SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

**Optimizacija PUF-ova**

Marina Krček

Voditelj: prof. dr. sc. Domagoj Jakobović

Zagreb, svibanj, 2016.

**Sadržaj**

[1. Uvod 1](#_Toc452392482)

[2. Physical unclonable functions 2](#_Toc452392483)

[2.1 Tipovi PUF-ova 3](#_Toc452392484)

[2.2 Arbiter PUF 4](#_Toc452392485)

[3. Evolucijsko računanje 6](#_Toc452392486)

[4. Programsko rješenje 8](#_Toc452392487)

[4.1 Programski model Arbiter PUF-a 8](#_Toc452392488)

[4.2 Korištene tehnologije 9](#_Toc452392489)

[5. Rezultati 12](#_Toc452392490)

[6. Zaključak 15](#_Toc452392491)

[7. Literatura 16](#_Toc452392492)

[8. Sažetak 17](#_Toc452392493)

# Uvod

U ovom radu obrađuju su fizički ostvarene funkcije čija je glavna karakteristika nemogućnost stvaranja dvije identične funkcije. Engleski naziv za takve funkcije je physical unclonable functions od kojeg dolazi i kratica PUF. Njihova karakteristika vrlo je korisna u kriptografiji, za primjerice generiranje ključeva i njihovu pohranu. Rad opisuje model PUF-a korišten za simulaciju i dobivanje potrebnih podataka za optimizaciju. Za optimiziranje PUF-ova koristi se okruženje ECF – Evolutionary Computation Framework u kojem postoje razni optimizacijski algoritmi. Objašnjeno je ostvareno programsko rješenje te način na koji su se upotrijebili algoritmi implementirani u ECF-u. Provedena su ispitivanja čiji su rezultati prikazani u petom poglavlju.

# Physical unclonable functions

Physical unclonable functions (PUF, physically unclonable function) prvi put su predstavljene 2001. godine, a od 2010. su se počele proučavati kako bi se mogle koristiti za izradu jedinstvenih kriptografskih ključeva. PUF su funkcije ostvarene fizičkim komponentama i vrlo je lako izračunati njihov izlaz (evaluirati ih), ali je teško predvidjeti taj izlaz. PUF treba biti jednostavan za izradu, a gotovo nemoguće bi trebalo biti napraviti isti takav (duplikat, klonirati ga) čak i ako se zna proces izrade i struktura.

Izrada PUF-ova zapravo se oslanja na karakteriste fizičkih komonenti od kojih se gradi i „greškama“ u proizvodnji. Točnije, svaka fizička komponenta ima svoje karakteristike koje u proizvodnji mogu varirati, kao što su na primjer kašnjenja. Takve karakteristike nisu kontrolirane te su zato nepredvidljive što daje PUF-ovima svojstvo nemogućnosti „kloniranja“, izrade identičnog PUF-a.

PUF-ovi implementiraju challenge-response ovjeru autentičnosti kako bi evaluirali mikrostrukturu. Challenge-response ovjera je skup protokola u kojima jedna strana postavlja pitanje (challenge), a druga strana treba dati točan odgovor (response) na to pitanje. Na strukturu se pošalje fizički stimulans (ulaz) na koji se dobije reakcija (izlaz) uređaja – nepredvidljiva, ali ponavljajuća (za isti stimulans uvijek se dobije isti izlaz, ista reakcija). Taj stimulans je zapravo pitanje (challenge), a reakcija PUF-a, tj. izlaz je odgovor (response).

S obzirom da PUF-ovi svaki put izračunavaju izlaz (odgovor) na zadani ulaz ne zahtjevaju pohranjivanje tablica s odgovorima na sve moguće ulaze (pitanja), što postaje veliki problem kako broj bitova challenga raste. Hardverska izrada PUF-a proporcionalna je broju challenge i response bitova, što ga čini puno jednostavnijim.

Fizički unclonability znači da svaki PUF uređaj ima jedinstven i nepredvidljiv način računanja izlaza na osnovu ulaza, čak i kad je izrađen istim procesom kao slični uređaji. Matematički unclonability znači da bi trebalo biti jako teško predvidjeti nepoznati odgovor ako su poznati ostali parovi challenge-response ili neke slučajne karakteristike PUF-a.

S obzirom na sve navedene karakteristike PUF uređaja zaključuje se da se PUF-ovi mogu koristiti kao jedinstveni uređaji za identifikacije. Također, mogu se koristiti kao siguran generator ključeva i za pohranu ključeva. Zapravo se često koriste za pametne kartice (kartice od banaka, SIM kartice za mobitele, kartice za javni prijevoz i slično).

## Tipovi PUF-ova

Postoje različiti načini da dobivanje slučajnosti (nasumičnosti) u PUF-ovima. Dijelimo PUF-ove na one u kojima se slučajnost dobiva eksplicitno i na one u kojima je slučajnost već unutar fizikalnog sustava.

Kod PUF-ova s eksplicitnom slučajnošću veća je otpornost na varijacije u okolini (temeratura i slično) te se uređaji više razlikuju i to zbog parametara koji se mogu direktno kontrolirati i optimizirati. Takvi su optički PUF-ovi (Optical PUF) i PUF-ovi s prevlakom (Coating PUF). Optički koriste raspršivače svjetlosti, a Coating PUF ima zaštitnu prevlaku preko čipa koja je popunjena dielektričnim česticama različitih veličina i na različitim lokacijama i gdje se relativna dielektrična konstatna razlikuje od dielektrične konstante prevlake.

Kod druge vrste PUF-ova nije potrebno mijenjati proces proizvodnje. Tu pripadaju PUF-ovi na osnovi kašnjenja (Delay PUF), SRAM PUF-ovi koji se oslanjanju na SRAM memoriju. Kod SRAM PUF-ova generira se jedinstveni „otisak prsta“ tako da se pri pokretanju (inicijalizaciji) ćelije SRAM-a pune nasumično jedinicama i nulama. Iz toga se onda može dobiti ključ. Bistable Ring PUF-ovi primaju ulaz od N bitova, i svaki bit odabire jedan od dva moguća izlaza na logičkom sklopu koji su osnovne komponente PUF-a. Tu pripadaju još i Magnetic PUF, Metal Based PUF i drugi.

Osnovni princip kod Delay PUF-ova je usporedba parova strukturno jednakih elemenata i mjerenje razlika u kašnjenjima koja nastaju zbog varijacija u proizvodnji, a ne u dizajnu. Postoje različite arhitekture Delay PUF-ova, kao primjerice Arbiter PUF, Ring Oscillator PUF i Butterfly PUF. U ovom seminaru koristimo Arbiter PUF te je on detaljnije objašnjen.

## Arbiter PUF

Arbiter PUF ima gornji i donji signal koji prolazi kroz k razina kašnjenja (delay stages). Svaka od tih razina sastoji se od dva 2-bitna multipleksora (MUX-evi) koji su identični i oba primaju gornji i donji signal kao ulaz. Kako je raspored puta jednak, očekivali bismo da je i kašnjenje identično. No, u praksi svaki tranzistor u multipleksorima ima malo drugačija svojstva kašnjenja zbog varijacija u procesu proizvodnje, zbog toga je kašnjenje drugačije za gornji i donji signal. Također, te razlike u kašnjenjima su jedinstvene za svaki čip. Shematski prikaz Arbiter PUF-a može se vidjeti na slici 1.

Za ulaz (challenge bit) $c\_{i}=1$, gdje je i razina (stage) i, multipleksori zamijene gornji i donji signal, ako je $c\_{i}=0$, nema zamjene. Na ovaj način, signal (race signal) može ići različitim putevima – Arbiter PUF za k razina ima $2^{k}$ različitih puteva. Arbiter (sudac) na kraju svih delay razina odlučuje koji je signal bio brži, gornji ili donji. Ako je brži bio gornji izlaz je 1, ako je donji brži, onda je izlaz 0.


Slika 1. Arbiter PUF [7]

Problem s Arbiter PUF-om je taj da se može simulirati programski na računalu i parametri za takav programski model mogu se aproksimirati uporabom tehnika strojnog učenja. Kako bi se povećala otpornost Arbiter PUF-a od napada strojnim učenjem rješenje se vidjelo u dodavanju nelinearnih elemenata u dizajn.

Uobičajeno je dodavanje XOR elemenata, pa tako imamo XOR PUF. U n-XOR PUF-u, n Arbiter PUF-ova postavljeno je na čip. Svaki Arbiter PUF prima iste ulazne bitove (challenge), a XOR izračunamo na izlazima (responses) od n PUF-ova te tako dobijemo završni izlaz. Povećavanjem broja PUF-ova koji ulaze u XOR povećava se otpornost od napada, no, time opada pouzadnost XOR PUF-a. Iz navedenih razloga u praksi je broj PUF-ova koji ulaze u XOR ograničen. Izlaz k-razinskog Arbiter PUF-a je određen razlikom kašnjenja između gornjeg i donjeg signala. Razlika kašnjenja je suma razlika kašnjenja u svim razinama (k). Razlika kašnjenja pojedine razine ovisi o odgovarajućem challenge bitu. Dakle, postoje dvije razlike kašnjenja za svaku razinu i, $δ\_{1, i}$ razlika je kašnjenja za challenge bit 1, a $δ\_{0, i}$ za challenge bit 0. Kako je Arbiter PUF realiziran kao programski model objašnjeno je u poglavlju 4.

# Evolucijsko računanje

Umjetna inteligencija se dijeli na nekoliko područja, a jedno od njih je evolucijsko računanje. Evolucijsko računanje temelji svoj rad na populaciji. Nad populacijom se izvršavaju određene operacije koje vode cjelokupnu populaciju određenom cilju. Spomenute operacije najčešće su inspirirane evolucijskim procesima iz prirode, primjerice križanja i mutacije. No, javljaju se i algoritmi inspirirani drugačijim prirodnim pojavama kao što je inteligencija rojeva. Evolucijsko računanje pripada heuristikama i metahuristikama, što znači da ove tehnike ne garantiraju pronalazak pravog globalnog optimuma, već dovoljno dobrog. Takve se metode koriste kada je optimalno rješenje nemoguće pronaći, pa se ovako pronalazi zadovoljavajuće rješenje u konačnom vremenu. Također, to su iterativne metode, gdje se kroz svaku iteraciju pokušava približiti globalnom optimumu.

Evolucijsko računanje (EC - evolutionary computation) dijeli se na tri područja:

* evolucijski algoritmi (EA - evolutionary algorithms)
* algoritmi temeljeni na inteligenciji rojeva (SI - swarm intelligence)
* ostali algoritmi

Evolucijski algoritmi dijele se na:

* genetske algoritme (GA - genetic algorithm)
* evolucijske strategije (ES - evolution strategy)
* evolucijsko programiranje (EP - evolutionary programming)
* genetsko programiranje (GP - genetic programming)
* učenje sustava klasifikatora (LCS – learning classifier systems)

Algoritmi temeljeni na inteligenciji rojeva dijele se na:

* optimizacija kolonijom mrava (ACO - ant colony optimization)
* optimizacija rojem čestica (PSO - particle swarm optimization)
* stohastička difuzijska pretraga (SDS - stochastic diffusion search)

Ostali algoritmi su:

* diferencijska evolucija (DE - differential evolution)
* kulturološki algoritmi (CA - cultural algorithms)
* algoritmi traženja sklada (HS - harmony search algorithm)
* umjetni imunološki sustavi (AIS - artificial immune systems)
* i drugi


Slika 2.: Podjela algoritama evolucijskog računanja [13]

U ovom radu od genetskih algoritama isprobat će se Steady State Tournament (SST) algoritam (k-turnirska selekcija) te Roulette Wheel (RW, proporcionalna selekcija). Isprobat će se i algoritmi temeljeni na inteligenciji rojeva – Particle Swarm Optimization (PSO) i Artificial Bee Colony (ABC). Također, koristit će se i spomenuti Differential Evolution (DE) te iz umjetnih imunoloških sustava - Clonal selection algorithm (ClonAlg) i Optimization Immune algorithm (OptIA).

# Programsko rješenje

U ovom poglavlju objašnjen je programski model Arbiter PUF-a [1], te korištene tehnologije.

## Programski model Arbiter PUF-a

Rečeno je da izlaz k-razinskog Arbiter PUF-a čini razlika između kašnjenja gornjeg i donjeg signala te da je to zapravo suma razlika kašnjenja u svim pojedinim razinama PUF-a. Također, za svaku razinu imamo dvije razlike kašnjenja, ovisno o ulaznom bitu.

k-razinski Arbiter PUF modeliran je izračunom vektora kašenjenja $\vec{ω}=\left(ω\_{1}, …, ω\_{k+1}\right)$, gdje

|  |  |
| --- | --- |
| $$ω\_{1}= δ\_{0,1 }- δ\_{1, 1}$$ | (1) |
| $$ω\_{i}= δ\_{0,i-1 }+ δ\_{1, i-1}+δ\_{0,i }-δ\_{1,i } za 2\leq i\leq k$$ | (2) |
| $$ω\_{k+1}= δ\_{0, k }+ δ\_{1, k}$$ | (3) |

Razlika kašnjenja $ΔD$ na kraju rezultat je skalarnog produkta transponiranog vektora kašnjenja $\vec{ω}$ i svojstvenog vektora $\vec{ϕ}$:

|  |  |
| --- | --- |
| $$ΔD= \vec{ω}^{T}\vec{ϕ}$$ | (4) |
| $$r= \left\{\begin{array}{c}1, ΔD <0\\0, ΔD>0\end{array}\right.$$ | (5) |

Svojstveni vektor $\vec{ϕ}$ dobije se iz ulaznog vektora $\vec{c}$ (challenge vector):

|  |  |
| --- | --- |
| $$ϕ\_{i}= \prod\_{l=i}^{k}(-1)^{c\_{l}} za 1 \leq i \leq k$$ | (6) |
| $$ϕ\_{k+1}=1$$ | (7) |

## Korištene tehnologije

Koristi se već implementirani Arbiter PUF u Matlab skripti koji simulira Arbiter PUF te u datoteke „response.txt“ i „feature.txt“ zapisuje dobiveni izlaz, odnosno svojstveni vektor $\vec{ϕ}$. Navedene datoteke dalje koristi program za evaluaciju napisan u C++ programskom jeziku unutar Visual Studia koji povezuje Evolutionary Computation Framework (ECF) i problem optimizacije PUF-a.

Korišteni program za evaluaciju služi da pomoću algoritama implementiranih u ECF-u optimiziramo vektor kašnjenja $\vec{ω}=\left(ω\_{1}, …, ω\_{k+1}\right)$, za k-razinski Arbiter PUF. Pseudokod za evaluaciju napisan je u nastavku:

error = 0

za svaki response bit

 delay = skalarni\_produkt(feature\_vektor, jedinka)

 ako delay < 0

 r = 1

 inače

 r = -1

 ako r != trenutni\_response\_bit

 error++

Razlika između objašnjenog programskog modela Arbiter PUF-a i implementacije istog je ta da se umjesto binarnih vrijednosti 1 i 0, koriste vrijednosti 1 i -1, kao što se vidi u psedokodu.

Za strukturu jedinke koja predstavlja vektor kašnjenja $\vec{ω}$ koristit će se tri genotipa, jedan od njih je FloatingPoint koji postoji u ECF-u. Kako bi se uspješno koristio potrebno je odrediti gornju i donju granicu (interval) dopuštenih vrijednosti koje vektor može pohraniti prilikom inicijalizacije. Također, potrebno je odrediti i veličinu vektora, što će u našem slučaju biti jednako broju razina uvećanih za jedan, dakle u k-razinskom Arbiter PUF-u, to će biti k+1. Uvećava se za 1, zbog formule ([3](#Formula3)).

<Genotype>

<FloatingPoint>

 <Entry key="lbound">-1</Entry>

 <Entry key="ubound">1</Entry>

 <Entry key="dimension">65</Entry>

 </FloatingPoint>

</Genotype>

Sljedeći genotip je Binary, s istim parametrima kao i FloatingPoint, uz jedan dodatni – precision koji određuje koliko decimalnih mjesta se uzima u obzir pri pretvorbi iz realne vrijednosti u niz bitova.

<Genotype>

<Binary>

<Entry key="lbound">-127</Entry>

<Entry key="ubound">128</Entry>

<Entry key="precision">0</Entry>

<Entry key="dimension">65</Entry>

</Binary>

</Genotype>

Postoji i IntGenotype genotip koji ima jednake parametre, s time da se veličina, tj. broj varijabli zapisuje pod parametrom size umjesto dimension.

<Genotype>

 <IntGenotype>

 <Entry key="lbound">0</Entry>

 <Entry key="ubound">15</Entry>

 <Entry key="size">8</Entry>

 </IntGenotype>

 </Genotype>

Također, potrebno je odabrati i algoritam iz ECF-a kojim se PUF želi optimirati. Za odabrani algoritam potrebno je odrediti i njegove parametre ukoliko postoje.

<Algorithm>

 <SteadyStateTournament>

 <Entry key="tsize">3</Entry>

 </SteadyStateTournament>

 </Algorithm>

Algoritam i genotip odabiru se zapisom u parametarsku datoteku, najčešće nazvanu „parameters.txt“ u prikazanom obliku.

Za grafički prikaz rezultata iz provedenih ispitivanja koristi se online alat – BoxPlotR [14].

# Rezultati

Matlab simulatorom k-razinskog Arbiter PUF-a generirani su challenge i response bitovi, te zapisani u datoteke, za različite vrijednosti k – 64, 256 te 512. Broj response bitova postavljen je na 50 za sve navedene vrijednosti k.

Dobiveni podaci koristili su se za optimizaciju algoritmima Steady State Tournament (SST) te Roulette Wheel (RW). Kroz parametarsku, tj. konfiguracijsku datoteku zadaje se veličina populacije, te razni uvjeti zaustavljanja. U ovim ispitivanjima veličina populacije postavljena je na 50, a kao uvjeti zaustavljanja maksimalan broj evaluacija postavljen je na 5000000, generacija na 100000, broj generacija bez poboljšanja 50000, te pošto minimiziramo broj krivo dobivenih response bitova, dakle minimalna vrijednost je 0, postavljena je 0 kao vrijednost dobrote (fitness) čijim se postizanjem program zaustavlja. Radi statističkih podataka svaka se konfiguracija izvršila 10 puta.

<Registry>

 <Entry key="population.size">50</Entry>

 <Entry key="term.eval">5000000</Entry>

 <Entry key="term.stagnation">50000</Entry>

 <Entry key="term.maxgen">100000</Entry>

 <Entry key="term.fitnessval">0</Entry>

 <Entry key="batch.repeats">10</Entry>

 <Entry key="batch.statsfile">stats.txt</Entry>

 </Registry>

Za FloatingPoint genotip zadan je interval [-1, 1], te veličina vektora na k+1, gdje je k broj razina Arbiter PUF-a. S obzirom da su sva pokretanja u svim slučajevima došla do minimalne vrijednosti 0, slika 3 prikazuje koliko je evaluacija bilo potrebno da bi se došlo do optimuma. Broju evaluacija proporcionalno je i vrijeme te broj generacija.


Slika 3.: Broj evaluacija potrebnih za postizanje optimuma

Kako se vidi sa slike, za RW algoritam s većim brojem razina raste i broj evaluacija, tj. potrebno je više vremena kako bi se dosegao optimum. Za SST algoritam najbrže se došlo do optimuma kada je broj razina bio 256. Usporedbom SST i RW algoritama vidimo da je RW većinom bio brži, tj. s manje evaluacija je došao do minimalne pogreške.

Genotip Binary iskorišten je s intervalom [-127, 128], preciznošću 0 i veličinom vektora k+1, gdje je k broj razina Arbiter PUF-a. Za 64-razinski Arbiter PUF i 50 response bitova na slici 4 može se usporediti brzina algoritama SST i RW s Binary i FloatingPoint genotipima. Ovdje je također dana usporedba brzina jer su oba algoritma došla do optimalnog rezultata – 0.


Slika 4.: Usporedba Binary i FloatingPoint genotipa

Sa slike 4 može se uočiti da je Binary genotip u usporedbi s FloatingPoint genotipom puno sporiji, tj. sporije konvergira s algoritmima SST i RW, s time da je puno veća razlika kod SST algoritma nego kod RW. Također, RW je i s Binary genotipom brži od SST algoritma.

Isproban je i IntGenotype genotip s algoritmom RW program se zaustavio zbog dosezanja maksimalnog broja generacija bez poboljšanja (50000) na dobroti od 11 krivih response bitova.

# Zaključak

Physical unclonable functions (PUF-ovi) korisne su u kriptografiji jer im je izrada brza i jednostavna. Također, činjenica da nije potrebna memorija za pohranu svih mogućih ulaza i odgovarajućih izlaza PUF-a čini ih vrlo zanimljivima jer to omogućuje lakšu primjenu te povećanjem broja ulaznih bitova ne javljaju se problemi s memorijom. PUF-ovi svaki puta računaju svoj izlaz te je taj izračun brz, što bi s upotrebom memorije i velikim brojem ulaznih bitova bilo puno sporije. Korištenje „grešaka“ u proizvodnji, tj. varijacija kao izvor jedinstvenosti ovih fizičkih funkcija zanimljiv je pristup koji se pokazao dosta dobar jer se PUF-ovi koriste u svakodnevnom životu upotrebom pametnih kartica. S obzirom da su to fizički ostvarene funkcije, problem nastaje zbog utjecaja okoline – povišene temperature, napon napajanja i slično. Postoje mehanizmi ispravljanja grešaka zbog takvih slučajeva jer izlaz PUF-ova mora biti stabilan i ponavljajuć kako bi se npr. generirao svaki puta isti kriptografski ključ.

Kako je već napomenuto korišteni Arbiter PUF zbog toga što se može programski simulirati otvoren je za napade tehnikama strojnog učenja. Otpornost se povećava dodavanjem nelinearnih elemenata, u ovom slučaju – XOR. Smatra se da su PUF-ovi uz dovoljan broj nelinearnih elemenata sigurni od napada, no neki su članci pokazali da se strojnim učenjem i dalje može napasti Arbiter PUF.

Stoga, u ovom seminaru pokušalo se optimizirati vektor kašnjenja za Arbiter PUF korištenjem evolucijskog računanja. Algoritmima SST i RW uspješno se minimizirala pogreška „pogodaka“ response bitova s FloatingPoint i Binary genotipom. IntGenotype pokazao se lošijim jer se zaustavljao zbog dosezanja maksimalnog broja generacija bez poboljšanja i maksimalnog broja generacija. Između FloatingPoint i Binary genotipa možemo zaključiti da je FloatingPoint genotip brže konvergirao, a od navedena dva algoritma, RW je u manje evaluacija dosegnuo optimum s oba genotipa.

# Literatura

1. G. T. Becker, The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs
2. U. Rührmair, C. Hilgers, S. Urban, A. Weiershäuser, E. Dinter, B. Forster, C. Jirauschek, Optical PUFs Reloaded
3. P. Tuyls, Geert-Jan Schrijen, Boris Škorić, J. van Geloven, N. Verhaegh, R. Wolters, Read-Proof Hardware from Protective Coatings
4. S. Morozov, A. Maiti, P. Schaumont, A Comparative Analysis of Delay Based PUF Implementations on FPGA
5. NXP Semiconductors N.V., Protecting next-generation Smart Card ICs with SRAM-based PUFs, 2013.
6. D. Yamamoto, M. Takenaka, K. Sakiyama, N.Torii, Security Evaluation of Bistable Ring PUFs on FPGAs using Differential and Linear Analysis, 2014.
7. M. Ayat, R. E. Atani, S. Mirzakuchaki, On Design of PUF-Based Random Number Generators, 2011.
8. Wikipedia, <https://en.wikipedia.org/wiki/Physical_unclonable_function>
9. Wikipedia,<https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication>
10. Wikipedia, <https://en.wikipedia.org/wiki/Evolutionary_computation>
11. Wikipedia, <https://en.wikipedia.org/wiki/Heuristic>
12. Algoritmi zasnovani na evolucijskom računanju, FER - PROJEKT 2008./2009., <http://www.zemris.fer.hr/~golub/ga/studenti/projekt2008/evo.html>
13. M. Golub, prezentacija Evolucijsko računarstvo u okviru predmeta Neizrazito, evolucijsko i neuro-računarstvo, 2014.15. i 2015./16.
14. Online alat, <http://boxplot.bio.ed.ac.uk/>

# Sažetak

Physical Unclonable Functions (PUF) su fizički ostvarene funkcije čije najbitnije svojstvo je unclonability, što znači da bi trebalo biti gotovo nemoguće ostvariti dvije fizičke instance PUF-a s jednakim svojstvima. PUF-ovi su zbog svojih svojstava korisni i upotrebljavaju se u kriptografiji za autentifikaciju te često za sigurno generiranje i pohranu ključeva. Evolucijsko računanje temelji svoj rad na populaciji jedinki gdje kroz iteracije određenim operacijama nad jedinkama pokušava doći do najboljeg rješenja. Spomenute operacije najčešće su inspirirane procesima iz prirode, a s obzirom da se radi o (meta)heurističkim metodama dobivena rješenja su zadovoljavajuća, ne nužno i optimalna. Za optimizaciju koriste se postojeći algoritmi u Evolutionary Computation Frameworku (ECF-u). Objašnjen je programski model PUF-a koji se koristi u seminaru te korišteni prikazi rješenja u optimizaciji. Prikazani su i objašnjeni dobiveni rezultati ispitivanja.

**Ključne riječi:** PUF, unclonability, kriptografija, optimizacija, evolucijsko računanje, ECF, model PUF-a