametri i varijable su sljedeći:

1) „-a“ ili „attempts“ (hrv. pokušaj) – definira se kao broj pokušaja faktoriranja predanog broja. Ukoliko se dođe do maksimalne vrijednosti izvršavanje kôda se obustavlja. Opcionalan je za unos. Mora biti cijeli broj, a predefinirana vrijednost iznosi 20.
2) „-n“ ili „neighborhood“ (hrv. susjedstvo) – definira se kao susjedna veličina za provjeru kandidata. Izražava se kao postotak od N. Opcionalan je za unos. Mora biti

realan broj, a predefinirana vrijednost iznosi 0,01. Ime varijable u pseudokodu glasi „broj_susjeda“.

3) „-p“ ili „periods“ (hrv. periodi) – definira se kao broj dobivenih perioda prije utvrđivanja najmanjeg zajedničkog višekratnika. Opcionalan je za unos. Mora biti cijeli broj, a predefinirana vrijednost iznosi 2.
4) „-v“ ili „verbose“ (hrv. verbalan, govorljiv) – definira želi li se ili ne ispisivati postupak za vrijeme izvršavanja kôda. Opcionalan je za unos. Binarna je varijabla koja poprima vrijednost „Istina“ ili „Laž, a predefinirana vrijednost iznosi "Istina".
5) „N“ – definira se kao broj koji se želi faktorirati. Obavezan je za unos. Mora biti cijeli broj.

5.2. funkcija Shor

Funkcija Shor centralni je dio kôda. Ona će uzeti unesene vrijednosti i parametre, provesti ih kroz svoj kôd, što uključuje i pozivanje drugih funkcija, te vratiti vrijednosti faktora „f1“ i „f2“za unesenu vrijednost N. Pseudokod je prikazan na slici 8.

Algoritam 1 Shor

Input: $N, broj\_susjeda \leftarrow 0.01, broj\_perioda \leftarrow 2, broj\_pokusaja \leftarrow 20$
Output: *faktori* f_1, f_2

if $broj\_bitova(N) > granicna\_vrijednost$ OR $N < 3$ **then**
  $f_1 \leftarrow -1,\ f_2 \leftarrow -1$ # *Nisu pronadeni faktori*
end if
$periodi \leftarrow []$
$broj\_susjeda \leftarrow najmanj\_cijeli\_dio(N \cdot broj\_susjeda) + 1$
for i IN $domet(broj\_pokusaja)$ **do**
  $a \leftarrow odaberi\_slucajan\_broj\_manji\_od(N)$
  while $a < 2$ **do**
    $a \leftarrow odaberi\_slucajan\_broj\_manji\_od(N)$
  end while
  $d \leftarrow najveci\_zajednicki\_djelitelj(a, N)$
  if $d > 1$ **then**
    ponovi_pokusaj # *Faktori su pronadeni slucajno (klasicno)*
  end if
  $r \leftarrow pronadi\_period(a, N)$
  $r \leftarrow provjeri\_period\_najblize\_vrijednosti(a, r, N, broj\_susjeda)$
  if r *prazan* **then**
    ponovi_pokusaj # *Period nije pronaden*
  end if
  if $r \bmod 2 > 0$ **then**
    ponovi_pokusaj # *Period je neparan broj*
  end if
  $d \leftarrow inverzni\_mod(a, najvei\_cijeli\_dio(r/2), N)$
  if $r == 0$ OR $d == (N - 1)$ **then**
    ponovi_pokusaj # *Period je trivialan*
  end if
  dodaj r u periodi
  if $duljina(periodi) < broj\_perioda$ **then**
    ponovi_pokusaj
  end if
  $r \leftarrow 1$
  for *period* IN *periodi* **do**
    $d \leftarrow najveci\_zajednicki\_djelitelj(period, r)$
    $r \leftarrow najvei\_cijeli\_dio(r \cdot period/d)$
  end for
  $b \leftarrow inverzni\_mod(a, najvei\_cijeli\_dio(r/2), N)$
  $f_1 \leftarrow najveci\_zajednicki\_djelitelj(N, b + 1)$
  $f_2 \leftarrow najveci\_zajednicki\_djelitelj(N, b - 1)$
  vrati f_1, f_2
end for
vrati praznu vrijednost # *Vratit ce praznu vrijednost ukoliko niti jedan if, te niti jedan for nisu vratili vrijednost*

Slika 8. Pseudokod za funkciju Shor

U funkciji Shor ugrađene su implementirane funkcije programskog jezika Python. Te funkcije su:

1) Najmanji_cijeli_dio – služi za transformaciju broja iz domene realnih brojeva u prvi najmanji broj iz domene cijelih brojeva, neovisno o veličini decimale. To znači da bi broj 10.95 postao 10, dok bi broj -101,45 postao -102. Funkcija se poziva naredbom „math.floor(x)“ u kojoj „x“ simbolizira broj koji se želi zaokružiti.
2) Nasumićan_broj (engl. random number) – služi za dobivanje realne vrijednosti koja je član skupa unutar intervala [0, 1⟩. Funkcija se poziva naredbom: „random. random()“.
3) Domet – (engl. range) služi za dohvaćanje skupa cijelih brojeva od x do. Ova se funkcija poziva naredbom: „range(x, y)“.
4) Dodaj (engl. append) – služi za „lijepljenje“ podataka iz jedne liste, na primjer liste s nazivom „b“, na podatke u drugu listu, na primjer s nazivom „a“. Funkcija se poziva naredbom: „a.append(b)“.
5) Duljina (engl. length) – služi za izvlačenje broja elemenata liste. Na primjer, inicijalizirana je lista „a = [1, 2, 3, 4, 5]“. Funkcija „duljina“ poziva se naredbom len(a), a vratit će vrijednost per(5) jer lista „a“ ima pet elemenata u listi.

Funkcija Shor vrši postupak koji se može opisati kroz sljedeće korake:

1) Provjerava se istinitost uvjeta:
    a. dužinu vrijednosti „N“ u bitovima veća je od definirane granične vrijednosti granicna_vrijednost,
    b. vrijednost „N“ manja je od 3.

    Ukoliko uvjet nije ispunjen, Shorov algoritam se prekida i nisu pronađeni faktori.

2) varijabla periodi inicijalizira se kao prazna lista,
3) u varijablu broj_susjeda pohranjuje se rezultat funkcije najmanji_cijeli_dio uvećan za jedan,
4) pokreće for petlju dok se ne odabere vrijednost varijable „a“ koja zadovoljava uvjet da je „a“ veći od 2,
5) u varijablu „d“ pohranjuje se rezultat funkcije najveci_zajednicki_djelitelj,

6) provjerava se uvjet „d“ veći od jedan (ukoliko je istinit, funkcija Shor vraća se na prvi korak),
7) u varijablu r pohranjuje se rezultat funkcije pronadi_period,
8) u varijablu r pohranjuje se rezultat funkcije provjeri_period_najblize_vrijednosti,
9) provjerava ima li r unesenu vrijednost (ukoliko je nema, funkcija Shor vraća se na prvi korak),
10) provjerava se parnost vrijednosti r (ukoliko je r neparan, funkcija Shor vraća se na prvi korak),
11) u varijablu „d“ pohranjuje se rezultat funkcije inverzni mod,
12) provjerava se istinitost uvjeta r jednak nula ili d jednak „N“ umanjen za jedan (ukoliko je istinit, funkcija Shor vraća se na prvi korak),
13) vrijednost r pohranjuje se u listu „periodi“,
14) provjerava se istinitost tvrdnje da je rezultat funkcije duljina manji od broja_perioda (ukoliko je istinit, funkcija Shor vraća se na prvi korak),
15) u varijablu r pohranjuje se vrijednost jedan,
16) ako je period unutar varijable periodi tada se izvršavaju koraci:
    a. u varijablu „d“ pohranjuje se rezultat funkcije najveci_zajednicki_djelitelj,
    b. u varijablu „r“ pohranjuje se rezultat funkcije najveci_cijeli_dio,
17) u varijablu „b“ pohranjuje se rezultat funkcije inverzni_mod,
18) u varijablu „f1“ pohranjuje se rezultat funkcije najveci_zajednicki_djelitelj,
19) u varijablu „f2“ pohranjuje se rezultat funkcije najveci_zajednicki_djelitelj.

5.3. Pronalazak slučajnog broja

Sljedeća funkcija na koju funkcija Shor nailazi služi za pronalazak slučajnog broja za koji vrijedi da je član podskupa cijelih brojeva [0, $N\rangle$. Realizacija takvog kôda prikazana je na slici 9.

Algoritam 2 odaberi_slucajan_broj_manji_od

Input: N
Output: a
1: $a \leftarrow najmanji\_cijeli\_dio(((nasumican\_broj()) \cdot (N-1)) + 0.5)$
2: $vrati\ a$

Slika 9. Pseudokod za odabir nasumičnog broja

Iz pseudokoda sa slike 9 vidljivo je da će se u varijablu „a“ spremiti rezultat. Rezultat će ovisiti o dvije stvari; veličini varijable „N“ koja definira gornju graničnu vrijednost, te o veličini koju računalo vrati funkcijom nasumican_broj.

5.4. Najveći zajednički djelitelj

Funkcija najveći_zajednički_djelitelj traži najveći zajednički djelitelj od unesenih vrijednosti. Vrijednosti ulaza navode se nakon imena varijable, unutar zagrada te su odvojene zarezom. Pseudokod koji prikazuje proces traženja djelitelja prikazan je na slici 10.

Algoritam 3 Najveci_zajednicki_djelitelj

Input: a, b
Output: a
1: **while** $b! = 0$ **do**
2: $tA \leftarrow a \mod b$
3: $a \leftarrow b$
4: $b \leftarrow tA$
5: *vrati* a
6: **end while**

Slika 10. Pseudokod za pronalazak najvećeg zajedničkog djelitelja

Prilikom zadnjeg koraka u while (hrv. dok) petlji, u varijablu „a“ spremit će se vrijednost najvećeg djelitelja, a u varijablu „b“ ostatak cjelobrojnog dijeljenja koji će iznositi nula. Nakon toga kôd neće ponovno ući u while petlju jer neće biti zadovoljen uvjet da je nula različita od nule nakon čega će funkcija vratiti vrijednost a.

5.5. Pronalazak perioda

Za funkciju „Pronađi_period“ moglo bi se reći da je jedna od najvažnijih funkcija koje funkcija Shor poziva. Razlog tomu je što je ova funkcija prva koja koristi kvantne komponente. Da bi ih mogla koristiti na klasičnom računalu, ova funkcija će pozivati druge funkcije poput funkcije „map“ (hrv. karta), i „qft_ulazni_registar“ koje služe za simulaciju kvantnog okruženja čime se omogućava implementacija Shorovog algoritma na klasičnim računalima. Pseudokod funkcije „Pronađi_period“ prikazan je na slici 11.

Algoritam 4 Pronadi_period

Input: a, N
Output: r

```
1: n_broj_bitova ← broj_bitova_od(N)
2: ulazni_broj_bitova ← (2 * n_broj_bitova) − 1
3: if ((1 << ulazni_broj_bitova) < (N * N)) then
4:     ulazni_broj_bitova ← ulazni_broj_bitova + 1
5:
6:     Q ← 1 << ulazni_broj_bitova
7:
8:     ulazni_registar ← Qubit_registar(ulazni_broj_bitova)
9:     hmd_ulazni_registar ← Qubit_registar(ulazni_broj_bitova)
10:    qft_ulazni_registar ← Qubit_registar(ulazni_broj_bitova)
11:    izlazni_registar ← Qubit_registar(ulazni_broj_bitova)
12:
13:    ulazni_registar.map(hmd_ulazni_registar, lambda x : hadamard(x, Q), Laz)
14:    hmd_ulazni_registar.map(qft_izlazni_registar, lambda x : qModExp(a, x, N), Laz)
15:    hmd_ulazni_registar.map(qft_ulazni_registar, lambda x : qft(x, Q), Laz)
16:
17:    ulazni_registar.propagate()
18:    y ← izlazni_registar.measure()
19:    x ← qft_ulazni_registar.measure()
20:
21:    r ← cf(x, Q, N)
22:
23:    vrati r
24: end if
```

Slika 11. Pseudokod za pronalazak perioda

Ova funkcija, uz pomoć drugih implementiranih funkcija vrši postupak koji se može opisati kroz sljedeće korake:

1) generiranje registara,
2) izrada cjelovite Hadamardove matrice,
3) izvršiti modularno eksponenciranje,
4) izvršiti kvantnu Fourierovu transformaciju,
5) izmjeriti izlazni registar,
6) izmjeriti QFT registar,
7) pronaći period putem kontinuiranih razlomaka.

Tek kada prođe sve navedene korake program dobiva moguću vrijednost perioda. Taj period će se dodatno provjeriti u funkciji „Provjeri_period_najbliže_vrijendosti“, te ukoliko ne

zadovolji uvijete ponovno će se morati proći kroz dobar dio funkcije Shor što uključuje i navedenih sedam koraka funkcije „Pronađi_period“.

5.6. Provjera perioda

Funkcija „Provjeri_period_najbliže_vrijendosti“ (engl. checkCandidates) služi za provjeru susjednih veličina. Pseudokod ove funkcije prikazan je na slici 12.

Algoritam 5 Provjeri_period_najblize_vrijednosti

Input: $a, r, N, broj_susjeda$
Output: tR

```
1: if r prazan then
2:    vrati praznu vrijednost
3: end if
4:
5: Provjera visekratnika
6: for k IN domet(1, broj_susjeda + 2) do
7:    tR ← k · r
8:    if inverzni_mod(a, a, N) == inverzni_mod(a, a + tR, N) then
9:       vrati tR
10:   end if
11: end for
12:
13: Provjera manjeg susjednog broja
14: for tR IN domet(r − broj_susjeda, r) do
15:   if inverzni_mod(a, a, N) == inverzni_mod(a, a + tR, N) then
16:      vrati tR
17:   end if
18: end for
19:
20: Provjera veceg susjednog broja
21: for tR IN domet(r + 1, r + broj_susjeda + 1) do
22:   if inverzni_mod(a, a, N) == inverzni_mod(a, a + tR, N) then
23:      vrati tR
24:   end if
25: end for
26:
27: vrati praznu vrijednost  # Vratit ce praznu vrijednost ukoliko ni if te
    niti jedan for nisu vratili vrijednost
```

Slika 12. Pseudokod za provjeru perioda najbližih vrijednosti

Postupak provjere perioda radi se iz razloga što postoji povećana šansa da se nije pogodila točna vrijednost perioda već da se pronašla ili njegova susjedna vrijednost ili da je rješenje perioda njegov višekratnik.

5.7. Inverzni mod

Funkcija „inverzni_mod“ uzima varijable „a“, „exp“ i „modul“ kako bi izračunala vrijednost „fx“. Programski pristup rješavanju inverznog moda prikazan je pseudokodom na slici 13.

Algoritam 6 inverzni_mod

Input: $a, exp, modul$
Output: fx

```
1: fx ← 1
2: while exp > 0 do
3:    tA ← a mod b
4:    if (exp & 1) == 0 then
5:       fx ← fx · a mod modul
6:    end if
7:    a ← (a · a) mod modul
8:    exp ← exp >> 1
9:    vrati fx
10: end while
```

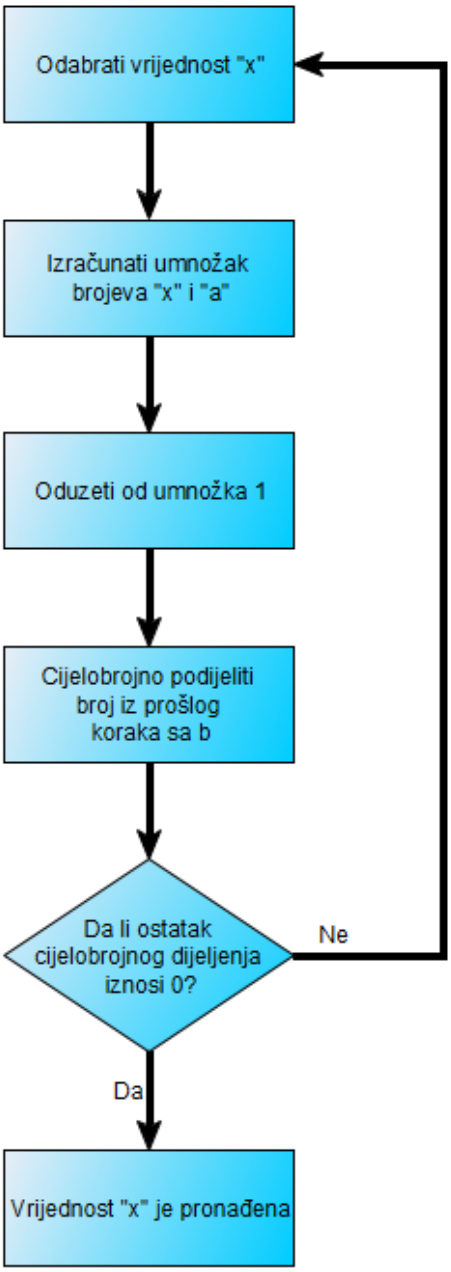
Slika 13. Pseudokod za rješavanje inverznog moda

Osnovni princip pronalaska inverznog modula lakše je razumjeti preko dijagrama toka nego preko programskog jezika. Za pronalazak određene vrijednosti, na primjer „x“ iz izraza (15), pojašnjen je postupak slikom 14 i raspisanim izrazom (15).

$$ax \equiv 1 \pmod{b} \tag{15}$$

Raspisani oblik izraza (15).

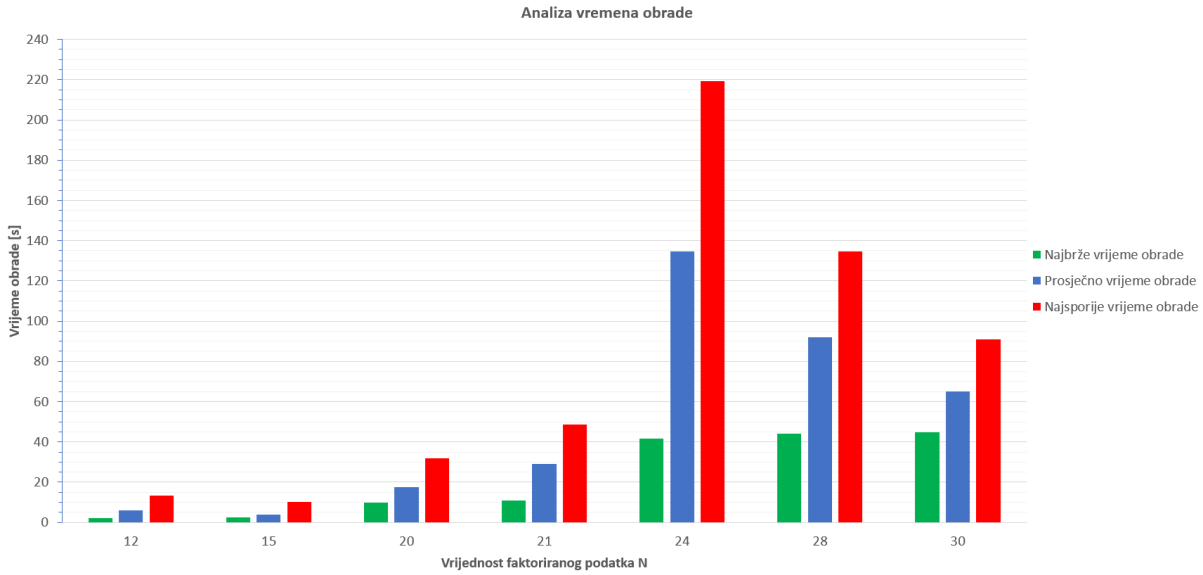
$$(ax - 1) \bmod b == 0$$



Slika 14. Postupak pronalaska vrijednosti x

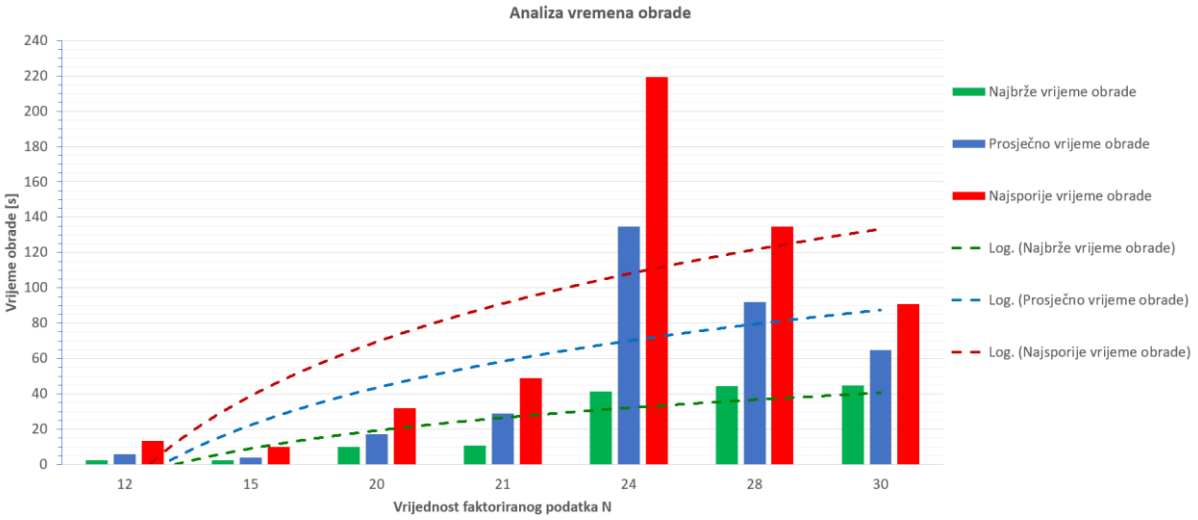
5.8. Rezultati implementacije

Brojevi koje je ovo računalo uspjelo faktorirati su 12, 15, 20,21, 24, 28, i 30. Zanimljivo je primijetiti da je vrijeme faktoriranja broja 24 trajalo najduže, što je vidljivo na slici 15.



Slika 15. Analiza vremena obrade

Slika 15 prikazuje odnos vrijednosti najbržeg vremena obrade (zeleno), prosječnog vremena obrade (plavo) i najsporijeg vremena obrade (crveno). Iz slike 15 primjećuje se da za vrijednost N = 24 prosječno vrijeme obrade i najsporije vrijeme obrade dostiže najveću vrijednost. To je neobično uzme li se u obzir grafikon sa slike 16 sa kojeg je vidljivo da se trend rasta povećava s veličinom vrijednosti podataka, odnosno vrijednosti veće od 24 trebale bi u prosjeku imati nešto duže vrijeme obrade.



Slika 16. Analiza trendova rasta vremena obrade

Pritom treba uzeti u obzir da kôd dolazi do vrijednosti rješenja korištenjem nasumično odabranog broja, što može bitno utjecati na duljinu obrade podatka ukoliko računalu treba više pokušaja da pogodi pravi broj. Ograničenje na koje se nailazi prilikom ispitivanja

različitih vrijednosti ne krije se u kôdu, već u snazi računala na kojem se kôd pokreće. Tako za računalo sa Intel i5 4460 procesorom koji radi na 3,2 GHz (Giga herc) i 8 GB (Gigabyte) RAM memorije, kratice od Random-Access Memory (hrv. memorija s nasumičnim pristupom), vrijedi da ne može provesti proračun s ovim kôdom za brojeve iz domene $[33, \infty\rangle$ zbog ograničenja u memoriji. Ove vrijednosti poklapaju se sa tvrdnjama o brzini obrade podataka koji su vidljivi na slici 5. Grafovi i vrijednosti na grafovima su produkt 10 individualnih mjerenja s identičnim vrijednostima ulaza broja N.

6. Zaključak

Kriptografija je znanost koja se bavi skrivanjem korisne informacije podatka primjenom složenih algoritama i ključeva kojima se informacije zaključavaju. Današnji razvoj kriptografije je brži nego ikada prije ponajviše zbog činjenice da se kriptografske metode poput RSA kriptografije koriste za zaštitu privatnih informacija na internetu. Razvojem razumijevanja kvantne mehanike ljudska vrsta nikada prije nije bila bliže razvoju kvantnog računala nego što je danas. Tu na scenu stupa Shorov algoritam. To je algoritam koji upotrebom svojstva kvantne mehanike uspijeva u relativno kratkom vremenskom intervalu izvršiti procese koje klasično računalo ne bi izvršilo unutar razdoblja ljudskog životnog vijeka. Svrha tog algoritma je pronaći proste faktore nekog velikog broja. S obzirom da ih može naći u relativno kratkom vremenskom intervalu, Shorov algoritam na jakom kvantnom računalu predstavljao bi potencijalno veliku opasanost za svakoga tko šalje povjerljive podatke putem interneta. U svijetu u kojem se glavni dio komunikacija odvija upravo putem interneta, Shorov algoritam, blago rečeno, uveo bi veliki nemir. To je zbog činjenice da današnje sigurnije internetske stranice koriste RSA kriptografiju za garantiranje sigurnosti. Ovaj završni rad prikazao je da se primjenom Shorovog algoritma na kvantnim računalima u relativno kratkom vremenu mogu odrediti prosti faktori velikih brojeva, što ugrožava sigurnost informacija kriptiranim upravo RSA algoritmom. Kao i mnogi drugi izumi na Zemlji, činjenica da se nešto može koristiti u pogrešne svrhe ne znači da je za to i izvorno namijenjen. Shorov algoritam je osmišljen sa ciljem da se ljudski rod usmjeri prema daljnjem razvitku. S razvojem razumijevanja kvantne mehanike i razvojem novih i boljih prototipova kvantnih računala zasigurno će se osmisliti još mnogo algoritama koji će olakšati i/ili ubrzati današnje proračunske procese, sve u svrhu dosezanja novih znanstvenih postignuća.

U ovom je radu ispitan rad kôda Shorovog algoritma u programskom jeziku Python. Analizom je utvrđeno da se kôd može uspješno izvršiti na računalu klasične arhitekture, no za faktoriranje velikih brojeva, kao što su to brojevi koji se koriste u praksi, velikom brzinom, potrebno je kvantno računalo kakvo danas još ne postoji.

Literatura

[1] N. D. Mermin, Lecture notes on quantum computation, Cornell University, 2006 http://www.lassp.cornell.edu/mermin/qcomp/chap1.pdf (pristupljeno: srpanj 2019.)

[2] V. Silva, Practical Quantum Computing for Developers: Programming Quantum Rigs in the Cloud using Python, Quantum Assembly Language and IBM QExperience, Apress Media LLC: Welmoed Spahr, 2018

[3] Š. Anić, N.Klaić, Ž. Domović, Rječnik stranih riječi: Tuđice, posuđenice, izrazi, kratice i fraze, Sani-Plus, 1999

[4] D. Miessler, Encoding vs. Encryption vs. Hashing vs. Obfuscation, 2019, https://danielmiessler.com/study/encoding-encryption-hashing-obfuscation/ (pristupljeno: kolovoz 2019.)

[5] https://searchsecurity.techtarget.com/definition/cryptography (pristupljeno: srpanj 2019.)

[6] http://practicalcryptography.com/ciphers/caesar-cipher/ (pristupljeno: kolovoz 2019.)

[7] https://i.stack.imgur.com/kWOBS.jpg (pristupljeno: kolovoz 2019.)

[8] https://www.ithistory.org/honor-roll/dr-arthur-scherbius (pristupljeno: kolovoz 2019.)

[9] L. Dade, Enigma Emulator, 2019, http://enigma.louisedade.co.uk/howitworks.html (pristupljeno: kolovoz 2019.)

[10] https://www.geeksforgeeks.org/python-program-for-basic-and-extended-euclidean-algorithms-2/, (pristupljeno: kolovoz 2019.)

[11] J. Lake, What is RSA encryption and how does it work?, 2018, https://www.comparitech.com/blog/information-security/rsa-encryption/, (pristupljeno: kolovoz 2019.)

[12] Massachusetts Institute of Technology, Peter Shor biography, https://math.mit.edu/directory/profile.php?pid=247, (pristupljeno: kolovoz 2019.)

[13] https://whatis.techtarget.com/definition/algorithm, (pristupljeno: kolovoz 2019.)

[14] J. Dressel, Quantum Computing: State of Play, Chapman University, 2018, http://oc.acm.org/docs/qc-state-of-play.pdf, (pristupljeno: kolovoz 2019.)

[15] https://www.dhushara.com/book/quantcos/qcompu/shor/s.htm, (pristupljeno: kolovoz 2019.)

[16] Study.com, How to Find the Period of Sine Functions, 2019, https://study.com/academy/lesson/how-to-find-the-period-of-sine-functions.html, (pristupljeno: kolovoz 2019.)

[17] M. Hayward, Quantum Computing, Shor's Algorithm, and Parallelism GitHub Repository, https://quantum-algorithms.herokuapp.com/433/shor-par/node11.html, (pristupljeno : kolovoz 2019.)

[18] I. Styles, Lecture 6: The Quantum Fourier Transform, http://www.cs.bham.ac.uk/internal/courses/intro-mqc/current/lecture06_handout.pdf, (pristupljeno: kolovoz 2019.)

[19] https://github.com/toddwildey/shors-python, (pristupljeno: kolovoz 2019.)

[20] Bennett, Coleman & Co. Ltd., 2019, https://economictimes.indiatimes.com/definition/pseudocode, (pristupljeno: rujan 2019.)

Popis kratica

RSA	(Rivest–Shamir–Adleman)
IBM	(International Business Machines) američka računalna tvrtka, jedna od najrazvijenijih
LCM	(Least common multiple) operator za najniži zajednički višekratnik
ASCII	(American Standard Code for Information Interchange) američki standard za razmjenu informacija
GCD	(Greatest common divisor) predstavlja oznaku za funkciju za pronalazak najvećeg zajedničkog djelitelja
(QFT)	(Quantum Fourier transform) kvantna Fourierova transformacija
HTTPS	(Hypertext transfer protocol secure)
CMD,	(Command Prompt)
RAM	(Random Access Memory) memorija s nasumičnim pristupom

Popis slika

Dodatak

```
1.  #!/usr/bin/env python
2.
3.  """shors.py: Shor's algorithm for quantum integer factorization"""
4.
5.  import math
6.  import random
7.  import argparse
8.  import time
9.
10. def printNone(str):
11.     pass
12.
13. def printVerbose(str):
14.     print(str)
15.
16. printInfo = printNone
17.
18. ##################################################################################
    ##################################
19. #
20. #                                Quantum Components
21. #
22. ##################################################################################
    ##################################
23.
24. class Mapping:
25.     def __init__(self, state, amplitude):
26.             self.state = state
27.             self.amplitude = amplitude
28.
29.
30. class QuantumState:
31.     def __init__(self, amplitude, register):
32.             self.amplitude = amplitude
33.             self.register = register
34.             self.entangled = {}
35.
36.     def entangle(self, fromState, amplitude):
37.             register = fromState.register
38.             entanglement = Mapping(fromState, amplitude)
39.             try:
40.                     self.entangled[register].append(entanglement)
41.             except KeyError:
```

```
42.                         self.entangled[register] = [entanglement]
43.
44.     def entangles(self, register = None):
45.             entangles = 0
46.             if register is None:
47.                     for states in self.entangled.values():
48.                             entangles += len(states)
49.             else:
50.                     entangles = len(self.entangled[register])
51.
52.             return entangles
53.
54.
55. class QubitRegister:
56.     def __init__(self, numBits):
57.             self.numBits = numBits
58.             self.numStates = 1 << numBits
59.             self.entangled = []
60.             self.states = [QuantumState(complex(0.0), self) for x in
    range(self.numStates)]
61.             self.states[0].amplitude = complex(1.0)
62.
63.     def propagate(self, fromRegister = None):
64.             if fromRegister is not None:
65.                     for state in self.states:
66.                             amplitude = complex(0.0)
67.
68.                             try:
69.                                     entangles = state.entangled[fromRegister]
70.                                     for entangle in entangles:
71.                                             amplitude +=
    entangle.state.amplitude * entangle.amplitude
72.
73.                                     state.amplitude = amplitude
74.                             except KeyError:
75.                                     state.amplitude = amplitude
76.
77.             for register in self.entangled:
78.                     if register is fromRegister:
79.                             continue
80.
81.                     register.propagate(self)
82.
83.     # Map will convert any mapping to a unitary tensor given each element v
84.     # returned by the mapping has the property v * v.conjugate() = 1
85.     #
86.     def map(self, toRegister, mapping, propagate = True):
```

```
87.             self.entangled.append(toRegister)
88.             toRegister.entangled.append(self)
89.
90.             # Create the covariant/contravariant representations
91.             mapTensorX = {}
92.             mapTensorY = {}
93.             for x in range(self.numStates):
94.                     mapTensorX[x] = {}
95.                     codomain = mapping(x)
96.                     for element in codomain:
97.                             y = element.state
98.                             mapTensorX[x][y] = element
99.
100.                                    try:
101.                                            mapTensorY[y][x] = element
102.                                    except KeyError:
103.                                            mapTensorY[y] = { x: element }
104.
105.                    # Normalize the mapping:
106.                    def normalize(tensor, p = False):
107.                            lSqrt = math.sqrt
108.                            for vectors in tensor.values():
109.                                    sumProb = 0.0
110.                                    for element in vectors.values():
111.                                            amplitude = element.amplitude
112.                                            sumProb += (amplitude *
    amplitude.conjugate()).real
113.
114.                                    normalized = lSqrt(sumProb)
115.                                    for element in vectors.values():
116.                                            element.amplitude =
    element.amplitude / normalized
117.
118.                    normalize(mapTensorX)
119.                    normalize(mapTensorY, True)
120.
121.                    # Entangle the registers
122.                    for x, yStates in mapTensorX.items():
123.                            for y, element in yStates.items():
124.                                    amplitude = element.amplitude
125.                                    toState = toRegister.states[y]
126.                                    fromState = self.states[x]
127.                                    toState.entangle(fromState, amplitude)
128.                                    fromState.entangle(toState,
    amplitude.conjugate())
129.
130.                    if propagate:
```

```
131.                                    toRegister.propagate(self)
132.
133.            def measure(self):
134.                    measure = random.random()
135.                    sumProb = 0.0
136.
137.                    # Pick a state
138.                    finalX = None
139.                    finalState = None
140.                    for x, state in enumerate(self.states):
141.                            amplitude = state.amplitude
142.                            sumProb += (amplitude * amplitude.conjugate()).real
143.
144.                            if sumProb > measure:
145.                                    finalState = state
146.                                    finalX = x
147.                                    break
148.
149.                    # If state was found, update the system
150.                    if finalState is not None:
151.                            for state in self.states:
152.                                    state.amplitude = complex(0.0)
153.
154.                            finalState.amplitude = complex(1.0)
155.                            self.propagate()
156.
157.                    return finalX
158.
159.            def entangles(self, register = None):
160.                    entangles = 0
161.                    for state in self.states:
162.                            entangles += state.entangles(None)
163.
164.                    return entangles
165.
166.            def amplitudes(self):
167.                    amplitudes = []
168.                    for state in self.states:
169.                            amplitudes.append(state.amplitude)
170.
171.                    return amplitudes
172.
173.    def printEntangles(register):
174.            printInfo("Entagles: " + str(register.entangles()))
175.
176.    def printAmplitudes(register):
177.            amplitudes = register.amplitudes()
```

```
178.            for x, amplitude in enumerate(amplitudes):
179.                    printInfo('State #' + str(x) + '\'s amplitude: ' + str(amplitude))
180.
181.  def hadamard(x, Q):
182.            codomain = []
183.            for y in range(Q):
184.                    amplitude = complex(pow(-1.0, bitCount(x & y) & 1))
185.                    codomain.append(Mapping(y, amplitude))
186.
187.            return  codomain
188.
189.  # Quantum Modular Exponentiation
190.  def qModExp(a, exp, mod):
191.            state = modExp(a, exp, mod)
192.            amplitude = complex(1.0)
193.            return [Mapping(state, amplitude)]
194.
195.  # Quantum Fourier Transform
196.  def qft(x, Q):
197.            fQ = float(Q)
198.            k = -2.0 * math.pi
199.            codomain = []
200.
201.            for y in range(Q):
202.                    theta = (k * float((x * y) % Q)) / fQ
203.                    amplitude = complex(math.cos(theta), math.sin(theta))
204.                    codomain.append(Mapping(y, amplitude))
205.
206.            return codomain
207.
208.  def findPeriod(a, N):
209.            nNumBits = N.bit_length()
210.            inputNumBits = (2 * nNumBits) - 1
211.            inputNumBits += 1 if ((1 << inputNumBits) < (N * N)) else 0
212.            Q = 1 << inputNumBits
213.
214.            printInfo("Finding the period...")
215.            printInfo("Q = " + str(Q) + "\ta = " + str(a))
216.
217.            inputRegister = QubitRegister(inputNumBits)
218.            hmdInputRegister = QubitRegister(inputNumBits)
219.            qftInputRegister = QubitRegister(inputNumBits)
220.            outputRegister = QubitRegister(inputNumBits)
221.
222.            printInfo("Registers generated")
223.            printInfo("Performing Hadamard on input register")
224.
```

```
225.            inputRegister.map(hmdInputRegister, lambda x: hadamard(x, Q), False)
226.            # inputRegister.hadamard(False)
227.
228.            printInfo("Hadamard complete")
229.            printInfo("Mapping input register to output register, where f(x) is a^x
    mod N")
230.
231.            hmdInputRegister.map(outputRegister, lambda x: qModExp(a, x, N),
    False)
232.
233.            printInfo("Modular exponentiation complete")
234.            printInfo("Performing quantum Fourier transform on output register")
235.
236.            hmdInputRegister.map(qftInputRegister, lambda x: qft(x, Q), False)
237.            inputRegister.propagate()
238.
239.            printInfo("Quantum Fourier transform complete")
240.            printInfo("Performing a measurement on the output register")
241.
242.            y = outputRegister.measure()
243.
244.            printInfo("Output register measured\ty = " + str(y))
245.
246.            # Interesting to watch - simply uncomment
247.            # printAmplitudes(inputRegister)
248.            # printAmplitudes(qftInputRegister)
249.            # printAmplitudes(outputRegister)
250.            # printEntangles(inputRegister)
251.
252.            printInfo("Performing a measurement on the periodicity register")
253.
254.            x = qftInputRegister.measure()
255.
256.            printInfo("QFT register measured\tx = " + str(x))
257.
258.            if x is None:
259.                    return None
260.
261.            printInfo("Finding the period via continued fractions")
262.
263.            r = cf(x, Q, N)
264.
265.            printInfo("Candidate period\tr = " + str(r))
266.
267.            return r
268.
```

```
269. ##########################################################################
    ##################################
270. #
271. #                          Classical Components
272. #
273. ##########################################################################
    ##################################
274.
275. BIT_LIMIT = 12
276.
277. def bitCount(x):
278.         sumBits = 0
279.         while x > 0:
280.                 sumBits += x & 1
281.                 x >>= 1
282.
283.         return sumBits
284.
285. # Greatest Common Divisor
286. def gcd(a, b):
287.         while b != 0:
288.                 tA = a % b
289.                 a = b
290.                 b = tA
291.
292.         return a
293.
294. # Extended Euclidean
295. def extendedGCD(a, b):
296.         fractions = []
297.         while b != 0:
298.                 fractions.append(a // b)
299.                 tA = a % b
300.                 a = b
301.                 b = tA
302.
303.         return fractions
304.
305. # Continued Fractions
306. def cf(y, Q, N):
307.         fractions = extendedGCD(y, Q)
308.         depth = 2
309.
310.         def partial(fractions, depth):
311.                 c = 0
312.                 r = 1
313.
```

```
                        for i in reversed(range(depth)):
                                tR = fractions[i] * r + c
                                c = r
                                r = tR

                        return c

        r = 0
        for d in range(depth, len(fractions) + 1):
                tR = partial(fractions, d)
                if tR == r or tR >= N:
                        return r

                r = tR

        return r

# Modular Exponentiation
def modExp(a, exp, mod):
        fx = 1
        while exp > 0:
                if (exp & 1) == 1:
                        fx = fx * a % mod
                a = (a * a) % mod
                exp = exp >> 1

        return fx

def pick(N):
        a = math.floor((random.random() * (N - 1)) + 0.5)
        return a

def checkCandidates(a, r, N, neighborhood):
        if r is None:
                return None

        # Check multiples
        for k in range(1, neighborhood + 2):
                tR = k * r
                if modExp(a, a, N) == modExp(a, a + tR, N):
                        return tR

        # Check lower neighborhood
        for tR in range(r - neighborhood, r):
                if modExp(a, a, N) == modExp(a, a + tR, N):
                        return tR

```

```
361.            # Check upper neigborhood
362.            for tR in range(r + 1, r + neighborhood + 1):
363.                    if modExp(a, a, N) == modExp(a, a + tR, N):
364.                            return tR
365.
366.            return None
367.
368.  def shors(N, attempts = 1, neighborhood = 0.0, numPeriods = 1):
369.            if(N.bit_length() > BIT_LIMIT or N < 3):
370.                    return False
371.
372.            periods = []
373.            neighborhood = math.floor(N * neighborhood) + 1
374.
375.            printInfo("N = " + str(N))
376.            printInfo("Neighborhood = " + str(neighborhood))
377.            printInfo("Number of periods = " + str(numPeriods))
378.
379.            for attempt in range(attempts):
380.                    printInfo("\nAttempt #" + str(attempt))
381.
382.                    a = pick(N)
383.                    while a < 2:
384.                            a = pick(N)
385.
386.                    d = gcd(a, N)
387.                    if d > 1:
388.                            printInfo("Found factors classically, re-attempt")
389.                            continue
390.
391.                    r = findPeriod(a, N)
392.
393.                    printInfo("Checking candidate period, nearby values, and
    multiples")
394.
395.                    r = checkCandidates(a, r, N, neighborhood)
396.
397.                    if r is None:
398.                            printInfo("Period was not found, re-attempt")
399.                            continue
400.
401.                    if (r % 2) > 0:
402.                            printInfo("Period was odd, re-attempt")
403.                            continue
404.
405.                    d = modExp(a, (r // 2), N)
406.                    if r == 0 or d == (N - 1):
```

```
407.                                    printInfo("Period was trivial, re-attempt")
408.                                    continue
409.
410.                    printInfo("Period found\tr = " + str(r))
411.
412.                    periods.append(r)
413.                    if(len(periods) < numPeriods):
414.                                    continue
415.
416.                    printInfo("\nFinding least common multiple of all periods")
417.
418.                    r = 1
419.                    for period in periods:
420.                                    d = gcd(period, r)
421.                                    r = (r * period) // d
422.
423.                    b = modExp(a, (r // 2), N)
424.                    f1 = gcd(N, b + 1)
425.                    f2 = gcd(N, b - 1)
426.
427.                    return [f1, f2]
428.
429.            return None
430.
431.  ##################################################################################
    ##################################
432.  #
433.  #                       Command-line functionality
434.  #
435.  ##################################################################################
    ##################################
436.
437.  def parseArgs():
438.            parser = argparse.ArgumentParser(description='Simulate Shor\'s
    algorithm for N.')
439.            parser.add_argument('-a', '--attempts', type=int, default=20,
    help='Number of quantum attemtps to perform')
440.            parser.add_argument('-n', '--neighborhood', type=float, default=0.01,
    help='Neighborhood size for checking candidates (as percentage of N)')
441.            parser.add_argument('-p', '--periods', type=int, default=2, help='Number
    of periods to get before determining least common multiple')
442.            parser.add_argument('-v', '--verbose', type=bool, default=True,
    help='Verbose')
443.            parser.add_argument('N', type=int, help='The integer to factor')
444.            return parser.parse_args()
445.
446.  def main():
```

```
447.            """
448.            args = parseArgs()
449.
450.            global printInfo
451.            if args.verbose:
452.                    printInfo = printVerbose
453.            else:
454.                    printInfo = printNone
455.            """
456.
457.            global printInfo
458.            printInfo = printVerbose;
459.            example=[10,12,14,15,18,20,21,22,24,26,28,30];
460.            f=open("Shor_results.txt","w+")
461.            for ex in example:
462.                    print("Working on number"+str(ex))
463.                    f.write("Number to factor:"+str(ex)+";")
464.                    start_time = time.time()
465.                    #factors = shors(args.N, args.attempts, args.neighborhood,
    args.periods)
466.                    factors = shors(ex, 200, 0.3, 2)
467.                    f.write("Time:%s seconds;" %(time.time() - start_time))
468.                    print("--- %s seconds ---" % (time.time() - start_time))
469.                    if factors is None:
470.                            f.write("Factors:None")
471.                    else:
472.                            f.write("Factors:"+ str(factors[0]) + "," + str(factors[1]))
473.                            print("Factors:\t" + str(factors[0]) + ", " +
    str(factors[1]))
474.                    f.write("\n")
475.            f.close()
476.
477.  if __name__ == "__main__":
478.            main()
```

Sveučilište u Zagrebu
Fakultet prometnih znanosti
10000 Zagreb
Vukelićeva 4

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOST

Izjavljujem i svojim potpisom potvrđujem kako je ovaj završni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu što pokazuju korištene bilješke i bibliografija.

Izjavljujem kako nijedan dio rada nije napisan na nedozvoljen način, niti je prepisan iz necitiranog rada, te nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem također, kako nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog rada pod naslovom **SHOROV ALGORITAM ZA KRIPTIRANJE**

na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

Student/ica:

U Zagrebu, 6.9.2019

E. Vajsović

(potpis)