Univerzitet u Beogradu

Fakultet organizacionih Nauka

Seminarski rad

Tema: Protokol razmene neuronskih ključeva

Mentor: Student:

Marija Bogićević Sretenović Stefan Petković 276/18

Beograd, 2022.

Sadržaj

[1. Uvod 3](#_Toc108525637)

[2. Mašina za uparivanje stabala (Tree parity machine) 4](#_Toc108525638)

[3. Protokol uparivanja 5](#_Toc108525639)

[4. Implementacija 6](#_Toc108525640)

[5. Sigurnost i napadi 9](#_Toc108525641)

[5.1. *Brute force* napad 9](#_Toc108525642)

[5.2. Učenje sopstvene mreže 9](#_Toc108525643)

[5.3. Ostali napadi 9](#_Toc108525644)

[6. Zaključak 10](#_Toc108525645)

[7. Literatura 11](#_Toc108525646)

# Uvod

Tradicionalni protokoli razmene ključeva zasnivaju se na teoriji brojeva, najkorišćeniji među njima je Diffie-Hellman protokol. Sa druge strane, pojavljuje se novi metod razmene ključeva koji se zasniva na veštačkim neuronskim mrežama. Ovaj metod koristi osobinu sinhronizacije dve mreže, kako bi kreirao deljenu tajnu.

U protokolu razmene neuronskih ključeva, svaka strana kreira neuronsku mrežu sa jednim skrivenim slojem, koja se naziva *Tree parity machine.* One uče na osnovu međusobnih izlaza sve do potpune sinhronizacije, nakon čega se njihove matrice težine mogu koristiti kao ključevi za enkripciju.

Najbitnija stvar kod ovog protokola je to što on osigurava da se ključ ne može zaključiti, iako napadač ima sve parametre neuronske mreže i prisluškuje njihovu komunikaciju.

U ovom radu ćemo pogledati kako je matematički definisana mašina za uparivanje stabla. Nakon toga ćemo proći kroz protokol uparivanja, implementirati i testirati mašinu u programskom jeziku Python i na kraju proći kroz sigurnost i moguće napade na ovaj metod razmene ključeva.

# Mašina za uparivanje stabala (Tree parity machine)

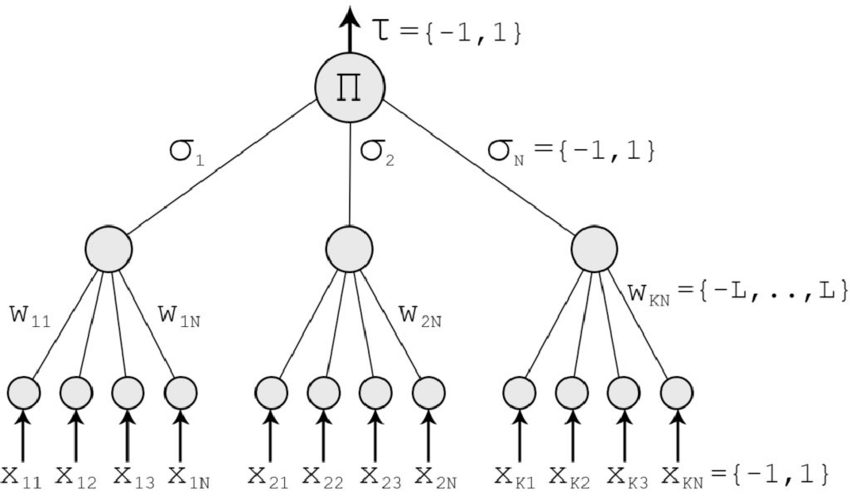
Mašina za uparivanje stabala je specijalan tip višeslojne neuronske mreže. Sastoji se od **jednog izlaznog neurona**, **K skrivenih neurona** i **K \* N ulaznih neurona**. Ulazni neuroni uzimaju tri vrednosti:

Težinski koeficijenti između ulaznih neurona i skrivenih neurona uzimaju sledeće vrednosti:

Izlazna vrednost svakog skrivenog neurona se računa kao zbir svih proizvoda ulaznih neurona i težinskih koeficijenata:

Signum je jednostavna funkcija koja vraća -1, 0 ili 1:

Ako je skalarni proizvod 0, izlaz skrivenog neurona postaje -1, kako bi izlaz iz mreže bio binaran. Izlaz mreže se računa kao proizvod svih vrednosti skrivenih neurona i uzima vrednost 1 ili -1:



Slika 1 Mašina za uparivanje stabala

# Protokol uparivanja

Svaka strana (A i B) kreira mašinu za uparivanje stabala sa istim brojem skrivenih neurona K, ulaznih neurona N (po skrivenom neuronu) i istim rasponom vrednosti težinskih koeficijenata L. Nakon toga počinje proces sinhronizacije koji se izvršava sledećim koracima:

1. Inicijalizuj težinske koeficijente nasumičnim vrednostima
2. Izvršavaj sledeće korake do potpune sinhronizacije:
   1. Generiši nasumični ulazni vektor X
   2. Izračunaj vrednosti skrivenih neurona
   3. Izračunaj vrednost izlaznog neurona
   4. Uporedi izlaze obe mašine za uparivanje stabala
      1. Ako su izlazi jednaki: upotrebi odgovarajuće pravilo učenja nad težinama
      2. Ako su izlazi različiti: vrati se na korak 2.1

Potpuna sinhronizacija se dešava kada su svi težinski koeficijenti identični kod obe strane (A i B).

Jedno od sledeća tri pravila učenja se može koristiti u procesu sinhronizacije:

* *Hebbian learning rule:*
* *Anti-Hebbian learning rule:*
* *Random walk:*

Gde je:

Sinhronizacija neuronskih mreža može odmah biti prevedena u generaciju ključeva u kriptografiji. Identične matrice težine nalaze se kod obe strane A i B i one se mogu iskoristiti kao ključ za enkripciju; bilo odmah kao *one-time pad*, kao ulaz za generator nasumičnih bitova ili kao ključ za drugi algoritam za enkripciju (AES, DES).

# Implementacija

Mašinu za uparivanje stabala implementiraćemo u pajtonu. Potrebno je definisati pravila učenja, kreirati klasu mašine za uparivanje stabala sa pomoćnim metodama i onda pokrenuti test u kome ćemo simulirati komunikaciju između Alice i Boba, dok će Eve tu komunikaciju prisluškivati.

Definišemo theta funkciju i *hebbian* pravilo učenja u fajlu learning\_rules.py:

def theta(t1, t2):

    return int(t1 == t2)

def hebbian(W, X, sigma, tau1, tau2, l):

    k, n = W.shape

    for (i, j), \_ in np.ndenumerate(W):

        W[i, j] += X[i, j] \* tau1 \* theta(sigma[i], tau1) \* theta(tau1, tau2)

        W[i, j] = np.clip(W[i, j] , -l, l)

Nakon toga, kreiramo klasu **TreeParityMachine** koja predstavlja mašinu za uparivanje stabala. Ona će u svom konstruktoru primiti hiperparametre K, N i L i na osnovu njih će definisati matricu težina dimenzija K x N.

class TreeParityMachine:

    def \_\_init\_\_(self, k=3, n=4, l=5):

        self.k = k

        self.n = n

        self.l = l

        self.W = np.random.randint(-l, l+1, [k, n])

        self.count = 0

Potrebno je da kreiramo metode za računanje izlaza i za ažuriranje težina. To ćemo uraditi na sledeći način:

    def get\_output(self, X):

        k = self.k

        n = self.n

        W = self.W

        X = X.reshape([k, n])

        sigma = np.sign(np.sum(X \* W, axis=1))

        tau = np.prod(sigma)

        self.X = X

        self.sigma = sigma

        self.tau = tau

        return tau

def update(self, tau2, learning\_rule='hebbian'):

        X = self.X

        tau1 = self.tau

        sigma = self.sigma

        W = self.W

        l = self.l

        if(tau1 == tau2):

            self.update\_count += 1

            if learning\_rule == 'hebbian':

                hebbian(W, X, sigma, tau1, tau2, l)

            elif learning\_rule == 'anti\_hebbian':

                anti\_hebbian(W, X, sigma, tau1, tau2, l)

            elif learning\_rule == 'random\_walk':

                random\_walk(W, X, sigma, tau1, tau2, l)

            else:

                raise Exception("Pravila učenja mogu imati sledeće vrednosti: ['hebbian', 'anti\_hebbian', 'random\_walk']")

U metodi **get\_output** računamo izlaz mašine na osnovu prosleđene matrice X, a u metodi **update** ažuriramo vrednosti težinskih koeficijenata na osnovu izlaza druge mašine i prosleđenog pravila. Ako pravilo nije prosleđeno koristimo pravilo učenja hebbian.

Kada smo definisali strukturu mašine, kreiraćemo demonstraciju rada u fajlu main.py. Prvo ćemo definisati hiperparametre mašine, izabrati pravilo učenja i kreirati funkciju za generisanje matrice nasumičnih brojeva dimenzije K x N i funkciju za računanje nivoa sinhronizacije koja računa srednju apsolutnu grešku matrica težina dve mašine.

# Hiperparametri mreže

k = 100

n = 10

l = 10

# Pravila učenja

learning\_rules = ['hebbian', 'anti\_hebbian', 'random\_walk']

learning\_rule = learning\_rules[0]

# Generator matrice nasumičnih brojeva

def random\_matrix():

    return np.random.randint(-l, l+1, [k, n])

# Funkcija za evaluaciju sinhronizacije dve mašine

def sync\_score(m1, m2):

    return 1.0 - np.average(1.0 \* np.abs(m1.W - m2.W)/(2 \* l))

Kreiramo 3 mašine: Alice, Bob i Eve i inicijalizujemo pomoćne promenljive i tajmer.

# Kreiranje 3 mašine: Alice, Bob, Eve. Eve će pokušati da presretne komunikaciju između Alice i Boba

print(f'Kreiranje mašina sa parametrima: k={k}, n={n}, l={l}')

print(f'Korišćenjem {learning\_rule} pravila učenja:')

Alice = TreeParityMachine(k, n, l)

Bob = TreeParityMachine(k, n, l)

Eve = TreeParityMachine(k, n, l)

# Sinhronizacija težina

sync = False

updates\_cnt = 0

eve\_updates\_cnt = 0

sync\_history = []

eve\_history = []

start\_time =  time.time()

Sada možemo pokrenuti proces sinhronizacije. U njemu se, kao što je navedeno u protokolu uparivanja, generiše matrica nasumičnih brojeva, mašine preračunavaju izlaze i ažuriraju se. Ciklus se ponavlja sve dok dve mašine nisu u potpunosti sinhronizovane.

while(not sync):

    X = random\_matrix() # Kreiranje matrice dimenzija [k, n] sa nasumičnim vrednostima.

    tauA = Alice.get\_output(X)

    tauB = Bob.get\_output(X)

    tauE = Eve.get\_output(X)

    Alice.update(tauB, learning\_rule) # Ažuriranje Alice-ine mašine Bobovim izlazom

    Bob.update(tauA, learning\_rule) # Ažuriranje Bobove mašine Alice-inim izlazom

    # Eve ažurira vrednost samo ako važi uslov: tauA == tauB == tauE

    if tauA == tauB == tauE:

        Eve.update(tauA, learning\_rule)

        eve\_updates\_cnt += 1

    updates\_cnt += 1

    score = 100 \* sync\_score(Alice, Bob) # Računanje nivoa sinhronizacije Alice-ine i Bobove mašine

    eve\_score = 100 \* sync\_score(Alice, Eve) # Računanje nivoa sinhronizacije Alice-ine i Eve-ine mašine

    sync\_history.append(score) # Čuvanje rezultata za kasnije plotovanje

    eve\_history.append(eve\_score)

    sys.stdout.write('\r' + "Sinhronizacija = " + str(int(score)) + "% / Broj iteracija = " + str(updates\_cnt) + " / Broj ažuriranja Eve = " + str(eve\_updates\_cnt) + " / Broj ažuriranja pravila učenja :" +str(Alice.update\_count))

    # Ako je rezultat 100, mašine su sinhronizovane

    if score == 100:

        sync = True

Nakon sinhronizacije zaustavljamo tajmer i računamo proteklo vreme. Ispisaćemo rezultate sinhronizacije, to jest potrebno vreme, broj iteracija i broj ažuriranja težina. Proveravamo da li je Eve uspela da sinhronizuje svoju mašinu prisluškujući komunikaciju između Alice i Boba i crtamo grafik u kome plava linija predstavlja nivo sinhronizacije Alice-ine mašine i Bobove mašine u vremenu, a žuta nivo sinhronizacije Eve-ine mašine sa ostale dve.

end\_time = time.time()

time\_taken = end\_time - start\_time

# Ispisivanje rezultata

print('\nMašine su se uspešno sinhronizovale.')

print(f'Proces je trajao {time\_taken} sekundi.')

print(f'Broj iteracija: {updates\_cnt}.')

print(f'Procenat ažuriranja mašina u odnosu na broj iteracija: {float(Alice.update\_count / updates\_cnt)\*100:.2f}%')

# Provera rezultata Eve

if eve\_score >= 100:

    print('Eve je uspela da sinhronizuje svoju mašinu sa Alice i Bobom.')

else:

    print(f'Eve je uspela da sinhronizuje svoju mašinu {str(int(eve\_score))}% i uradila je {eve\_updates\_cnt} ažuriranja.')

# Crtanje grafika sinhronizacije

plt.plot(sync\_history)

plt.plot(eve\_history)

plt.title("Grafik toka sinhronizacije mašina")

plt.xlabel("Broj iteracija")

plt.ylabel("Nivo sinhronizacije")

plt.show()

Pokretanjem programa sa parametrima K = 100, N = 10, L = 10 i pravilom učenja hebbian dobijamo sledeći izlaz:

Sinhronizacija = 100% / Broj iteracija = 918 / Broj ažuriranja Eve = 170 / Broj ažuriranja pravila učenja :518

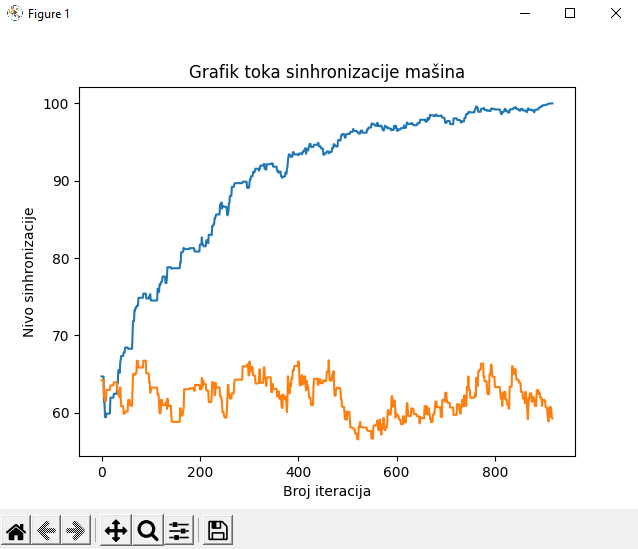
Mašine su se uspešno sinhronizovale.

Proces je trajao 33.19 sekundi.

Broj iteracija: 918.

Procenat ažuriranja mašina u odnosu na broj iteracija: 56.43%

Eve je uspela da sinhronizuje svoju mašinu 59% i uradila je 170 ažuriranja.



Slika 2 Grafik toka sinhronizacije mašina

# Sigurnost i napadi

Siguran protokol razmene ključeva mora imati sledeću osobinu: bilo koji napadač koji zna sve detalje protokola i sve informacije razmenjene između strana A i B ne sme imati dovoljno računarske snage da izračuna tajni ključ. U sledećim primerima napada, pretpostavljamo da napadač E poseduje sve informacije protokola i prisluškuje komunikaciju dve strane A i B.

## *Brute force* napad

U *brute force* napadu, napadač mora da testira sve moguće vrednosti ključa (sve vrednosti težinskih koeficijenata ). Sa K skrivenih neurona, K\*N ulaznih neurona i graničnom vrednošću težina L, dobijamo (2L + 1)KN mogućnosti. Na primer, sa konfiguracijom K = 3, L = 3 i N = 100 broj mogućnosti iznosi 3\*10253 dokazujući sigurnost protiv ovakvog napada.

## Učenje sopstvene mreže

Jedna od osnovnih ideja napada je učenje sopstvene mašine za uparivanje stabala na osnovu izlaza strana A i B. Napadač pokušava da sinhronizuje svoju mrežu sa mrežama koje prisluškuje i tako dođe do tajnog ključa. U svakom koraku postoje tri moguće situacije:

1. Izlaz(A) ≠ Izlaz(B): Ni jedna strana ne ažurira težine.
2. Izlaz(A) = Izlaz(B) = Izlaz(E): Sve strane ažuriraju težine.
3. Izlaz(A) = Izlaz(B) ≠ Izlaz(E): Strane A i B ažuriraju svoje težine, ali napadač E ne može ažurirati svoje. Zbog ove situacije njegovo učenje je sporije i samim tim ne može da stigne da se sinhronizuje sa stranama A i B.

## Ostali napadi

Postoji još broj drugih napada kao što su geometrijski napad koji koristi osobinu lokalnog polja, genetski napad koji se oslanja na mutaciju početne mašine za uparivanje stabala koristeći evolutivni algoritam predviđa skrivene neurone, kao i većinski napad koji koristi ansambl TPMa.

Većina ovih napada može se izbeći povećanjem sinaptičke dubine L i povećanjem broja skrivenih neurona K i ulaznih neurona N. Takođe je utvrđeno da je pravilo učenja  *Random walk* najsigurnije.

Pored svega, još uvek nije dokazana potpuna sigurnost ove metode i zbog toga nastaju i nove varijacije modela TPMa, kao što je *Vector-valued tree parity machine* čije su unutrašnje vrednosti vektori.

# Zaključak

Neuronske mreže nalaze sve veću primenu u svim domenima informacionih tehnologija. U domenu zaštite računarskih sistema, neuronske mreže se pojavljuju u vidu mašine za uparivanje stabala (*Tree parity machine)* koja predstavlja prvi algoritam za razmenu ključeva preko javnog kanala koji nije baziran na teoriji brojeva. Jedinstvena osobina sinhronizacije ove mreže dozvoljava joj da dve strane generišu tajni ključ preko nesigurnog medijuma, bez prethodne komunikacije.

Kao što smo videli, implementacija ovog algoritma je jako jednostavna jer je sam algoritam lako razumljiv. Model sadrži samo 3 hiper parametra: Broj skrivenih neurona K, broj ulaznih neurona N i sinaptičku dubinu L. Samim tim sve što je potrebno za pokretanje modela je postavljanje odgovarajućih parametara i pravila učenja i mreže su spremne za sinhronizaciju.

Algoritam je sa odgovarajućim parametrima siguran od *brute force* napada, napada prisluškivanja u kojima treće lice uparuje svoju mrežu sa mrežama dve strane koje komuniciraju, kao i od nekih kompleksnijih napada kao što su geometrijski napad, genetski napad i većinski napad. Pored osnovnih algoritma postoji i veći broj varijacija originalnog modela koji unapređuju njegovu sigurnost. Budućnost će pokazati da li je ovaj vid razmene ključeva dovoljno siguran za realnu upotrebu.

# Literatura

1. Kinzel, W., & Kanter, I. (2002, November). Neural cryptography. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.* (Vol. 3, pp. 1351-1354). IEEE.
2. Jeong, S., Park, C., Hong, D., Seo, C., & Jho, N. (2021). Neural cryptography based on generalized tree parity machine for real-life systems. *Security and Communication Networks*, *2021*.
3. *Neural cryptography* (2022, March 26). Preuzeto 11. jula 2022, sa <https://en.wikipedia.org/wiki/Neural_cryptography>
4. *Diffie–Hellman Key Exchange* (2022, July 9). Preuzeto 10. jula*,* 2022, sa[https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman\_key\_exchange](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange%20)
5. Rahman, F. (2015). *Neuralkey.* Preuzeto 11. jula 2022, sa <https://github.com/farizrahman4u/neuralkey>