FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**


LABORATORIJSKA VJEŽBA 1

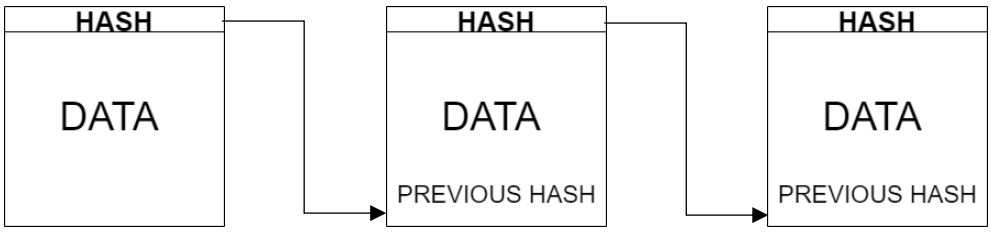
# Blockchain i kriptografija

# Sadržaj

# 1. Uvod

U ovoj laboratorijskoj vježbi baviti ćemo se nekim osnovnim konceptima na kojima počiva blockchain. Započeti ćemo sa osnovnim opisima blockchaina a potom ulaziti dublje u svaki od njih.

Blockchain je tehnologija distribuirane baze podataka koja omogućuje pohranu i prijenos informacija na siguran i transparentan način. Sastoji se od blokova podataka povezanih u lanac, pri čemu svaki blok sadrži transakcije ili druge informacije koje se žele pohraniti. Ovi blokovi su povezani jedan s drugim putem kriptografskih hash funkcija, što osigurava integritet podataka i sprječava manipulaciju. Najjednostavniji prikaz možemo dati slikom 1.




Slika 1. Ulančavanje podataka

Blockchain tehnologija je postala popularna zbog svoje primjene u kriptovalutama kao što je Bitcoin, gdje se koristi za pohranu i prijenos financijskih transakcija. Međutim, blockchain ima širu primjenu, a koristi se u različitim industrijama kao što su financije, zdravstvo, logistika, nekretnine, umjetnost i druge.

Glavna prednost blockchain tehnologije je u tome što omogućuje siguran prijenos podataka bez posrednika, što dovodi do uštede u vremenu i novcu. Također, blockchain je transparentan i njime se jako teško može manipulirati, što osigurava pouzdanost podataka. Paradoks blockchaina je da su pojedinci i organizacije koji si međusobno ne vjeruju stvorili platformu u kojoj si upravo zbog toga i mogu vjerovati. U narednim poglavljima biti će pokazan način kojim je ova pouzdanost osigurana.

# 2. Base58

Base58 je zapisivanje brojeva i slova koje se koristi u blockchain tehnologiji za pretvaranje dužih nizova podataka u kraće, lako čitljive oblike. Base58 je osmislio Satoshi Nakamoto a popularizirana je zahvaljujući Bitcoinu.

U kriptovalutama kao što je Bitcoin, adrese novčanika se obično sastoje od dugih nizova od 26-35 znakova. Ovi nizovi mogu sadržavati slova velikog i malog pisma, kao i brojeve, što ih može učiniti teškim za čitanje i upisivanje. Base58 se koristi za pretvaranje ovih adresa u kraći i jednostavniji oblik koji je lakše prepoznati i upisati.

Base58 se razlikuje od standardne baze 64 (Base64) kodiranja jer izbjegava korištenje slova i brojeva koji izgledaju slično, poput "0" i "O" ili "1" i "I", što može dovesti do pogreške u upisivanju. Umjesto toga, Base58 koristi samo određene znakove koji su lako razlikovati, što povećava pouzdanost i točnost kodiranja.

Znakovi koji se koriste u Base58 su:
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Ukratko, Base58 kodiranje se koristi u blockchainu za pretvaranje dužih nizova podataka, poput adresa novčanika, u kraće i jednostavnije oblike koji su lakši za čitanje i upisivanje.

Kako bismo mogli koristiti Base58 u slijedećim vježbama. Potrebno je znati i kako se pojedini bajtovi preslikavaju u pojedine znakove. Preslikavanje je vidljivo u slijedećoj tablici.

| Bajt | Znak | Bajt | Znak | Bajt | Znak |
|---|---|---|---|---|---|
| 0 | 1 | 20 | M | 40 | h |
| 1 | 2 | 21 | N | 41 | i |
| 2 | 3 | 22 | O | 42 | j |
| 3 | 4 | 23 | P | 43 | k |
| 4 | 5 | 24 | R | 44 | m |
| 5 | 6 | 25 | S | 45 | n |
| 6 | 7 | 26 | T | 46 | o |
| 7 | 8 | 27 | U | 47 | p |
| 8 | 9 | 28 | V | 48 | q |
| 9 | A | 29 | W | 49 | r |
| 10 | B | 30 | X | 50 | s |
| 11 | C | 31 | Y | 51 | t |
| 12 | D | 32 | Z | 52 | u |
| 13 | E | 33 | a | 53 | v |
| 14 | F | 34 | b | 54 | w |
| 15 | G | 35 | c | 55 | x |
| 16 | H | 36 | d | 56 | y |
| 17 | I | 37 | e | 57 | z |
| 18 | J | 38 | f | | |
| 19 | K | 39 | g | | |

Tab.1 Preslikavanje bajta u Base58 znak

Ovo preslikavanje koristi se kod Bitcoina. Postoje druga mapiranja i druge abecede koje ćemo spomenuti po potrebi.

## 2.1 Kodiranje

Satoshi Nakamoto je postavio slijedeći algoritam:

Za kodiranje niza bajtova u Base58 kodiranu vrijednost, pokrenite sljedeći algoritam. Sve matematičke operacije MORAJU se izvoditi koristeći cjelobrojnu aritmetiku. Počnite tako da inicijalizirate 'zero_counter' na nulu (0x0), 'encoding_flag' na nulu (0x0), niz 'b58_bytes', niz 'b58_encoding' i 'carry' vrijednost na nulu (0x0). Za svaki bajt u nizu bajtova i dok 'prenošenje' nije jednako nuli (0x0) nakon prve iteracije:

1. Ako 'encoding_flag' nije postavljena i ako je bajt nula (0x0), povećajte vrijednost 'zero_counter'. Ako vrijednost nije nula (0x0), postavite 'encoding_flag' na true (0x1).
2. Ako je postavljena 'encoding_flag', pomnožite trenutnu vrijednost bajta s 256 i dodajte je u 'carry'.
3. Postavite odgovarajuću vrijednost bajta u 'b58_bytes' na vrijednost 'carry' modula 58.
4. Postavite 'carry' na vrijednost 'carry' podijeljenu s 58.

Nakon što je polje 'b58_bytes' konstruirano, generirajte konačno 'b58_encoding' pomoću sljedećeg algoritma. Postavite prve bajtove 'zero_counter' u 'b58_encoding' na '1'. Zatim, za svaki bajt u 'b58_array', preslikajte vrijednost bajta koristeći Base58 abecedu u prethodnom odjeljku na odgovarajući znak u 'b58_encoding'. Vrati 'b58_encoding' kao Base58 reprezentaciju ulaznog niza bajtova.

### 2.2 Dekodiranje

Sve matematičke operacije MORAJU biti izvedene upotrebom cijelih brojeva. Započnite inicijaliziranjem polja 'raw_bytes' i vrijednosti 'carry' na nulu (0x0). Za svaki ulazni bajt u nizu ulaznih bajtova:

1. Postavite 'carry' na vrijednost bajta pridruženu ulaznom znaku bajta. Ako preslikavanje ne postoji, vrati šifru pogreške.
2. Dok 'carry' nije jednako nuli i ima preostalih ulaznih bajtova:
    a) Pomnožite ulaznu vrijednost bajta s 58 i dodajte je u 'carry'.
    b) Postavite izlaznu vrijednost bajta na 'carry' modul 256.
    c) Postavite 'carry' na vrijednost 'carry' podijeljenu s 256.
3. Postavite odgovarajuću vrijednost bajta u 'raw_bytes' na vrijednost 'carry' modula 58.
4. Postavite 'carry' na vrijednost 'carry' podijeljenu s 58.

## 3. Hash funkcije i SHA-256

Iako je struktura nalik blockchainu predstavljena u znanstvenom radu 1982. godine, kriptografski je zaštićen tek 1991. godine. Satoshi Nakamoto je svoj Bitcoin blockchain zaštitio primjenom SHA-256. Uloga hashiranja je zaštita podataka. Njime se proizvodi podatak točno određene duljine bez obzira na duljinu ulaznog podatka. Rezultat hashiranja jednog jedinog slova će dati identičnu duljinu izlaza kao i hash ove skripte. Također, bilo kakva promjena podataka (npr. Promjena jednog slova u ovoj skripti), dati će veoma uočljivu razliku ukoliko se ponovo hashira.

SHA-256 je algoritam za kriptografsko sažimanje poruka koji se široko koristi u sigurnosnim aplikacijama. SHA-256 je dio SHA-2 (Secure Hash Algorithm 2) skupa kriptografskih funkcija koje je razvio Nacionalni institut za standarde i tehnologiju (NIST) SAD-a.

SHA-256 koristi blokovski algoritam koji može obraditi poruke do veličine od 2^64 bita i proizvesti fiksni izlazni niz duljine 256 bita, bez obzira na veličinu ulazne poruke. Ovaj izlazni niz zove se "sažetak" ili "hash" poruke i obično se koristi za provjeru integriteta podataka ili za potvrdu autentičnosti poruke.

SHA-256 se sastoji od nekoliko koraka, uključujući inicijalizaciju varijabli, obradu blokova podataka, te konačni izračun sažetka. Ovaj algoritam je dizajniran tako da proizvodi jedinstveni i nepredvidljivi sažetak za svaku ulaznu poruku, što ga čini vrlo teško napadnutim i sigurnim za korištenje u raznim sigurnosnim aplikacijama.

SHA-256 algoritam se smatra vrlo otpornim na brute force napade. To je zato što bi napadač morao isprobati sve moguće kombinacije ulaznih poruka kako bi pronašao ulaz koji bi proizveo željeni izlazni sažetak. Međutim, SHA-256 generira sažetak veličine 256 bita, što znači da postoji ogroman broj mogućih kombinacija. Stvaranje identičnog sažetka iz nekoliko različitih poruka praktički je nemoguće, jer bi bilo potrebno isprobati nevjerojatno veliki broj kombinacija.

Uz to, SHA-256 koristi složene matematičke funkcije koje su dizajnirane da otežaju pronalaženje odgovarajuće ulazne poruke na temelju izlaznog sažetka. To čini brute force napade vrlo teškim i dugotrajnim procesom, koji bi trajao milijune ili čak milijarde godina za velike ključeve.

Broj mogućih kombinacija je:

$2^{256} =$
1157920892373161954235709850086879078532699846656405640394575840079131
29584007963129 558400796312951296664039457584007963129129

Ako bi dva podatka dala istu izlaznu vrijednost, taj događaj bismo nazvali „collision“ i on je teoretski moguć ali je njegova vjerojatnost manja nego da ćete nekoliko puta za redom osvojiti Eurojackpot.

Ukratko, SHA-256 je snažan algoritam za kriptografsko sažimanje poruka koji se koristi za sigurno čuvanje i prijenos osjetljivih podataka. On je jedan od najboljih načina za provjeru integriteta podataka i autentičnosti poruka u različitim sigurnosnim aplikacijama.

## 3.1 Algoritam

Korak 1 - Prethodna obrada na primjeru „hello world“

Pretvorite "hello world" u binarni kôd:

01101000 01100101 01101100 01101100 01101111 00100000 01110111

01101111 01110010 01101100 01100100

Dodaj jednu jedinicu - 1:

01101000 01100101 01101100 01101100 01101111 00100000 01110111

01101111 01110010 01101100 01100100 1

Dodajte 0 dok duljina podataka bude višekratnik 512 bita i oduzmite 64 bita (448 bita u našem slučaju):

01101000 01100101 01101100 01101100 01101111 00100000 01110111

01101111 01110010 01101100 01100100 10000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

Dodajte 64 bita na kraj, gdje su 64 bita cijeli broj zapisan u big-endian formatu koji predstavlja duljinu izvornog ulaza u binarnom obliku. U našem slučaju, 88, ili u binarnom obliku, "1011000".

01101000 01100101 01101100 01101100 01101111 00100000 01110111

01101111 01110010 01101100 01100100 10000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000

01011000

Sada imamo svoj unos koji će uvijek biti jednako djeljiv sa 512.

Korak 2 - Inicijalizacija hash vrijednosti (h)

Sada kreiramo 8 hash vrijednosti. To su fiksne konstante koje predstavljaju prva 32 bita razlomaka korijena prvih 8 prostih brojeva: 2, 3, 5, 7, 11, 13, 17, 19.

h0 := 0x6a09e667

h1 := 0xbb67ae85

h2 := 0x3c6ef372

h3 := 0xa54ff53a

h4 := 0x510e527f

h5 := 0x9b05688c

h6 := 0x1f83d9ab

h7 := 0x5be0cd19

Korak 3 - Inicijalizirajte okrugle konstante (k)

Slično koraku 2, stvaramo 64 konstante Svaka vrijednost (0-63) je zapis prva 32 bita razlomaka trećih korijena prvih 64 prostih brojeva (2 - 311).

0x428a2f98 0x71374491 0xb5c0fbcf 0xe9b5dba5 0x3956c25b 0x59f111f1

0x923f82a4 0xab1c5ed5 0xd807aa98 0x12835b01 0x243185be 0x550c7dc3

0x72be5d74 0x80deb1fe 0x9bdc06a7 0xc19bf174 0xe49b69c1 0xefbe4786

0x0fc19dc6 0x240ca1cc 0x2de92c6f 0x4a7484aa 0x5cb0a9dc 0x76f988da

0x983e5152 0xa831c66d 0xb00327c8 0xbf597fc7 0xc6e00bf3 0xd5a79147

0x06ca6351 0x14292967 0x27b70a85 0x2e1b2138 0x4d2c6dfc 0x53380d13

0x650a7354 0x766a0abb 0x81c2c92e 0x92722c85 0xa2bfe8a1 0xa81a664b

0xc24b8b70 0xc76c51a3 0xd192e819 0xd6990624 0xf40e3585 0x106aa070

0x19a4c116 0x1e376c08 0x2748774c 0x34b0bcb5 0x391c0cb3 0x4ed8aa4a

0x5b9cca4f 0x682e6ff3 0x748f82ee 0x78a5636f 0x84c87814 0x8cc70208

0x90befffa 0xa4506ceb 0xbef9a3f7 0xc67178f2

Korak 4 - Chunk petlja

Sljedeći koraci će se dogoditi za svaki 512-bitni "komad" podataka iz našeg ulaza. U našem slučaju, jer je "hello world" tako kratak, imamo samo jedan dio. U svakoj iteraciji petlje, mi ćemo mutirati hash vrijednosti h0-h7, što će biti konačni izlaz.

Korak 5 - Kreirajte raspored poruka (w)

Kopirajte ulazne podatke iz koraka 1 u novi niz gdje je svaki unos 32-bitna riječ:

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000001011000

Dodajte još 48 riječi inicijaliziranih na nulu, tako da imamo niz w[0…63]

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000001011000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

...

...

00000000000000000000000000000000 00000000000000000000000000000000

Izmijenite nulirane indekse na kraju niza koristeći algoritam:

za i u w[16…63]:

s0 := (w[i-15] ror 7) xor (w[i-15] ror 18) xor (w[i-15] >> 3)

s1 := (w[i- 2] ror 17) xor (w[i- 2] ror 19) xor (w[i- 2] >> 10)

w[i] := w[i-16] + s0 + w[i-7] + s1

Primjer za w[16]:

w[1] zakreni desno (ror) 7:

01101111001000000111011101101111 -> 11011110110111100100000011101110

w[1] zakreni desno 18:

01101111001000000111011101101111 -> 00011101110110111101101111001000

w[1] pomak udesno(>>) 3:

01101111001000000111011101101111 -> 00001101111001000000111011101101

s0 = 11011110110111100100000011101110 XOR

00011101110110111101101111001000 XOR 00001101111001000000111011101101

s0 = 11001110111000011001010111001011

w[14] desno zakreni 17:

00000000000000000000000000000000 -> 00000000000000000000000000000000

w[14] desno rotiranje19:

00000000000000000000000000000000 -> 00000000000000000000000000000000

w[14] pomak udesno 10:

00000000000000000000000000000000 -> 00000000000000000000000000000000

s1 = 00000000000000000000000000000000 XOR

00000000000000000000000000000000 XOR 00000000000000000000

s1 = 00000000000000000000000000000000

w[16] = w[0] + s0 + w[9] + s1

w[16] = 01101000011001010110110001101100 +

11001111011100001100101011001011 +

00000000000000000000000000000000

// zbrajanje se izračunava po modulu 2^32

w[16] = 00110111010001110000001000110111

Ovo nam ostavlja 64 riječi u našem rasporedu poruka (w):

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000010

11000001101110100011100000010001 10111100001101101000011000000011

00011101001110111101000100010000 10110111100000111111010001111000

00100010101010010000011111001110 11010100101100101111011111001100

10010011000111100001100101000101 11011000100100110110010010010110

01000111111101111010000011011011 01011000001011110011010100100111

01010111011111010001111011001010 10100001100000110101110001111100

11010110000111111100001101011111 01010111110110111001010101100100

11000000001000100110011011010100 10000001111111000111001010100101

00001110110101111111100101110101 10011010000110010101100010111001

10110010000100111000001010100111 10000001101010111110011011001001

01100100101110111101100100110101 11011000111111100101011110010110

10111000110001011001100111110101 11100001000011101111011111000101

10111011101110110010101000010110 11101000000100111111110010001000

01111110010001100010101101101110 00000101001010100010010010000110

01000100001000010001010011000111 01011000001001001111110000110011

01100000110000001101011111111010 01110101011010111001111001001111

00000111001001111110000101101011 01111110110100101100011011111011

11111010110111010111111111001010 01011010100011011010010101001101

00001000101111110010011100110110 00101010010111010000001101001010

11011000001100111100111000100001 01110001001010110010011000001110

10000100010100001011111101101011 01101100001011000000111111101100

11001100011110000110000110110111 10111001000010111101110000001111

01010001011010100011001111001101 00000000100001111100110010111101

11111110000010111010011110000101 01100001011000010111010110001011

Korak 6 - Kompresija

Inicijalizirajte varijable a, b, c, d, e, f, g, h i postavite ih jednakima trenutnim hash vrijednostima h0,

h1, h2, h3, h4, h5, h6, h7

Pokrenite petlju kompresije. Petlja kompresije će mutirati vrijednosti a…h. Petlja kompresije je sljedeća:

za i od 0 do 63

S1 = (e ror 6) xor (e ror 11) xor (e ror 25)

ch = (e and f) xor ((not e) and g)

temp1 = h + S1 + ch + k[i] + w[i]

S0 = (a ror 2) xor (a ror 13) xor (a ror 22)

maj = (a and b) xor (a and c) xor (b and c)

temp2 := S0 + maj

h = g

g = f

f = e

e = d + temp1

d = c

c = b

b = a

a = temp1 + temp2

Idemo kroz prvu iteraciju, svo se zbrajanje izračunava po modulu $2^{32}$:

a = 0x6a09e667 = 01101010000010011110011001100111

b = 0xbb67ae85 = 10111011011001111010111010000101

c = 0x3c6ef372 = 00111100011011101111001101110010

d = 0xa54ff53a = 10100101010011111111010100111010

e = 0x510e527f = 01010001000011100101001001111111

f = 0x9b05688c = 10011011000001010110100010001100

g = 0x1f83d9ab = 00011111100000111101100110101011

h = 0x5be0cd19 = 01011011111000001100110100011001

e ror 6: 01010001000011100101001001111111 ->

11111101010001000011100101001001

e ror 11: 01010001000011100101001001111111 ->

01001111111010100010000111001010

e ror 25: 01010001000011100101001001111111 ->

10000111001010010011111110101000

S1 = 11111101010001000011100101001001 XOR

01001111111010100010000111001010 XOR 10000111001010010011111110101000

S1 = 00110101100001110010011100101011

e and f:

01010001000011100101001001111111 & 10011011000001010110100010001100 =

00010001000001000100000000001100

not e: 01010001000011100101001001111111 ->

10101110111100011010110110000000

(not e) and g: 10101110111100011010110110000000 &

00011111100000111101100110101011 = 00001110100000011000100110000000

ch = (e and f) xor ((not e) and g) = 00010001000001000100000000001100

xor 00001110100000011000100110000000 =

00011111100001011100100110001100

temp1 = h + S1 + ch + k[i] + w[i]

temp1 = 01011011111000001100110100011001 +

00110101100001110010011100101011 + 00011111100001011100100110001100 +

01000010100010100010111110011000 + 01101000011001010110110001101100

temp1 = 01011011110111010101100111010100

a ror 2: 01101010000010011110011001100111 ->

11011010100000100111100110011001

a ror 13: 01101010000010011110011001100111 ->

00110011001110110101000001001111

a ror 22: 01101010000010011110011001100111 ->

00100111100110011001110110101000

S0 = 11011010100000100111100110011001 XOR

00110011001110110101000001001111 XOR 00100111100110011001110110101000

S0 = 11001110001000001011010001111110

a and b:

01101010000010011110011001100111 & 10111011011001111010111010000101 =

00101010000000011010011000000101

a and c: 01101010000010011110011001100111 &

00111100011011101111001101110010 = 00101000000010001110001001100010

b and c: 10111011011001111010111010000101 &

00111100011011101111001101110010 = 00111000011001101010001000000000

maj = (a and b) xor (a and c) xor (b and c) =

00101010000000011010011000000101 xor 00101000000010001110001001100010

xor 00111000011001101010001000000000 =

00111010011011111110011001100111

temp2 = S0 + maj = 11001110001000001011010001111110 +

00111010011011111110011001100111 = 00001000100100001001101011100101

h = 00011111100000111101100110101011

g = 10011011000001010110100010001100

f = 01010001000011100101001001111111

e = 10100101010011111111010100111010 +

01011011110111010101100111010100 = 00000001001011010100111100001110

d = 00111100011011101111001101110010

c = 10111011011001111010111010000101

b = 01101010000010011110011001100111

a = 01011011110111010101100111010100 +

00001000100100001001101011100101 = 01100100011011011111010010111001

Cijeli se izračun radi još 63 puta, mijenjajući varijable a-h. Nećemo to raditi ručno, ali bismo završili sa:

h0 = 6A09E667 = 01101010000010011110011001100111

h1 = BB67AE85 = 10111011011001111010111010000101

h2 = 3C6EF372 = 00111100011011101111001101110010

h3 = A54FF53A = 10100101010011111111010100111010

h4 = 510E527F = 01010001000011100101001001111111

h5 = 9B05688C = 10011011000001010110100010001100

h6 = 1F83D9AB = 00011111100000111101100110101011

h7 = 5BE0CD19 = 01011011111000001100110100011001

a = 4F434152 = 01001111010000110100000101010010

b = D7E58F83 = 11010111111001011000111110000011

c = 68BF5F65 = 01101000101111110101111101100101

d = 352DB6C0 = 00110101001011011011011011000000

e = 73769D64 = 01110011011101101001110101100100

f = DF4E1862 = 11011111010011100001100001100010

g = 71051E01 = 01110001000001010001111000000001

h = 870F00D0 = 10000111000011110000000011010000

Korak 7 - Izmjena konačnih vrijednosti

Nakon petlje kompresije, ali i dalje, unutar petlje chunk, mijenjamo hash vrijednosti dodajući im odgovarajuće varijable, a-h. Kao i obično, svo zbrajanje je po modulu $2^{32}$.

h0 = h0 + a = 10111001010011010010011110111001

h1 = h1 + b = 10010011010011010011111000001000

h2 = h2 + c = 10100101001011100101001011010111

h3 = h3 + d = 11011010011111011010101111111010

h4 = h4 + e = 11000100100001001110111111100011

h5 = h5 + f = 01111010010100111000000011101110

h6 = h6 + g = 10010000100010001111011110101100

h7 = h7 + h = 11100010111011111100110111101001

Korak 8 - Spojite konačni hash

digest = h0 || h1 ||h2 ||h3 ||h4 ||h5 ||h6 ||h7 =

B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9

## 4. Kriptografija javnog i privatnog ključa

Jedan od osnovnih pojmova kriptografije je tajni i javni ključ. Ovi ključevi se u blockchainu koriste za potpisivanje i verifikaciju transakcija te osiguravanje sigurnosti i privatnosti u mreži.

U blockchainu, svaki korisnik ima svoj par tajnog i javnog ključa koji se koristi za digitalno potpisivanje transakcija. Korisnik koristi svoj tajni ključ za potpisivanje transakcije, a javni ključ se koristi za verifikaciju digitalnog potpisa.

Kada se transakcija stvori, korisnik koristi svoj tajni ključ za generiranje digitalnog potpisa koji se pridodaje transakciji. Nakon toga, transakcija se prenosi u mrežu blockchaina i ostali sudionici u mreži mogu koristiti javni ključ korisnika za verifikaciju digitalnog potpisa i provjeru autentičnosti transakcije.

Tajni ključ korisnika se nikada ne dijeli s drugima, a javni ključ je dostupan svima u mreži blockchaina. Ovo omogućuje privatnost transakcija i osigurava da samo vlasnik tajnog ključa može potpisati transakciju.

Korištenje tajnih i javnih ključeva u blockchainu također omogućuje implementaciju sustava pametnih ugovora koji se izvršavaju automatski bez potrebe za posrednicima ili trećim stranama. Ovo dodatno pojačava sigurnost i transparentnost u blockchainu.

## 4.1 Eliptične krivulje

Eliptičke krivulje su matematički objekti koji se koriste u kriptografiji, uključujući i kriptovalute poput Bitcoina i Ethereuma. Eliptičke krivulje su vrsta krivulja koje imaju svojstvo da se dvije točke na krivulji mogu zbrojiti da bi se dobila treća točka na krivulji.

Ovo svojstvo omogućuje stvaranje sigurnih kriptografskih ključeva za digitalni potpis i enkripciju. U kriptografiji, eliptičke krivulje se koriste za generiranje parova ključeva: javnog i privatnog. Javni ključ je poznat svima i koristi se za enkripciju podataka ili provjeru digitalnog potpisa. Privatni ključ se čuva tajnim i koristi se za dekripciju podataka ili potpisivanje digitalnih transakcija.

Jedna od prednosti korištenja eliptičkih krivulja u kriptografiji je učinkovitost u odnosu na druge vrste krivulja. Za razliku od drugih krivulja, poput kvadratnih krivulja, eliptičke krivulje zahtijevaju manje bitova za stvaranje sigurnih kriptografskih ključeva.

Brute force napadi pokušavaju probiti kriptografski ključ pokušajem svih mogućih kombinacija ključeva dok se ne pronađe ispravni ključ. Međutim, eliptičke krivulje koriste velike brojeve kako bi stvorile ključeve koji su preveliki za brute force napade. Što je ključ duži, to je veća vjerojatnost da će biti teže probiti ga brute force napadom.

Što se tiče kvantnih računala, ona su u stanju dešifrirati neke vrste kriptografskih ključeva, posebno one koje se temelje na faktorizaciji velikih brojeva. Međutim, kvantna računala se još uvijek razvijaju i nisu dovoljno napredna da bi dešifrirala kriptografske ključeve koji se temelje na eliptičkim krivuljama.

# 5. Zadaci

## Zadatak 1.

Koristeći bilo koji programski jezik na raspologanju kao i materijale s Interneta, napravite program koji kodira tekst koristeći Base58 te ga ponovo vratite u original tekst. Testni podatak:

| Tekst | Base58 |
|---|---|
| FERIT | 8vpytW7 |

## Zadatak 2.

Koristeći bilo koji programski jezik na raspolganju kao i materijale s Interneta, napravite program koji će prikazati hash vrijednost unešenog teksta koristeći SHA-256. Testni podatak:

| Tekst | SHA-256 |
|---|---|
| FERIT | 0ec299f96b70e36bc823d0a546b725fccc97c00e00dba67f7322abc9ab6ce0eb |

## Zadatak 3.

Koristeći bilo koji programski jezik na raspolganju kao i materijale s Interneta, napravite program koji će generirati par ključeva. Javnim ključem kodirati poruku a privatnim ključem ju dekodirati. Prilikom pokazivanja zadatka, potrebno je objasniti logiku i algoritam.