Machine learning and evolutionary computation in design and analysis of symmetric key cryptographic algorithms

Knežević, Karlo

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Karlo Knežević

MACHINE LEARNING AND EVOLUTIONARY COMPUTATION IN DESIGN AND ANALYSIS OF SYMMETRIC KEY CRYPTOGRAPHIC ALGORITHMS

DOCTORAL THESIS

Zagreb, 2023

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Karlo Knežević

MACHINE LEARNING AND EVOLUTIONARY COMPUTATION IN DESIGN AND ANALYSIS OF SYMMETRIC KEY CRYPTOGRAPHIC ALGORITHMS

DOCTORAL THESIS

Supervisors:
Professor Domagoj Jakobović, Ph.D.
Associate Professor Stjepan Picek, Ph.D.

Zagreb, 2023

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Karlo Knežević

STROJNO UČENJE I EVOLUCIJSKO RAČUNARSTVO U OBLIKOVANJU I ANALIZI KRIPTOGRAFSKIH ALGORITAMA SA SIMETRIČNIM KLJUČEM

DOKTORSKI RAD

Mentori:
Prof. dr. sc. Domagoj Jakobović
Izv. prof. dr. sc. Stjepan Picek

Zagreb, 2023.

Doctoral thesis has been made at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Electronics, Microelectronics, Computer and Intelligent Systems.

Mentors: Professor Domagoj Jakobović, Ph.D., Associate Professor Stjepan Picek, Ph.D.

Doctoral thesis has: 174 pages

Number of doctoral thesis: _____

About the Supervisors:

Domagoj Jakobović was born in Našice in 1973. He received B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1996, 2001, and 2005, respectively. He received PhD degree in December 2005. on the subject of generating scheduling heuristics with genetic programming.

Since April 1997, he has worked in the Department of electronics, microelectronics, computer, and intelligent systems at FER. In January 2018, he was promoted to Full Professor. He led six scientific projects and was included in several domestic and international projects. He was a principal investigator of two projects financed by the Croatian science foundation. He published more than 120 papers in journals and conference proceedings on the application of stochastic optimization and machine learning in scheduling and cryptography, as well as the development of parallel evolutionary algorithms.

Prof. Jakobović is a senior member of IEEE and ACM. He is a member of a journal editorial board, serves as a technical reviewer for various international journals and a program committee member for several conferences.

Stjepan Picek was born in Rijeka in 1984. Stjepan finished his Ph.D. in 2015 with a topic on cryptology and evolutionary computation techniques. Moreover, he has several years of experience working in industry and government.

He is an associate professor in the Digital Security Group at Radboud University, The Netherlands. His research interests are security/cryptography, machine learning, and evolutionary computation. Before the associate professor position, Stjepan was an assistant professor at TU Delft, The Netherlands, a postdoctoral researcher at MIT, USA, and KU Leuven, Belgium. Up to now, Stjepan has given more than 30 invited talks at conferences and summer schools and published more than 140 refereed papers.

Stjepan is a member of the organization committee for the International Summer School in Cryptography. He is a program committee member and reviewer for several conferences and journals and a senior IEEE member.

O mentorima:

Domagoj Jakobović rođen je u Našicama 1973. godine. Diplomirao je, magistrirao i doktorirao u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1996., 2001. odnosno 2005. godine.

Od travnja 1997. godine radi na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave FER-a. U siječnju 2018. godine izabran je u zvanje redovitog profesora. Vodio je šest znanstvenih projekata i bio suradnik na nekoliko domaćih i međunarodnih projekata. Bio je voditelj dva projekta koje financira Hrvatska zaklada za znanost. Objavio je više od 120 radova u časopisima i zbornicima konferencija u području primjene stohastičke optimizacije i strojnog učenja u problemima raspoređivanja i kriptografiji te razvoja paralelnih evolucijskih algoritama.

Prof. Jakobović član je stručnih udruga IEEE i ACM. Član je uredničkog odbora znanstvenog časopisa te sudjeluje kao recenzent u dvadesetak inozemnih časopisa.

Stjepan Picek rođen je u Rijeci 1984. godine. Doktorirao je 2015. s temom o primjeni evolucijskog računanja u kriptologiji. Dodatno, ima nekoliko godina radnog iskustva u industriji i državnim sigurnosnim tvrtkama.

Izvanredni je professor u grupi za sigurnost na Radboud sveučilištu u Nijmegenu, Nizozemska. Njegovi istraživački interesi su sigurnost/kriptografija, strojno učenje i evolucijsko računanje. Prije znanstveno-nastavnog zvanja izvanrednog profesora, Stjepan je bio docent na Tehničkom sveučilištu u Delftu, Nizozemska, postdoktorand na MIT-u u Sjedinjenim Američkim Državama, i Katoličkom sveučilištu u Leuvenu, Belgija. Do sada je održao više od 30 pozvanih predavanja na konferencijama i ljetnim školama te objavio više od 140 recenziranih radova.

Stjepan je član organizacijskog odbora Međunarodne ljetne škole kriptografije, te član programskioh odbora i recenzent nekoliko konferencija i časopisa te viši član IEEE-a.

Dedicated to my wife, parents, sister, and grandparents for all their love and support.

To my son, Jakov

Acknowledgements

I want to take this opportunity to express my gratitude and thank all the people who helped me throughout my life, without which I would be unable to reach this goal and complete my thesis. This thesis is not only about my success but the success of all the people who supported me over the years.

First and foremost, I extend my thanks to the Almighty God for His endless blessings, wisdom, strength, and humility that have made this accomplishment possible. Without Him, I am nothing.

Second, I want to thank my beloved wife Katarina for her support and understanding, especially when I lost motivation or felt lost. Thank you for being there for me when I worked late or on weekends. Thank you for accompanying me to conferences and being there when I needed you. Thank you for walking with me through life and making me a better person.

I want to thank my parents, Jasminka and Milovan, who supported me my entire life. Thank you for all the love and care you gave me throughout my life. Thank you for all the sacrifices you made for me. Thank you for all the support in difficult times. Thank you for motivating me to pursue my dreams, however hard it sometimes might have been. Thank you for everything you did for me. You always wished you could do more to help me, but you helped me more than you could ever imagine. Thanks to my sister Andrea, who always believed in me and was my advisor in difficult times. I also want to thank my grandparents, Božica, Zorka, and Stjepan, for my beautiful childhood and all their time with me. I will forever treasure those moments. Thank you for all the talks and stories you told me, which helped me to take my mind off the many difficulties I faced. Thank you for all your love and support and for always encouraging me to do my best and never give up.

I would not have completed this thesis without the support of my mentors, Professor Domagoj Jakobović and Associate Professor Stjepan Picek. Professor Jakobović has been my mentor since my undergraduate studies, and he is the one who instilled in me a love for evolutionary computing. I want to thank him for all the time and effort that he invested in me. I owe Associate Professor Picek immense gratitude for his invested time, trust, and firm support during the research. He is responsible for guiding me in moments of uncertainty. Not only is he a great mentor, but he has taught me a lot of everyday things. Thank you so much - you are the one who drastically positively influenced me and my life.

Furthermore, I want to thank Ivan Bešlić and Zlatko Hrkać, the directors of Sofascore. They made it possible for me to study while working in the first year of my doctoral studies.

My sincere appreciation goes to my colleagues at the Department of Electronics, Microelectronics, Computer, and Intelligent Systems, especially Igor Stančin, Maja Strika, Marko Đurasević, and Rudolf Lovrenčić, who have been supportive and helpful throughout my re-

search. It was always a pleasure to talk, laugh, cry, and have coffee with you. Additionally, I would like to thank all the colleagues with whom I collaborated: Alan Jović, Alberto Leporati, Annelie Heuser, Ante Đerek, Axel Legay, Claude Carlet, Julian Miller, Julio Hernandez-Castro, Juraj Fulir, Luca Mariot, Marko Đurasević, and Tania Richmond.

To all my friends, especially Andrija Krilić, Boris Komunjer, Herman Zvonimir Došilović, Jakov Vidulić, Juraj Fulir, Karmen Dežmar Kreinz, Kristijan Štruml, Luka Milić, Martin Jakopec, Matea Vidulić, and Nikola Sekulić, I extend my heartfelt thanks for their unwavering support and friendship over the years.

Finally, I want to express my gratitude to my loyal and loving dog Žuži, who has been a constant companion during my thesis writing process. Though she may never read this, her company and unwavering love have been a great comfort to me.

Thank you all for being a part of this journey with me.

Abstract

In the field of cryptography, Boolean functions and their generalizations, known as vectorial Boolean functions or S-boxes, play a crucial role in symmetric key cryptography. The use of carefully selected S-boxes is essential for ensuring the security of ciphers, as without them, the ciphers would be susceptible to attacks. Symmetric key cryptography can be classified into stream ciphers and block ciphers, both of which use Boolean functions (including vectorial Boolean functions) for different purposes but with the common goal of improving cipher resilience against various cryptanalyses. Since other ciphers have additional requirements for Boolean functions or S-boxes, designing a cipher is a complex process that requires adherence to multiple principles to create a strong cipher. During the design phase, one must consider the properties of cryptographic primitives and the complete cipher and test them against many possible attacks to ensure their strength. While computers are heavily used in the design process for testing specific aspects of the cipher, modern ciphers are exclusively designed by human experts. However, poor implementation choices can lead to side-channel leakage, making even mathematically secure ciphers vulnerable to attackers.

This thesis aims to achieve several objectives. Firstly, we demonstrate that it is possible to construct Boolean functions that satisfy the cryptographic criterion of non-linearity using a non-binary base. Secondly, we aim to build S-boxes with output dimensions smaller than input dimensions, meeting cryptographic criteria such as non-linearity and differential uniformity. The first two goals are considered challenging optimization problems, which we solve using evolutionary computing. Thirdly, we show how to automatically construct a symmetric block algorithm without requiring the intervention of human experts. Finally, we explore how to make side-channel attacks more successful by utilizing machine learning and neuroevolutionary computing.

Keywords: Boolean functions, S-boxes, bent functions, evolutionary algorithms, automatic cipher construction, symmetric cryptography, side-channel attack, machine learning, semi-supervised learning, neuroevolution

Prošireni sažetak

Strojno učenje i evolucijsko računarstvo u oblikovanju i analizi kriptografskih algoritama sa simetričnim ključem

Uvod

U 21. stoljeću digitalna tehnologija snažno je ušla u život pojedinaca. Gotovo je nemoguće zamisliti život današnjice bez uporabe računala. Čovječanstvo je izabralo vlastiti razvoj i napredak temeljiti na digitalnoj tehnologiji. Povećanjem razine digitalizacije društva pojavljuju se i prijetnje koje mogu ugroziti informacijsku sigurnost. Jedna od komponenti informacijske sigurnosti jest kriptografija. Pojednostavljeno, kriptografija ili tajnopis je umijeće pisanja poruka u takvom obliku da samo onaj kome su namijenjene može pročitati. U digitalnom komunikacijskom lancu, čiji dionici su većina ljudske populacije, postoje pošiljatelji poruke, primatelji poruke te napadači. Napadači su one osobe koje namjerno pokušavaju presresti poruku te je pročitati. Ako pošiljatelj poruku nije dovoljno dobro osigurao neovlašteno čitanje poruke, posljedice mogu biti katastrofalne.

Povijest kriptografije seže u doba antike. Već su stari Grci znali da informacija ima najveću cijenu. Stoljećima su se razvijale sofisticirane metode koje su neprestano unaprjeđivale načine skrivanja poruka. Paralelno s razvojem metoda skrivanja poruka, napadači su razvijali sve domišljatije metode kako pročitati skrivene poruke. Ta igra skrivanja i razotkrivanja poruka traje i dan danas. Svakodnevno svjedočimo medijskim vijestima koje objavljuju razne tajne podatke. Svijest pojedinaca o posljedicama otkrivanja tajnih digitalnih informacija postaje sve jača jer upravo oni sve češće postaju žrtvama.

Komunikacijski lanac ne čine samo ljudi, već i strojevi. Većina digitalnih uređaja koja se danas proizvodi, spaja se na internet i komunicira s drugim strojevima ili ljudima. Budući da računala postaju sve manja i lakše prenosiva, s ciljem boljeg praćenja, nosiva računala se ugrađuju u/na životinje ili biljke. Općenito govoreći, živimo u svijetu opće komunikacije gdje gotovo svatko može komunicirati s bilo kime ili bilo čime.

Već je Alan Turing krajem 2. svjetskog rata pokazao da ljudi nisu učinkoviti napadači na skrivene poruke, već strojevi. Razotkrivanje skrivenih poruka vrlo je težak problem, a zadaća mu je napasti slabosti postupka ili algoritma kojim je poruka skrivena. Dugi niz stoljeća, da bi se osigurala tajnost poruka u komunikacijskom lancu, skrivao se kriptoalgoritam - postupak kojim je pošiljatelj sakrio poruku. U 19. stoljeću, nizozemski kriptograf Auguste Kerchoff rekao je da kriptosustav mora ostati siguran čak i ako je sve o sustavu poznato, osim tajnog ključa. Navedena izjava nadovezuje se na izjavu francuskog kriptografa Jean Robert du Carleta koji je rekao da samo tajni ključevi moraju ostati tajnima - "*ars ipsi secreta magistro*". U simetričnom kriptoalgoritmu glavna ideja napadača kod čitanja skrivenih poruka jest otkriti

ključ koji se koristio u algoritmu šifriranja. Jednom kada napadač otkrije tajni ključ, nesmetano može presretati i čitati poruke u komunikacijskom lancu između pošiljatelja i primatelja poruke.

Svi moderni kriptoalgoritmi javno su poznati, a kriptografi neprekidno poboljšavaju svojstva kriptoalgoritma. Prilikom izgradnje novog kriptografskog algoritma, ljudski eksperti moraju istovremeno zadovoljiti više kriptografskih kriterija da bi se algoritam smatrao sigurnim. Načelno govoreći, simetrične algoritme koje danas koristimo, možemo smatrati sigurnima. Međutim, igra skrivanja i razotkrivanja poruka prebačena je na drugi teren. Naime, umjesto da se danas napada ranjivost kriptoalgoritma, napada se implementacija kriptoalgoritma na uređaju na kojem se izvodi. Svaki stroj koji izvodi kriptoalgoritam proizvodi sporedne informacije koje mogu napadaču osigurati uspješan napad. Te sporedne informacije očituju se u potrošnji električne energije, proizvedenoj toplinskoj energiji ili zvuku, vremenskom čekanju na izvođenje iduće računske operacije i slično. Budući da su takvi napadi već poznati, prilikom oblikovanja kriptoalgoritma ne ulažu se samo napori u istovremeno zadovoljavanje kriptografskih kriterija, već se vodi računa i o implementaciji algoritma na stroju te skrivanju proizvedenih sporednih informacija. Međutim, sporedna informacija uvijek će postojati, samo je pitanje koliko će ta informacija biti vrijedna napadaču u otkrivanju tajnog ključa.

Evolucijsko računarstvo i strojno učenje dva su velika područja unutar umjetne inteligencije. Evolucijsko računarstvo pokazalo se kao moćan alat u rješavanju optimizacijskih problema. Upravo je istovremeno zadovoljavanje različitih kriptografskih kriterija optimizacijski problem. Dodatno, većina kriptografskih kriterija mogu se opisati kao optimizacijski problem, kao što je, na primjer, visoka nelinearnost. Postoje mnogobrojna istraživanja koja su potvrdila opravdanost, unatoč skepsi matematičara i kriptografa, korištenja algoritama evolucijskog računarstva u konstrukciji kriptografskih primitiva.

S druge strane, razvojem računalne moći, strojno učenje pokazuje se kao snažan alat u sigurnosnim napadima na sporedna svojstva uređaja. Naime, danas su profilirani napadi najmoćniji u kategoriji sigurnosnih napada na sporedna svojstva uređaja. U takvom napadu, napadač dobiva pristup uređaju za izgradnju preciznog modela strojnog učenja koji se koristi za napad na drugi istovjetan uređaj. Pritom se koristi pretpostavka da napadač ima vrlo velike mogućnosti u fazi profiliranja, dok je faza napada ograničena. Zadatak modela strojnog učenja jest naučiti prepoznavati sporednu informaciju uređaja - trag - te je povezati s dijelom tajnog ključa koji se nalazi na računalu na kojem se izvodi kriptoalgoritam.

Iako je u području primjene evolucijskog računarstva u konstrukciji kriptografskih primitiva učinjeno mnogo, postoje mnogobrojna neodgovorena pitanja. Osim što su se algoritmi evolucijskog računarstva koristili u izgradnji kriptografskih primitiva, dosad nije poznat pokušaj korištenja evolucijskog računarstva u automatskoj izgradnji kriptografskih algoritama. S druge strane, unatoč tome što postoji puno istraživanja u domeni profiliranih napada, postavlja se pitanje uspješnosti takvog napada pod uvjetom da je napadač tijekom faze profiliranja ograničen

brojem tragova. Posljednje, dosad su se u profiliranim sigurnosnim napadima na pozadinska svojstva uređaja koristili fundamentalni algoritmi strojnog učenja i neuronske mreže. Dosad ostaje otvoreno pitanje može li spoj evolucijskog računarstva i neuronskih mreža pospješiti učinkovitost napada.

Motivacija

Glavna motivacija disertacije jest ostvariti novi znanstveni doprinos u području oblikovanja i analize kriptografskih algoritama sa simetričnim ključem, koristeći strojno učenje i evolucijsko računarstvo, koja do sada u postojećoj literaturi nisu bila prikladno istražena. Područja od značaja u ovoj disertaciji su:

1.konstrukcije kriptografskih primitiva,
   (a)konstrukcija visoko nelinearnih Booleovih funkcija u binarnoj i kvaternarnoj bazi korištenjem algoritama evolucijskog računarstva,
   (b)konstrukcija S-kutija izri čito manje dimenzije izlaza nego ulaza korištenjem algoritama evolucijskog računarstva,
2.automatska izgradnja kriptografskog algoritma sa simetri čnim ključem korištenjem algoritama evolucijskog računarstva,
3.izgradnja modela strojnog u čenja u profiliranom sigurnosnom napadu na sporedna svojstva uređaja,
   (a)izgradnja modela korištenjem polunadziranog u čenja,
   (b)korištenje neuroevolucije u modelu višeslojnog perceptrona i duboke neuronske mreže.

Booleove funkcije predstavljaju ključan faktor u oblikovanju kriptografskih algoritama. Iako postoje mnogobrojne primjene Booleovih funkcija u kriptografiji, naš fokus je na kriptografskim algoritmima tokova podataka. Da bi Booleove funkcije imale praktičnu primjenu u kriptografiji, one moraju zadovoljavati mnoga kriptografska svojstva. Jedno od tih svojstava jest visoka nelinearnost, kojom se pospješuje obrana od linearne kriptoanalize. Svojstvo nelinearnosti govori koliko je funkcija udaljena od svih afinih funkcija. Funkcija čija udaljenost je najveća od svih afinih funkcija naziva se maksimalno nelinearna funkcija (engl. *bent function*). Ovaj problem jest kombinatorički problem jer prostor pretrage raste eksponencijalno, i s obzirom na broj ulaznih varijabli n, prostor pretrage iznosi 2^{2^n}. Postoji više radova koji su konstrukciju kriptografskih primitiva promatrali kao kombinatoričku optimizaciju te rješavali navedeni problem algoritmima evolucijskog računarstva. Prvotna motivacija jest pokušati konstruirati maksimalno nelinearne funkcije dimenzije ulaza do $n = 12$, koristeći genetski algoritam i genetsko programiranje, te usporediti oba rezultate oba algoritma s teorijskim vrijednostima. Osim visoke nelinearnosti, jedan od kriptografskih kriterija jest balansiranost Booleove funkcije. Stoga, nadogradnja početne ideje jest konstruirati visoko nelinearnu balansiranu

Booleovu funkciju. Budući da su rezultati pokazali da se uspješno može konstruirati visoko nelinearna balansirana Booleova funkcija, postavilo se pitanje kako cijeli proces konstrukcije učiniti uspješnijim za veće ulazne dimenzije n. Zbog toga smo odlučili provesti eksperimente s kvaternarnim Booleovim funkcijama - funkcijama s vrijednostima 0, 1, 2 i 3. Posljedica promjene baze jest povećanje prostora pretrage visoko nelinearne funkcije 4^{4^n}, ali ako se u takvom prostoru pronađe visoko nelinearna funkcija, tada postoji mapiranje u binarnu Booleovu funkciju ulazne dimenzije $2n$.

Prirodna nadogradnja na Booleove funkcije su vektorske Booleove funkcije ili supstitucijske kutije (S-kutije). S-kutije imaju značajnu ulogu u kriptografskim algoritmima sa simetričnim ključem. Iako postoje različite strategije dizajna kriptografskih algoritama, zajedničko im je da implementiraju principe konfuzije i difuzije. Princip konfuzije znači da izlaz iz kriptografskog algoritma ne smije ovisiti o ulazu, odnosno da ne postoji linearna veza između ulaza i izlaza u kriptografski algoritam. Princip difuzije znači da promjena jednog bita na ulazu u kriptografski algoritam za posljedicu ima promjenu više bitova na izlazu iz algoritma. Jedan od načina osiguravanja principa konfuzije jest korištenje S-kutija. Jedno od praktičnih neodgovorenih pitanja jest kako konstruirati maksimalno nelinearnu S-kutiju kod koje je dimenzija izlaza nužno manja od dimenzije ulaza. Dodatan otvoreni problem jest pitanje postojanja S-kutija dimenzija $(n, n-k)$, gdje je $k = 2$, a čija diferencijalna uniformnost je manja od 8 ako je dimenzija ulaza veća ili jednaka 5. Ovo pitanje zanimljivo je s optimizacijskog stajališta, a s druge strane, diferencijalna uniformnost povezana je s diferencijalnom kriptoanalizom. Za rješavanje ovih problema, također se koriste algoritmi evolucijskog računarstva.

Nakon istraživanja i eksperimentiranja s kriptografskim primitivima, postavilo se pitanje o mogućnosti automatiziranog dizajna kriptografskog algoritma. Dizajniranje kriptografskog algoritma vrlo je složen proces jer dizajneri algoritma moraju slijediti više kriptografskih principa ne bi li stvorili kriptoalgoritam otporan na napade. Tijekom dizajniranja, mora se voditi računa o svojstvima korištenih kriptografskih primitiva te o potpunom algoritmu. Da bi se dizajnirani kriptografski algoritam pokazao dovoljno otpornim na napade, nad algoritmom se provode različiti napadi, kao što su diferencijalna ili linearna kriptoanaliza. Do danas, kriptografske algoritme isključivo dizajniraju ljudski eksperti. Iako od 2000. godine postaje aktivno područje kriptografije bazirane na neuronskim mrežama, jedan od nedostataka u takvom pristupu jest manjak interpretabilnosti. Budući da dosad nitko nije razmatrao korištenje evolucijskih algoritama u konstrukciji kriptografskih algoritama, ovaj problem ispituje moguća ograničenja evolucijskih algoritama u ovoj domeni. Dodatno, budući da unaprijed nisu postavljena nikakva ograničenja na dizajn kriptografskog algoritma, postoji mogućnost pronalaska novih primitiva. Također, s obzirom na to da se sustav sastoji od pošiljatelja, primatelja te napadača, postavlja se pitanje kakvu strategiju će koristiti napadač s ciljem otkrivanja tajnog ključa. Još jedna od motivacija za razvoj automatiziranog dizajna kriptografskog algoritma jest mogućnost prilagodbe na

nove sigurnosne prijetnje. Prilagodba algoritma svodi se na promjenu optimizacijske funkcije algoritma evolucijskog računanja.

Nakon početne motivacije konstrukcije kriptografskih primitiva i kriptografskog algoritma sa simetričnim ključem uporabom algoritama evolucijskog računarstva, stavljamo se u poziciju napadača koji pokušava doći do tajnog ključa u simetričnom kriptografskom algoritmu koristeći sporedna svojstva uređaja. Kod sigurnosnih napada na pozadinska svojstva uređaja, profilirani napadi smatraju se najsnažnijim napadima. Smatra se da u fazi profiliranja, napadač ima neograničene mogućnosti, odnosno može imati veliku količinu tragova na temelju kojih može naučiti model strojnog učenja. Iz ovoga proizlazi motivacija za istraživanjem slučaja kod kojeg je napadač u fazi profiliranja ograničen, a u fazi napada može prikupiti dovoljno veliku količinu tragova. Zbog takvog scenarija, proučava se primjena polunadziranog učenja tako da se u fazi profiliranja koristi mali broj označenih tragova, a zatim se model neprestano poboljšava koristeći neoznačene tragove iz faze napada. Koliko je poznato, dosad je napravljeno jedno istraživanje s identičnim scenarijem, ali zaključci provedenog istraživanja u potpunoj su suprotnosti s našim rezultatima. Istraživanje je provedeno na javno dostupnim skupovima podataka dobivenih na temelju fizičke implementacije kriptografskog algoritma AES.

Posljednji dio istraživanja motiviran je korištenjem neuronskih mreža u profiliranim sigurnosnim napadima na sporedna svojstva uređaja. Izbor aktivacijske funkcije može snažno utjecati na učinkovitost rada neuronske mreže. Mnogi radovi neprestano ispituju i predlažu nove aktivacijske funkcije čija svojstva mogu doprinijeti uspješnosti učenja modela neuronskih mreža. Koliko znamo, dosad nitko nije pokušao razviti aktivacijsku funkciju za neuronske mreže za sigurnosni napad na sporedna svojstva uređaja. Rezultat genetskog programiranja jest simbolički prikaz funkcije koja se može koristiti kao aktivacijska funkcija. Spoj algoritma evolucijskog računanja i neuroračunarstva naziva se neuroevolucijsko računarstvo. Početni dobiveni rezultati opravdali su korištenje neuroevolucije, kod višeslojnog perceptrona i konvolucijske neuronske mreže, kod napada na sporedna svojstva uređaja. Za eksperimentiranje je korišten javno dostupan skup podataka fizičke implementacije kriptografskog algoritma AES.

Pregled disertacije

Disertacija je podijeljena na osam poglavlja: uvod, teorijske pretpostavke o evolucijskom računarstvu, strojnom učenju, kriptografiji i implementacijskim napadima, konstrukcija Booleovih funkcija, konstrukcija vektorskih Booleovih funkcija, automatska izgradnja kriptografskog algoritma, strojno učenje u sigurnosnim napadima na sporedna svojstva uređaja, neuroevolucija u sigurnosnim napadima na sporedna svojstva uređaja i zaključak.

Prvo poglavlje daje kratak uvod u disertaciju. U navedenom poglavlju ukratko je izložena motivacija za proučavanje problema kod kriptografskih algoritama sa simetričnim ključem. Kroz poglavlje je istaknuto nekoliko otvorenih pitanja u tom području koja su proučavana u

okviru disertacije. U poglavlju je također dan i pregled izvornog znanstvenog doprinosa koji je ostvaren u sklopu disertacije. Konačno, poglavlje je zaključeno kratkim pregledom disertacije.

U drugom poglavlju ukratko su opisani pojmovi i teorijski koncepti korišteni u disertaciji. Potpoglavlje o evolucijskom računarstvu definira pojam optimizacije te opisuje evolucijske operatore korištene u evolucijskim algoritmima. Opisuje se rad genetskog algoritma, genetskog programiranja i Kartezijskog genetskog programiranja. Za svaki od algoritama objašnjava se prikaz rješenja. U potpoglavlju o strojnom učenju opisuju se vrste učenja: nadzirano učenje, nenadzirano učenje, polunadzirano učenje i podržano učenje. Ukratko su opisani fundamentalni algoritmi korišteni u disertaciji: algoritam naivnog Bayesa, stroj potpornih vektora, višeslojni perceptron te konvolucijska neuronska mreža. Dodatno je objašnjena uloga aktivacijskih funkcija u neuronskim mrežama. U potpoglavlju o kriptografiji posebno se opisuju svojstva kriptografskog algoritma sa simetričnim i asimetričnim ključem. Posljednje potpoglavlje objašnjava pojam implementacijskih napada, razlog nastanka sporednih svojstva uređaja na kojima se kriptografski algoritam izvodi te razliku između profiliranih i neprofiliranih sigurnosnih napada.

Treće poglavlje opisuje konstrukciju Booleovih funkcija korištenjem genetskog algoritma i genetskog programiranja, s ciljem optimizacije svojstva visoke nelinearnosti i balansiranosti. U poglavlju se razmatraju Booleove funkcije binarne i kvaternarne baze. U obje baze koristi se Walsh-Hadamardova transformacija funkcija s ciljem pronalaska maksimalno nelinearnih funkcija (*bent functions*). U ovom poglavlju ostvaren je izvorni znanstveni doprinos u konstrukciji Booleovih funkcija s prilagodljivim kriptografskim svojstvima. Pokazano je da unatoč tome što je prostor pretrage maksimalno nelinearnih funkcija u kvaternarnoj bazi puno veći, genetsko programiranje uspješno pronalazi kvaternarne *bent* funkcije ulazne dimenzije n koje se Grayevim mapiranjem transformiraju u Booleove funkcije ulazne dimenzije $2n$. Na taj način pokazana je uspješna primjena genetskog programiranja u pronalaženju *bent* Booleovih funkcija do ulazne dimenzije 16.

U četvrtom poglavlju opisuje se konstrukcija S-kutija različitih kriptografskih svojstava. Prvi problem koji se rješava jest konstrukcija *bent(n,m)* S-kutije kod koje je $m \leq \frac{n}{2}$, odnosno izlazna dimenzija striktno je manja od ulazne dimenzije. Navedeni problem rješava se koristeći genetski algoritam i genetsko programiranje. U genetskom algoritmu eksperimentira se različitim prikazima: niz bitova i niz brojeva s pomičnom točkom. Optimiziraju se tri različite funkcije, od kojih je jedna kompozicija derivacija Booleovih funkcija. Eksperimenti su provedeni do ulazne dimenzije 12. Rezultati istraživanja potvrđuju da evolucijski algoritmi mogu uspješno konstruirati S-kutiju različitih ulaznih dimenzija, gdje najbolje rezultate postiže genetsko programiranje. Dodatno, zaključujemo da genetsko programiranje postiže najučinkovitije rezultate zbog načina prikaza rješenja. Dodatno, istražujemo mogu li algoritmi evolucijskog računarstva generirati S-kutiju dimenzija $(n, n-2)$ i kod koje je vrijednost diferencijalne uni-

formnosti jednaka 6. Zbog veličine prostora pretrage, ovaj problem pokazuje se vrlo izazovnim. Globalne optimume postižemo za dimenzije $(4,2)$ i $(5,3)$, a zanimljivo je da optimume u različitim dimenzijama postižu različiti prikazi rješenja. Prilikom rješavanja ovog problema, eksperimentira se s različitim prikazima rješenja: prikaz cijelim brojevima, permutacijski prikaz rješenja, četverostruka permutacija te prikaz brojevima s pomičnom točkom. Rezultati ukazuju da najbolja rješenja postiže genetsko programiranje te genetski algoritam s prikazima rješenja cijelim brojem ili brojem s pomičnom točkom. U okviru ovog poglavlja ostvaren je izvorni znanstveni doprinos u konstrukciji vektorskih Booleovih funkcija s prilagodljivim kriptografskim svojstvima.

Peto poglavlje istražuje automatsku konstrukciju kriptografskog algoritma sa simetričnim ključem. Problem se promatra kao optimizacijski problem kod kojeg se želi konstruirati takav kriptografski algoritam koji ima visoku nelinearnost, da bi bio otporan na linearnu kriptoanalizu, poštuje princip difuzije te je bijektivan. Algoritam koristi dvorazinsku optimizaciju, kod koje je optimizacijska funkcija pošiljatelja vanjska optimizacijska funkcija, a optimizacijska funkcija napadača unutarnja optimizacijska funkcija. U optimizacijskom postupku koristi se Kartezijsko genetsko programiranje zbog interpretabilnosti, paralelne izgradnje rješenja te inherentno riješenog problema prekomjernog rasta jedinke. Razmatra se pet scenarija, s obzirom na mogućnosti pošiljatelja, primatelja te napadača. Napadač koristi napad po modelu poznatog otvorenog teksta. U eksperimentima se koriste umjetno stvoreni blokovi teksta veličine 4 i 8 bitova. Rezultati pokazuju da je prosječna uspješnost napadača u otkrivanju tajnog ključa jednaka nasumičnom pogađanju, iz čega zaključujemo da algoritmi evolucijskog računarstva imaju potencijal biti korišteni u automatskoj izgradnji kriptografskog algoritma. U ovom poglavlju ostvaren je izvorni znanstveni doprinos automatske izgradnje kriptografskih algoritama primjenom dinamike napadača i obrane u sigurnosnoj domeni.

U šestom poglavlju razmatra se profilirani sigurnosni napad na sporedna svojstva uređaja. Istražuje se scenarij u kojem je napadač ograničen u fazi profiliranja s malim brojem označenih tragova, dok u fazi napada ima mogućnost prikupljanja većeg broja tragova. U poglavlju se istražuje kako polunadzirano učenje može pospješiti učinkovitost modela. Kao modeli koriste se algoritam naivnog Bayesa, stroj potpornih vektora, a napadač koristi napad predloškom (engl. *template attack*). Eksperimenti su provedeni na javno dostupnim skupovima podataka DPAv2 i DPAv4 fizičke implementacije algoritma AES. Rezultati potvrđuju početnu hipotezu, a tragovi iz faze napada mogu, polunadziranim učenjem, učiniti model učinkovitijim. U ovom poglavlju ostvaren je izvorni znanstveni doprinos području algoritama strojnog učenja u sigurnosnim napadima na sporedna svojstva uređaja.

Sedmo poglavlje razmatra mogućnost da se u profiliranim sigurnosnim napadima na sporedna svojstva uređaja koriste neuronske mreže: višeslojni perceptron i konvolucijska neuronska mreža. Budući da izbor aktivacijske funkcije uvelike utječe na učinkovitost rada neuronske

mreže, u poglavlju se koristi genetsko programiranje kao optimizacijski evolucijski algoritam. Zadatak genetskog programiranja jest stvoriti prihvatljivu aktivacijsku funkciju koja smanjuje funkciju gubitka, odnosno pospješuje učinkovitost napada. Istraživanje je provedeno na javno dostupnom skupu podataka ASCAD fizičke implementacije algoritma AES. Eksperimenti pokazuju da evolucija aktivacijskih funkcija ostvaruje bolje rezultate na jednostavnijem skupu podataka što upućuje na mogućnost dodatnih napora u istraživanju. U ovom poglavlju ostvaren je izvorni znanstveni doprinos korištenjem postupaka neuroevolucije za optimiziranje arhitekture neuronskih mreža u sigurnosnoj domeni.

Zaključak

Glavni cilj disertacije je pokazati primjenu strojnog učenja i evolucijskog računarstva u oblikovanju i analizi kriptografskih algoritama sa simetričnim ključem. U sklopu disertacije ostvaren je znanstveni doprinos koji se sastoji od četiri točke:

- •Evolucijski algoritmi za konstrukciju skalarnih i vektorskih Booleovih funkcija s prilagodljivim kriptografskim svojstvima
- •Automatska izgradnja kriptografskih algoritama primjenom dinamike napada ča i obrane u sigurnosnoj domeni
- •Algoritmi strojnog u čenja u sigurnosnim napadima na sporedna svojstva uređaja
- •Postupci neuroevolucije za optimiziranje arhitekture neuronskih mreža u sigurnosnoj domeni

Kroz postignute rezultate može se zaključiti kako evolucijski algoritmi, u određenim slučajevima, mogu biti jednako učinkoviti kao i konstruktivni algoritmi za konstrukciju Booleovih i vektorskih Booleovih funkcija. S obzirom na optimizacijski problem, različiti algoritmi i različiti prikazi rješenja pokazuju bolje ponašanje, a što je očekivano s obzirom na *No Free Lunch* teorem. Korištenjem Kartezijskog genetskog programiranja uspješno je automatski izgrađen kriptografski algoritam u kojoj napadač nije bio uspješniji od nasumičnog pogađanja ključa. Polunadzirano učenje pokazuje se kao uspješna metoda učenja u profiliranim sigurnosnim napadima na sporedna svojstva uređaja, kod kojih je napadač ograničen brojem tragova u fazi profiliranja. Konačno, korištenje neuroevolucije s ciljem izgradnje aktivacijske funkcije neuronskih mreža, može doprinijeti uspješnijem profiliranom sigurnosnom napadu na sporedna svojstva uređaja.

Ključne riječi: Booleove funkcije, S-kutije, funkcije s najvećom nelinearnošću (engl. *bent functions*), evolucijski algoritmi, automatizirana izgradnja kriptografskih algoritma sa simetričnim ključem, simetrična kriptografija, sigurnosni napadi na sporedna svojstva uređaja (engl. *side-channel attacks*), strojno učenje, polunadzirano učenje, neuroevolucija

Contents

Chapter 1

Introduction

In the 21st century, digital technology has firmly entered the lives of individuals. It is almost impossible to imagine life today without the use of computers. Humanity has chosen to base its development and progress on digital technology. With the increase in the level of digitization of society, challenges appear that can threaten information security. One of the components of information security is cryptography. Cryptography is the art of writing messages in such a form that only the intended recipient can read them. In the digital communication chain, whose stakeholders are the majority of the human population, there are message senders, receivers, and attackers. Attackers are those people who deliberately try to intercept the message and read it. If the message's sender has not sufficiently secured the unauthorized reading of the message, the consequences can be catastrophic.

The history of cryptography goes back to ancient times. The ancient Greeks already knew that information had the highest price. For centuries, sophisticated methods were developed that constantly improved the ways of hiding messages. In parallel with the development of methods of hiding messages, attackers have been growing increasingly ingenious methods to read hidden messages. That game of hiding and revealing messages continues to this day. Every day we witness media news that publishes various secret information. The awareness of individuals about the consequences of revealing confidential digital data is becoming more vital because it is they who are becoming victims more and more often.

The communication chain consists not only of people but also of machines. Most digital devices connect to the Internet and communicate with other machines or people. As computers are getting smaller and more portable, wearable computers are being implanted in/on animals or plants for better monitoring. Generally speaking, we live in a world of general communication where almost anyone can communicate with anyone or anything.

At the end of World War II, Alan Turing already showed that humans are not effective attackers of hidden messages, but machines are. Uncovering hidden messages is a challenging problem, and its task is to attack the weaknesses of the procedure or algorithm by which the

message is hidden. For many centuries, to ensure the secrecy of messages in the communication chain, a crypto-algorithm was hidden - the procedure by which the sender hid the message. In the 19th century, Dutch cryptographer Auguste Kerckhoff said that a cryptosystem must remain secure even if everything about the system is known except the secret key. The statement above follows the French cryptographer Jean Robert du Carlet, who said that only private keys must remain secret - "*ars ipsi secreta magistro*." In a symmetric crypto-algorithm, the attacker's main idea when reading hidden messages is to discover the key used in the encryption algorithm. Once an attacker finds the secret key, he can easily intercept and read the messages in the communication chain between the sender and receiver of the message.

All modern crypto algorithms are publicly known, and cryptographers are constantly improving crypto algorithms' properties. When building a new cryptographic algorithm, human experts must simultaneously satisfy multiple cryptographic criteria for the algorithm to be considered secure. We hope the symmetric algorithms we use today are safe. However, the game of hiding and revealing the messages have been transferred to another field. Indeed, instead of attacking the vulnerability of a crypto algorithm, one attacks the implementation of the crypto algorithm on the device on which it runs. Any machine running a crypto algorithm produces secondary information that can provide an attacker with a successful attack. This secondary information is reflected in the consumption of electricity, produced thermal energy or sound, time waiting for the next calculation operation to be performed, etc. Since such attacks are already known, when designing a crypto algorithm, efforts are made to satisfy cryptographic criteria simultaneously. Measures are also taken to implement the algorithm on the machine and hide the secondary information produced. However, secondary information will always exist. The only question is how valuable this information will be to an attacker in discovering the secret key.

Evolutionary computing and machine learning are two major fields within artificial intelligence. Evolutionary computing has proven to be a powerful tool in solving optimization problems. It is precisely the simultaneous satisfaction of different cryptographic criteria that is an optimization problem. Additionally, most cryptographic criteria can be described as an optimization problem, such as, for example, high nonlinearity. Numerous studies have confirmed the justification of using evolutionary computing algorithms in constructing cryptographic primitives despite the skepticism of mathematicians and cryptographers.

On the other hand, with the development of computing capability, machine learning is proving to be a powerful tool in security attacks against secondary device properties. Indeed, today profiled attacks are the most powerful side-channel attacks. In such an attack, an attacker gains access to a device to build a precise machine learning model used to attack another identical (similar) device. In doing so, the assumption is made that the attacker has many possibilities in the profiling phase, while the attack phase is limited. The task of the machine learning model

is to learn to recognize the secondary information of the device - the trace - and connect it with part of the secret key located on the computer on which the crypto algorithm is executed.

Although much has been done in evolutionary computing in constructing cryptographic primitives, many unanswered questions exist. Apart from evolutionary computing algorithms being used in the construction of cryptographic primitives, there is no known attempt to use evolutionary computing in the automatic construction of cryptographic algorithms. On the other hand, even though there is a lot of research in the domain of profiling attacks, the question arises of the success of such an attack, provided that the number of traces limits the attacker during the profiling phase. Finally, fundamental machine learning algorithms and neural networks have been used in profiled side-channel attacks. So far, it remains an open question whether the combination of evolutionary computing and neural networks can improve the effectiveness of attacks.

1.1 Research Motivation

The main motivation of the dissertation is to achieve new scientific contribution in the design and analysis of symmetric key cryptographic algorithms, using machine learning and evolutionary computing, which have not been properly investigated in the existing literature. The areas of importance in this dissertation are:

1.construction of cryptographic primitives,
   (a)construction of bent Boolean functions in the binary and quaternary base using algorithms of evolutionary computing,
   (b)construction of S-boxes with explicitly smaller output than input dimensions using evolutionary computing algorithms,
2.automatic construction of a cryptographic algorithm with a symmetric key using evolutionary computing algorithms,
3.building a machine learning model in a profiled side-channel attack,
   (a)model building using semi-supervised learning,
   (b)using neuroevolution in a multi-layer perceptron and deep neural network model.

Boolean functions stand as a crucial factor in the design of cryptographic algorithms. Although there are many applications of Boolean functions in cryptography, our focus is on stream cryptographic algorithms. Boolean functions must satisfy many cryptographic properties to have practical application in cryptography. One of these properties is high nonlinearity, which improves defense against linear cryptanalysis. The nonlinearity property tells how far the function is from all affine functions. The function whose distance is the largest of all affine functions is called a maximally nonlinear function (*bent function*). This problem is combinatorial because the search space grows super-exponentially, and given the number of input variables n,

the search space is 2^{2^n}. Several works looked at the construction of cryptographic primitives as combinatorial optimization and solved the mentioned problem with evolutionary computing algorithms. The original motivation is to construct bent input dimension functions up to $n = 12$, using genetic algorithm and genetic programming, and to compare the results of both algorithms with theoretical values. In addition to high nonlinearity, one of the cryptographic criteria is the balancedness of the Boolean function. Therefore, an extension of the initial idea is to construct a highly nonlinear balanced Boolean function. Since the results showed that evolutionary algorithms could successfully build a highly nonlinear balanced Boolean function, the question arose about how to make the entire construction process more successful for larger input dimensions n. That is why we decided to experiment with quaternary Boolean functions with output values 0, 1, 2, and 3. The consequence of the base change is the increase of the search space of the highly nonlinear function 4^{4^n}. Still, if a highly nonlinear function is found in such a space, then there is a mapping into a binary Boolean function of input dimension $2n$.

A natural extension of Boolean functions is vectorial Boolean functions or substitution boxes (S-boxes). S-boxes play a significant role in symmetric key cryptographic algorithms. Although cryptographic algorithms have different design strategies, they all have in common that they implement the principles of confusion and diffusion. The principle of confusion means that the output of the cryptographic algorithm must not depend on the input. There is no linear connection between the cryptographic algorithm's input and output. The principle of diffusion means that a change of one bit at the input to the cryptographic algorithm results in a change of several bits at the algorithm's output. One way to ensure the principle of confusion is to use S-boxes. One of the practical unanswered questions is how to construct a maximally nonlinear S-box where the dimension of the output is necessarily smaller than the dimension of the input. An additional open problem is the question of the existence of S-boxes of sizes $(n, n-k)$, where $k = 2$, and whose differential uniformity is less than 8 if the input dimension is greater than or equal to 5. This question is interesting from an optimization point of view, and on the other hand, differential uniformity is related to differential cryptanalysis. Evolutionary computing algorithms are also used to solve these problems.

After researching and experimenting with cryptographic primitives, the question arose about the possibility of automated cryptographic algorithm design. Designing a cryptographic algorithm is very complex because algorithm designers must follow multiple cryptographic principles to create a cryptographic algorithm resistant to attacks. During the design, the properties of the cryptographic primitives are used, and designers must consider the complete algorithm. Various attacks are performed on the algorithm to show that the designed cryptographic algorithm is sufficiently resistant to attacks, such as differential or linear cryptanalysis. To date, cryptographic algorithms are exclusively designed by human experts. Although cryptography based on neural networks has become an active area since 2000, one of the disadvantages of such an

approach is the lack of interpretability[1]. Since no one has considered the use of evolutionary algorithms in constructing cryptographic algorithms, this problem examines the possible limitations of evolutionary algorithms in this domain. Additionally, since no constraints are set in advance on the design of the cryptographic algorithm, there is the possibility of finding new primitives. What is more, considering that the system consists of a sender, a receiver, and an attacker, the question arises as to what strategy the attacker will use to discover the secret key. Another motivation for the development of automated cryptographic algorithm design is the ability to adapt to new security threats. Adaptation of the algorithm is reduced to changing the optimization function of the algorithm of evolutionary computation.

After the initial motivation of the construction of cryptographic primitives and a cryptographic algorithm with a symmetric key using evolutionary computing algorithms, we put ourselves in the position of an attacker who tries to get the secret key in a symmetric cryptographic algorithm using secondary properties of the device. In side-channel attacks, profiling attacks are considered the most powerful attacks. It is considered that in the profiling phase, the attacker has unlimited possibilities, i.e., one can have a large number of traces based on which he can train the machine learning model. From this comes the motivation to investigate the case where the attacker is limited in the profiling phase and can collect a sufficiently large amount of traces in the attack phase. In such a scenario, the application of semi-supervised learning is studied by using a small number of labeled traces in the profiling phase and then continuously improving the model using unlabeled traces from the attack phase. As far as we know, only one research has been done with an identical scenario, but the conclusions of the conducted research contradict our results. We conducted the research on publicly available data sets obtained based on the physical implementation of the AES cryptographic algorithm.

The last part of the research is motivated by using neural networks in profiled side-channel attacks. The choice of the activation function can strongly influence the neural network's performance. Many papers examine and propose new activation functions whose properties can contribute to the learning success of neural network models. To our knowledge, no one has attempted to develop an activation function for neural networks for a security attack on secondary device properties. The result of genetic programming is a symbolic representation of a function that can be used as an activation function. The combination of evolutionary computing algorithms and neurocomputing is called neuroevolutionary computing. The initial results justified using neuroevolution, with a multilayer perceptron and a convolutional neural network in side-channel attacks. For experimentation, a publicly available dataset of the physical implementation of the AES cryptographic algorithm was used.

1.2 Outline of the Thesis

The dissertation is divided into eight chapters: introduction, background on evolutionary computing, machine learning, cryptography, and implementation attacks, construction of Boolean functions, construction of vectorial Boolean functions, automatic construction of a cryptographic algorithm, machine learning in side-channel attacks, neuroevolution in side-channel attacks, and conclusion.

The first chapter briefly introduces the dissertation and presents the motivation for studying problems with cryptographic algorithms with a symmetric key. Throughout the chapter, several open questions in that area were highlighted, which were studied in the framework of the dissertation. The chapter also provides an overview of the original scientific contribution as part of the dissertation. Finally, the chapter is concluded with a brief overview of the dissertation.

The terms and theories used in the dissertation are briefly described in the second chapter. The subsection on evolutionary computing defines optimization and describes the evolutionary operators used in evolutionary algorithms. Genetic algorithm, genetic programming, and Cartesian genetic programming are described. The representation of the solution is explained for each of the algorithms. The subchapter on machine learning describes the types of learning: supervised, unsupervised, semi-supervised, and supported learning. Fundamental algorithms used in the dissertation are briefly described: Naive Bayes algorithm, support vector machine, multilayer perceptron, and convolutional neural network. The role of activation functions in neural networks is additionally explained. In the subchapter on cryptography, the properties of cryptographic algorithms with symmetric and asymmetric keys are specifically described. The last subchapter explains the concept of implementation attacks, the reason for the emergence of side-channel properties of the devices on which the cryptographic algorithm is performed, and the difference between profiled and non-profiled side-channel attacks.

The third chapter describes the construction of Boolean functions using the genetic algorithm and genetic programming to optimize the properties of high nonlinearity and balance. The chapter discusses Boolean functions of binary and quaternary bases. In both bases, the Walsh-Hadamard transformation of functions is used to find bent functions. In this chapter, we made an original scientific contribution to constructing Boolean functions with adaptive cryptographic properties. It is shown that although the search space for bent functions in the quaternary base is much larger, genetic programming successfully finds quaternary bent functions of input dimension n, which are transformed into Boolean functions of input dimension $2n$ with Gray mapping. In this way, we demonstrated the successful application of genetic programming in finding bent Boolean functions up to the input dimension 16.

The fourth chapter describes the construction of S-boxes with different cryptographic properties. The first problem to be solved is the construction of a *bent(n,m)* S-box where $m \leq \frac{n}{2}$,

that is, the output dimension is strictly smaller than the input dimension. The mentioned problem is solved using a genetic algorithm and genetic programming. In the genetic algorithm, different representations are experimented with: a series of bits and a series of floating-point numbers. Three different functions are optimized, one of which is a composition of derivatives of Boolean functions. The experiments were conducted up to an input dimension of 12. The research results confirm that evolutionary algorithms can successfully construct an S-box with different input dimensions, where genetic programming achieves the best results. We conclude that genetic programming achieves the most efficient results due to the presented solution. Furthermore, we investigate whether evolutionary computing algorithms can generate an S-box of dimensions $(n, n-2)$ and where the differential uniformity value equals 6. Due to the search space size, this problem proves to be very challenging. We achieve global optima for the dimensions $(4,2)$ and $(5,3)$, and, interestingly, different representations of the solution achieve the optima in different sizes. When solving this problem, we experiment with different solution representations: integer representation, permutation representation, quadruple permutation, and floating-point representation. The results indicate that the best solutions are achieved by genetic programming and a genetic algorithm with integer or floating-point encoding. Within this chapter, we made an original scientific contribution to constructing vectorial Boolean functions with adaptive cryptographic properties.

The fifth chapter explores the automatic construction of a symmetric key cryptographic algorithm. The problem is viewed as an optimization problem where one wants to construct a cryptographic algorithm with high nonlinearity, is resistant to linear cryptanalysis, respects the principle of diffusion, and is bijective. The algorithm uses bi-level optimization, where the sender's optimization function is outer, and the attacker's optimization function is an inner optimization function. Cartesian genetic programming is used in the optimization procedure due to interpretability, parallel construction of multiple solutions, and the inherently solved problem of excessive individual growth - bloat. Five scenarios are considered, considering the capabilities of the sender, receiver, and attacker. The attacker uses a known-plaintext attack. The experiments use artificially created text blocks of sizes 4 and 8 bits. The results show that the average success of the attacker in discovering the secret key is equal to random guessing, from which we conclude that evolutionary computing algorithms have the potential to be used in the automatic construction of a cryptographic algorithm. In this chapter, we realized the original scientific contribution of the automatic construction of cryptographic algorithms by using attacker and defense dynamics in the security domain.

In the sixth chapter, a profiled side-channel attack is considered. A scenario is explored in which the attacker is limited in the profiling phase with a small number of labeled traces, while in the attack phase, he can collect a larger number of traces. The chapter explores how semi-supervised learning can improve model performance. We use the Naive Bayes algorithm,

the support vector machine, and the attacker uses a template attack. The experiments were performed on the publicly available datasets DPAv2 and DPAv4 of the physical implementation of the AES algorithm. The results confirm the initial hypothesis, and traces from the attack phase can, through semi-supervised learning, make the model more efficient. In this chapter, we made an original scientific contribution to the field of machine learning algorithms in side-channel attacks.

The seventh chapter considers the possibility of using neural networks in profiled side-channel attacks: multilayer perceptron and convolutional neural network. Since the choice of the activation function significantly affects the efficiency of the neural network, the chapter uses genetic programming as an evolutionary optimization algorithm. The task of genetic programming is to create a proper activation function that reduces the loss function and improves the effectiveness of the attack. We researched a publicly available dataset of the ASCAD physical implementation of the AES algorithm. Experiments show that the evolution of activation functions achieves better results on a more straightforward dataset, which indicates the possibility of additional research efforts. In this chapter, an original scientific contribution was made using neuroevolutionary procedures for optimizing neural network architecture in the security domain.

Finally, in the eighth chapter, we provide conclusions where we again emphasize our contribution and the conclusions we made, and we list a number of possible future research directions.

1.3 Contribution

The main goal of the dissertation is to demonstrate the application of machine learning and evolutionary computing in the design and analysis of symmetric key cryptographic algorithms. The dissertation has achieved a scientific contribution consisting of four points:

- •Evolutionary algorithms for the construction of scalar and vectorial Boolean functions with adaptive cryptographic properties $\longrightarrow$ Chapter 3 and Chapter 4
- •Automatic construction of cryptographic algorithms by using attacker and defense dynamics in the security domain $\longrightarrow$ Chapter 5
- •Machine learning algorithms in side-channel attacks $\longrightarrow$ Chapter 6
- •Neuroevolutionary procedures for optimization of neural network architecture in the security domain $\longrightarrow$ Chapter 7

Chapter 2

Background

This chapter serves as a concise overview of the topics explored in this thesis. Given the multidisciplinary nature of our study, we will refrain from delving into details that are commonly known or readily accessible in the literature. Instead, we will focus on the techniques and questions we aim to address.

Our research endeavors to provide answers to several important questions. Firstly, can evolutionary computing techniques be used to construct cryptographic primitives? Secondly, is it feasible to automatically generate symmetric cryptographic algorithms that meet specific requirements? Lastly, how can machine learning techniques be employed to enhance implementation attacks, and how can evolutionary computing and machine learning be combined to achieve better success in these attacks? To facilitate comprehension of the remainder of this thesis, this chapter introduces the essential concepts relevant to the aforementioned questions.

The subsequent sections are arranged as follows: Chapter 2.1 offers a brief yet comprehensive overview of evolutionary computing, explicitly focusing on the algorithms employed in this study. Chapter 2.2 covers the fundamental principles of machine learning, emphasizing supervised and semi-supervised learning and various models used in the learning process. Chapter 2.3 introduces cryptography with a focus on symmetric cryptography. Finally, Chapter 2.4 delves into implementation attacks.

2.1 Evolutionary Computation

With the development of computing and the increase in the processing power of computers, people began to solve highly complex problems that were unsolvable until then, simply by using the technique of exhaustive search, that is, by searching the entire solution space. To speed up the search, a family of directed search algorithms was developed, which includes algorithms that are guided during the search by information about the problem they are solving and by estimating the distance from the current one to the desired target solution. This information is

taken into account to direct the search and end it sooner. Unfortunately, we cannot solve many problems in the previously described way.

To describe evolutionary computation, it is necessary to understand the taxonomy of algorithms. Algorithms are divided into deterministic and non-deterministic algorithms. Deterministic algorithms are those that give a specific output for specific input. On the other hand, non-deterministic algorithms are those algorithms that can provide an arbitrary output for a given input. One version of non-deterministic algorithms are probabilistic algorithms, which have a built-in stochastic process. Probabilistic algorithms are divided into Monte Carlo and Las Vegas algorithms. The difference between the above two types of algorithms is that Las Vegas algorithms always end up with the correct output, while Monte Carlo algorithms can give an incorrect output.

2.1.1 Optimization

Optimization refers to enhancing a system by reducing the demands on resources such as time, memory, and other system properties. The discipline of optimization can be classified into two main categories: single-objective optimization and multi-objective optimization problems. In the context of single-objective optimization, the objective function space is one-dimensional, so each solution can be mapped to a numerical value that denotes its quality. Conversely, in multi-objective optimization, the objective function space is multi-dimensional, and solutions are mapped into a vector of numerical values.

Optimization algorithms refer to those designed to solve optimization problems and are commonly categorized into two distinct types: exact and approximate algorithms. The former pertains to algorithms that can identify the optimal solution and guarantee its optimality, while the latter denotes algorithms that generate solutions of sufficient quality but cannot ensure optimality. The key trade-off between these two types lies in the execution time of the algorithm, with approximate algorithms typically demonstrating much faster execution times.

Heuristic algorithms are classified as approximate algorithms and are characterized by providing solutions that lack a provable quality. Typically, heuristics are employed as a constituent element of optimization algorithms, particularly in instances where data can be gathered to evaluate the current solution status or in the creation of the next solution.

Heuristics can be categorized into two main groups: specific heuristics and metaheuristics, with the former being tailored to solve particular problems. At the same time, the latter comprises algorithmic frameworks designed to generate heuristics that can be applied to a diverse range of problems. Metaheuristics are regarded as general-purpose heuristics whose function is to guide problem-specific heuristics toward the region in the solution space where viable solutions can be located.

Metaheuristics can be divided into two large classes of algorithms: single-solution-based

and population-based metaheuristics, which work with sets of solutions. Evolutionary computing algorithms belong to the latter family of algorithms.

When discussing optimization, it is necessary to define local and global optimums. In single-objective optimization, the global optimum on the entire domain can be a minimum or a maximum. A local optimum is an optimum on a subset of the domain, and it can be a local minimum or a local maximum. Local optima represent a severe problem in optimization algorithms.

2.1.2 Solution Representation and Operators

The initial step in solving any optimization problem is crucial as it involves determining the appropriate method for representing a specific solution to the computer. The efficiency and effectiveness of the optimization algorithm depend on this decision.

The solutions are subjected to a series of actions, including evaluation, modification, combination, generation of a neighborhood of solutions, and selection.

This series of operations performed on solutions is called evolution, and an individual represents one solution in this process. A genotype represents each individual, which serves as a means of presenting a solution. On the other hand, the phenotype is a behavioral expression of the genotype in a specific environment. Whether the display is genotypic or phenotypic depends on the problem being solved. Representation is a way of encoding a genotype, and an allele is a possible value that can appear at a particular place in the chromosome. A position in a chromosome is called a locus, and a chromosome represents a solution to a problem. A chromosome consists of building blocks called genes. An individual is a data structure corresponding to an element in the search space and consists of a chromosome and additional information, such as a fitness function.

The fitness function represents the definition of the problem that needs to be solved by evolutionary computation. It is a function whose value is associated with each possible solution. The fitness function is sometimes called the objective function. The set of individuals used in the optimization process is called a population. A generation represents individuals in one iteration of evolutionary computation. The process that ensures that the individual with the best fitness value does not disappear from the population is called elitism.

The solution representation can be categorized into various types: a string of bits, floating-point vectors or fields, permutations and matrices, trees, and other complex data structures. Figure 2.1 illustrates the floating-point encoding.

How the solution is displayed determines how we will implement the variation operators. Variation operators are the name for all recombination and mutation operators. Namely, recombination or crossover creates a new individual by combining information from two or more individuals. A mutation is an operator that makes a random change over an individual's geno-

0.27	0.31	0.15	0.77	0.70	0.89	0.62	0.43	0.47	0.03

Figure 2.1: Floating point representation of the solution.

type, ensuring that the solution still belongs to the domain of the problem [2]. The mutation value parameter P_m defines the mutation rate.

One can generate the initial population in several ways, of which the four most common strategies are:

- •random generation: pseudo-random (generation of independent values) or quasi-random (generation of independent but intentionally scattered values),
- •sequential diversification: a uniform sampling of the solution space; solutions are generated in sequences to optimize diversification,
- •parallel diversification: solutions are generated in a parallel independent way,
- •heuristic generation: solutions are generated using heuristics.

Examples of binary genotype variation operators are:

- •crossover: one-point crossover, N-point crossover, uniform crossover.
- •mutation: simple mutation, complete circulating mutation.

Examples of floating point genotype variation operators are:

- •crossover: discrete recombination, simple arithmetic recombination, single arithmetic recombination, whole arithmetic recombination.
- •mutation: simple mutation.

2.1.3 Selections

Evolutionary algorithms use a selection mechanism to select individuals that will participate in recombination. Selection is a process that allows individuals to be differentiated according to their fitness and it enables the transmission of better genetic material from generation to generation. The common property of all types of selection is a higher probability of selecting better individuals for reproduction. According to the method of transferring the genetic material of better individuals to the next iteration, selection procedures are divided into:

- •of generational selection: selection directly selects better individuals whose genetic material will be transferred to the next iteration i
- •elimination selection: bad individuals are selected for elimination, and better individuals survive the selection process.

The previously mentioned two selections determine the typical implementations of the evolutionary algorithm: the steady-state algorithm and the generational algorithm.

In the case of the steady-state algorithm, in each generation, two or more parents are selected

from the entire population on which the crossing is performed, resulting in the creation of new offspring. The offspring is then mutated and inserted into the population. Since the size of the population must be constant, insertion is performed so that the offspring replaces an individual from the population (for example, the worst). Algorithm 1 shows the steady-state algorithm.

Algorithm 1 Steady-state evolutionary algorithm.

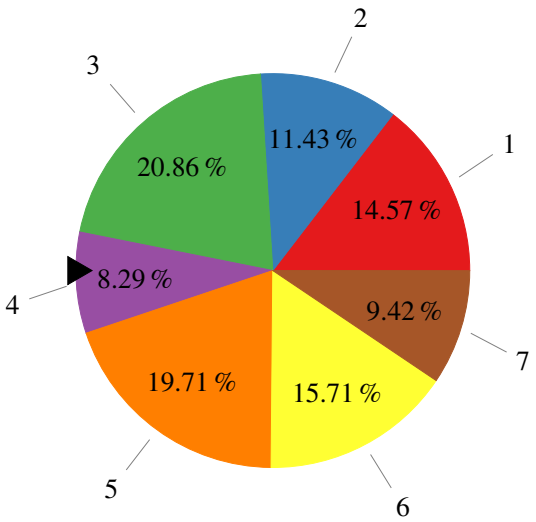
Input: Parameters of the algorithm
Output: Optimal solution set
Initialize population: $P \leftarrow CreateInitialPopulation()$
Evaluate population: $Evaluate(P)$
while $\neg TerminationCriterion$ **do**
  Select parents from population: $Q \leftarrow$ SelectMechanism(P)
  Apply variation operators on selected parents: $C \leftarrow$ VariationOperators(Q)
  Evaluate offspring: $Evaluate(C)$
  Insert offspring into new population: $P \leftarrow$ InsertIntoPopulation(C, P)
end while
return $OptimalSolutionSet(P)$

Unlike the previously described algorithm where there is no clear boundary between parents and offspring, the generational algorithm from generation to generation from the population of parents creates a population of offspring who then become parents — the old population of parents thus immediately dies out. Algorithm 2 shows the generational genetic algorithm.

Algorithm 2 Generational evolutionary algorithm.

Input: Parameters of the algorithm
Output: Optimal solution set
Initialize population: $P \leftarrow CreateInitialPopulation$
Evaluate population: $Evaluate(P)$
while $\neg TerminationCriterion$ **do**
  Create an empty population: $P' \leftarrow NewEmptyPopulation$
  while $Size(P') < Size(P)$ **do**
    Select parents from population: $Q \leftarrow$ SelectMechanism(P)
    Apply variation operators on selected parents: $C \leftarrow VariationOperators(Q)$
    Evaluate offspring: $Evaluate(C)$
    Insert offspring into new population: $InsertIntoPopulation(C, P')$
  end while
  Update population: $P \leftarrow P'$
end while

Selections differ according to the method of determining the value of the probability of selecting a particular individual. Proportional or roulette-wheel selection selects individuals with a chance proportionate to the fitness of the individual - the probability of selection depends on the ratio of the individual's fitness and the population's average fitness. Ranking selections select individuals with a chance that depends on the individual's position in the order of individuals sorted by fitness. Ranking selections are divided into sorting and tournament selections.



Index	Fitness
1	5.1
2	4.0
3	7.3
4	2.9
5	6.9
6	5.5
7	3.3

Figure 2.2: An example of roulette-wheel selection.

Sorting selections are linear selection, μ-λ, and truncated selection. Tournament selections are divided according to the number of individuals participating in the tournament. Figure 2.2 shows the example of roulette-wheel selection.

Pseudocode 3 shows steady-state tournament selection. k individuals are selected for recombination, and the $k-\mu$ worst individuals are replaced with offspring. Here, k denotes the tournament size, and μ is the number of surviving individuals. The thesis always considers the scenario with two parents.

Algorithm 3 Steady-state tournament selection.

Input: The population of candidate solutions P, the number of individuals k in the tournament, and the probability of mutation p_m
Output: Optimal solution set
for $i = 1$ to k **do**
  randomly select an individual x_i from P
end for
remove the individual with the worst fitness value from $x_1, ..., x_k$
randomly select two parents p_1, p_2 from $x_1, ..., x_k$
create an offspring o through crossover between p_1 and p_2
mutate the offspring o with probability p_m
return the parents p_1, p_2 and the offspring o

2.1.4 Evolutionary Algorithms

Evolutionary algorithms can be broadly categorized into four different approaches: genetic algorithms [3], genetic programming [4], evolutionary strategies [5], and evolutionary programming [6]. Evolutionary algorithms are based on the Darwinian theory of evolution[7].

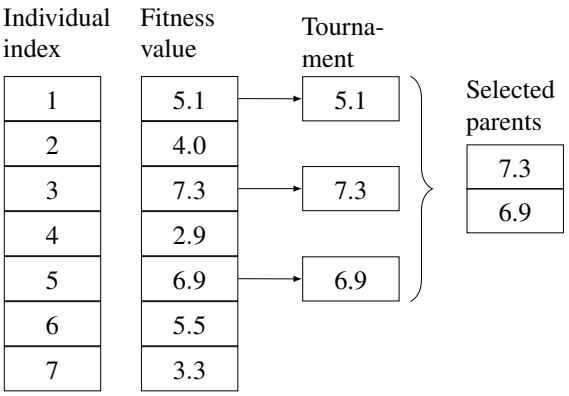


Figure 2.3: An example of tournament selection.

The optimization process involves a population of individuals that represent possible solutions. The population evolves towards a better solution through the application of evolutionary operators. The fitness function, which is always dependent on the specific problem, is used to evaluate the quality of a solution. Initially, the evolution begins from a pool of randomly selected individuals, and utilizing evolutionary operators generates a new and better population.

The main difference between the various evolutionary algorithms lies in the method of individual encoding and the implementation of evolutionary operators. The algorithms used in this thesis are briefly explained in the following.

Genetic Algorithm

In 1975, H. J. Holland presented the genetic algorithm[8], which drew inspiration from Darwin's theory of the origin of species. The genetic algorithm is a type of evolutionary algorithm that typically employs a binary string, a floating-point number array, or a permutation vector to represent the elements of the search space.

The genetic algorithm commonly utilizes a recombination operator and a mutation operator. The recombination operator facilitates a local search, while the mutation operator promotes diversity within the population. However, it is essential to note that a mutation rate that is too high can lead to a random search. The implementation of the variation operator depends on the genotype representation.

When discussing a genetic algorithm, it is imperative to specify the parameters utilized in the algorithm, such as the selection operator's type and parameters, population size, mutation type and parameters, recombination type, and parameters, termination criterion, and the number of trials.

In this thesis, the genetic algorithm is employed to optimize the cryptographic properties of Boolean functions and vectorial Boolean functions. The author has explored the application of genetic algorithms to a variety of problems in the cryptography field in [9].

Genetic Programming

Genetic programming (GP) is an optimization process for automatically generating computer programs to solve complex problems encountered in computing and everyday life. The concept is derived from general ideas from the theory of genetic algorithms and other evolutionary methods. The goal of GP is to create a universal computer program that can find solutions to problems described only by input data and desired results.

In 1992, J.R. Koza introduced GP [11]. GP is similar to genetic algorithms, but instead of a chromosome representing a candidate solution, it means a program that solves a given problem. A tree is a typical data structure used in genetic programming to describe chromosomes. The tree consists of functional nodes (inner nodes) and terminals (leaves or variables) collectively called primitives. The tree representation in memory can be realized in several ways, resulting in different versions of genetic programming.

When we talk about the tree, the syntax tree is often used in genetic programming. Syntax trees are built according to context-free grammar, have a root, and are ordered. The depth of the tree is an additional parameter of the genetic algorithm. The tree depth limits the excessive growth of individuals, also called bloat.

In this thesis, genetic programming is used to optimize the cryptographic properties of Boolean and vectorial Boolean functions.

Cartesian Genetic Programming

J.F. Miller presented in 1990 a new version of genetic programming that uses a directed graph instead of a tree representation[12]. Since then, various versions of Cartesian Genetic Programming (CGP) have been developed, and certain advantages over genetic programming have been shown.

In CGP, an individual's genotype is represented by a directed graph, and the phenotype is the program. The terminal set (inputs) and node outputs are numbered sequentially. Node functions are also numbered separately. CGP has three parameters to be chosen by the user; number of rows n_r, number of columns n_c, and levels-back l [13].

Genotype is of constant size and consists of integer values. Integer values uniquely determine the node, the function that the node performs, the inputs to the node, and the output nodes. A node that contains a function is called a functional node. In addition to the graph, the picture 2.4 also shows an example of a genotype. When decoding the genotype, some nodes may be excluded from the solution. Unconnected nodes are those that are not on the path from the output to the input of the graph. Nodes that affect the phenotype change are called active nodes.[13].

After decoding, the phenotype can consist of all the nodes defined by the size of the genotype or even none. The phenotype will not have any functional node if the inputs are directly

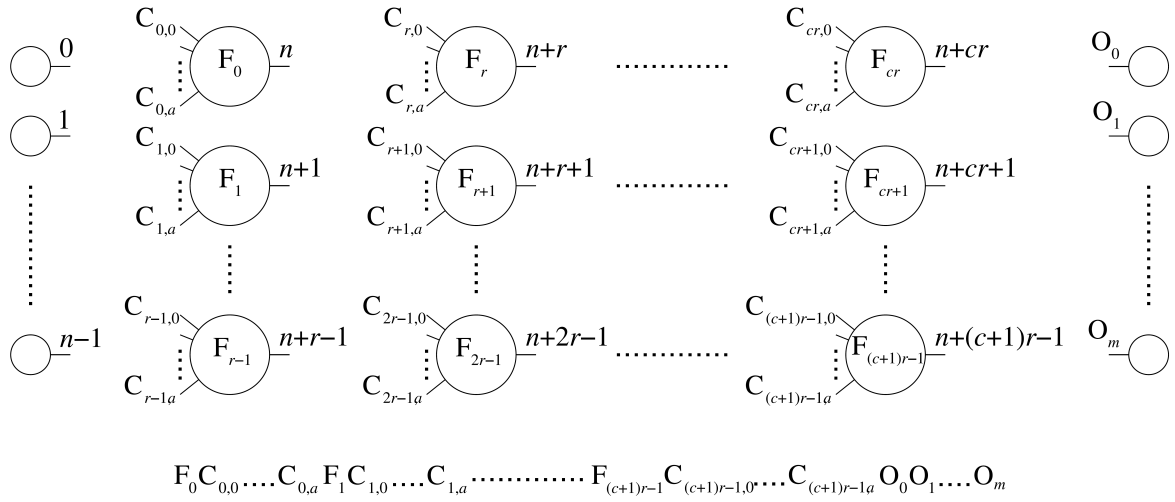


Figure 2.4: Phenotype and genotype representation of CGP.

connected to the outputs, and this is called a shortcut connection.[14].

The length of the genotype is determined by the Equation 2.1, where n_n is number of node input connections and n_o number of graph output connections.

One of the significant advantages of genetic programming is limited genotype length because there is no fear of noticeable and sudden growth of the genotype during evolution, called bloat. CGP has shown the most success in learning logical functions because it provides a large amount of local search, unlike genetic programming [12], but it has various applications. Authors in [15] use CGP as a classifier, and to enhance the model capacity, they introduce an amplitude.

$$genotype\_length = n_r \cdot n_c \cdot (n_n + l) + n_o \tag{2.1}$$

CGP as an optimization procedure usually uses the evolutionary strategy $(\mu + \lambda)$, where $\mu = 1$ and $\lambda = 4$. In [12], it was shown that the crossover operator does not contribute to the improvement of the individual, so it is not used. The only operator for creating new individuals is mutation. There are several ways of implementing the mutation operator, the most common of which are: probabilistic mutation, probabilistic mutation of active nodes and single mutation [13]. The advantage of point mutation of an active node over other implementations is that no mutation probability parameter is required. In [16], it is stated that this type of mutation achieves the best results.

2.2 Machine Learning

Machine learning is a subfield of computer science that aims to develop algorithms capable of effectively learning from a collection of examples of some phenomenon. The samples can

be obtained from various sources, such as nature, human-crafted data, or generated through an algorithmic process.

More formally, machine learning can be characterized as gathering a dataset and developing a statistical model using algorithms specifically designed to identify patterns and relationships within the data. The statistical model is then used to solve practical problems in some way, such as making predictions, classification, or optimization.

2.2.1 Types of Learning

Learning can be supervised, semi-supervised, unsupervised and reinforcement.

Supervised Learning

In supervised learning, the dataset is the collection of labeled examples $\{(\vec{x}_i, y_i)\}_{i=1}^{N}$. Each element $\vec{x}_i$ among N is called a feature vector. A feature vector is a vector in which each dimension $j = 1, ..., n$ contains a value that describes the example somehow. That value is called feature and is denoted as $\vec{x}^{(j)}$. The label y_i can be either an element belonging to a finite set of classes $1, 2, ..., C$, a real number, or a more complex structure, like a vector, a matrix, a tree, or a graph[17].

The goal of a supervised learning algorithm is to use the dataset to produce a model that takes a feature vector as input and outputs information that allows deducing the label for this feature vector.

Unsupervised Learning

In the context of machine learning, unsupervised learning pertains to the task of modeling a dataset consisting of unlabeled examples $\{\vec{x}_i\}_{i=1}^{N}$, where $\vec{x}$ denotes the feature vector. The objective of an unsupervised learning algorithm is to generate a model that takes a feature vector **x** as input, and either transforms it into another vector or a value that can be leveraged to solve a practical problem.

For instance, in clustering, the model identifies the cluster identifier corresponding to each feature vector in the dataset, while in dimensionality reduction, the model's output is a feature vector containing fewer features than the input $\vec{x}$[17].

Semi-Supervised Learning

Semi-supervised learning involves a dataset comprising labeled and unlabeled examples, where the quantity of unlabeled examples is typically much larger than that of labeled ones. Despite this imbalance, the goal of a semi-supervised learning algorithm remains the same as that of

a supervised learning algorithm: to produce a model capable of deducing the label for a given feature vector.

At first, incorporating more unlabeled examples into the learning process might seem counterintuitive, as it introduces more uncertainty to the problem. However, adding unlabeled examples can provide additional information about the problem, as a larger sample size better reflects the underlying probability distribution of the labeled data. Theoretically, a learning algorithm can exploit this additional information to improve its model [17].

Reinforcement Learning

Reinforcement learning is a subfield of machine learning that involves an agent operating within an environment and receiving feedback in the form of rewards based on the actions it takes. The agent perceives the state of the environment as a vector of features and can execute actions in response to this state. Each action has an associated reward and may also result in a transition to a new state. The objective of a reinforcement learning algorithm is to learn a policy that maps each state to an optimal action, where optimality is defined as maximizing the expected cumulative reward over a long-term horizon[17].

The policy function takes the state feature vector as input and outputs the recommended action. This paradigm is well-suited to sequential decision-making problems with long-term goals.

2.2.2 Fundamental Algorithms

Classification is a supervised machine learning technique that seeks to assign an example to the class to which it belongs. If the value associated with the example is nominal, the task is referred to as classification, whereas if the value is continuous, it is referred to as regression.

An example is a feature vector, $\vec{x} = (x_1, x_2, ..., x_n)^T$, where n is the dimensionality of the space in which the example is located. The space in which the example is located is called the input space or the example space. Let X be the set of all possible examples. All machine learning algorithms assume that the samples from X are sampled independently and from the same common distribution $P(\vec{x}, y)$. In supervised learning, the class label to which the example $\vec{x}$ from the learning set belongs is known in advance. If it is true that $y \in \{0,1\}$ then it is a binary classification, and if $y \in \{0, ..., n\}$ then it is it multi-class classification. The set of learning examples D consists of pairs of examples and associated labels.

The classification algorithm seeks to determine the hypothesis $h : X \to \{0,1\}$ which predicts whether the example $\vec{x}$ belongs to the class C or not. The set of all possible hypotheses H is called a model or hypothesis space.

The main components of the supervised learning algorithm are: the model, the loss function,

and the optimization procedure. The loss function for model parameters θ calculates the loss difference of the target value $y^{(i)}$ and its approximation $h\left(\vec{x}^{(i)}|\theta\right)$. The optimization procedure finds the values of θ^* for which the empirical error is the smallest, as shown by the Equation 2.2.

$$\theta^* = \underset{\theta}{argmin}\, E\left(\theta|D\right) \tag{2.2}$$

Here, we describe the fundamental machine learning algorithms used in this thesis.

Naive Bayes

The Naive Bayes (NB) classifier is representative of generative models [18]. Classification of examples is achieved using the Bayes rule, which for each class gives the probability that the sample belongs to that class. The NB is a parametric model, assuming that the examples obey some theoretical probability distribution. The parameters of this distribution are unknown and need to be learned based on learning examples.

The NB classifies the example $\vec{x}$ based on the calculated posterior probability $P\left(Y = C_j|X = x\right)$. This probability is calculated indirectly based on the joint density $p\left(\vec{x},C_j\right)$ of the exemplary Bayes rule described by the Equation 2.3.

$$P\left(C_j|\vec{x}\right) = \frac{p\left(\vec{x}|C_j\right)P\left(C_j\right)}{\sum_{k=1}^{K} p\left(\vec{x}|C_k\right)P\left(C_k\right)} \tag{2.3}$$

The marginal probability $P\left(C_j\right)$ is called the a priori probability of the class, and the conditional density $p\left(\vec{x}|C_j\right)$ is called the class conditional density or class likelihood [18]. The optimal classification decision is the one that maximizes the posterior probability $p\left(C_j|\vec{x}\right)$. Such a hypothesis is called the maximum a posteriori hypothesis and is described by the Equation 2.4.

$$h\left(\vec{x}\right) = \underset{C_k}{arg\,max}\; p\left(\vec{x}|C_k\right)P\left(C_k\right) \tag{2.4}$$

If the examples are discrete variables, then the corresponding values are used instead of the class's conditional density and a priori probability. If the samples are continuous variables, the class probabilities are usually modeled by a Gaussian normal distribution. The main reason is analytical simplicity. It is noted here that normal variables are not always normally distributed and that there are statistical tests that can be used to determine whether the attributes of an example obey a normal distribution.

The likelihood of a class for a multidimensional continuous random variable is modeled by the multivariate Gaussian distribution shown by the Equation 2.5.

$$p\left(\vec{x}|C_j\right) = \frac{1}{(2\pi)^{\frac{n}{2}}\left|\Sigma_j\right|^{\frac{1}{2}}} e^{-\frac{1}{2}\left(\vec{x}-\vec{\mu}_j\right)^T \Sigma_j{}^{-1}\left(\vec{x}-\vec{\mu}_j\right)} \tag{2.5}$$

The vector μ_j describes the prototypical value of the example in the class C_j, while the covariance matrix Σ_j describes the amount of noise in each variable and the correlation between noise sources. This distribution generates a model defined by the formula 2.6.

$$h_j\left(\vec{x}\right) = -\frac{n}{2} ln\ 2\pi - \frac{1}{2} ln\ \left|\Sigma_j\right| - \frac{1}{2}\left(\vec{x}-\vec{\mu}_j\right)^T \Sigma_j{}^{-1}\left(\vec{x}-\vec{\mu}_j\right) + ln\ P\left(C_j\right) \tag{2.6}$$

Support Vector Machines

Support vector machines (SVM) is a discriminative model of machine learning. Unlike the Bayesian classifier, this model does not have any probabilistic interpretation. SVM solves the problem of an arbitrary hypothesis that separates examples of two classes by introducing a maximum margin criterion. The name support vector machine comes from the hyperplane that separates examples of different classes and can be represented as a combination of selected sample vectors from the training set. Additional efficiency of SVM is achieved by applying kernel functions, which classify this model into a group called kernel machines.

A linear model for binary classification is defined by the formula 2.7

$$h\left(\vec{x}\right) = sgn\left(\vec{w}^T\vec{x} + w_0\right) \tag{2.7}$$

where $sgn\left(\cdot\right)$ denotes the signum function. If the examples are linearly separable, the constraints 2.8 and 2.9 apply.

$$y^i\left(\vec{w}^T\vec{x}^i + w_0\right) \geqslant 1, \tag{2.8}$$

$$d_{margin} = \frac{2}{\|\vec{w}\|} \tag{2.9}$$

where 2.9 represents the maximum margin width value. To solve the optimization with constraints, the optimization using the method of Lagrange multipliers is used. The Equation 2.10 describes the primary problem whose solution represents a minimum and is located in the saddle of the function L.

$$L\left(\vec{w}, w_0, \alpha\right) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{N} \alpha_i\{y^i\left(\vec{w}^T\vec{x}^i + w_0\right) - 1\} \tag{2.10}$$

For a more straightforward solution and reduced computational complexity, the primary problem is transformed into a dual problem whose goal is to maximize the expression 2.11.

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^i y^j \left(\vec{x}^i\right)^T \vec{x}^j \tag{2.11}$$

After the model is trained, new examples are classified according to the Equation 2.7.

In the previous discussion, the assumption was that the examples are linearly separable. However, in practice, such cases rarely occur. The solution to this problem is introducing a soft margin that allows some examples to be misclassified. With a soft margin, the wrong classification is allowed, but in the training process, the classifier will be penalized more the more examples are wrongly classified. Penalization is achieved by introducing reserve variables ξ. $\xi = 0$ is valid for examples $\vec{x}$ that are on the correct side of the classifier, while $\xi_i = \left|y^i - h\left(\vec{x}^i\right)\right|$ for misclassified examples. Examples inside the margin but on the correct side of the boundary, or lie right on the decision boundary, will be penalized by $0 < \xi_i \leqslant 1$. Examples on the wrong side of the decision boundary will be penalized by $\xi \geqslant 1$.

Soft margin limits can be defined by the formulas 2.12 and 2.13.

$$y^i \left(\vec{w}^T \vec{x}^i + w_0\right) \geqslant 1 - \xi_i, i \in \{1, ..., N\}, \tag{2.12}$$

$$fitness = \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^{N} \xi_i \tag{2.13}$$

The parameter C determines the trade-off between the size of the margin and the total penalty. A larger C leads to a more significant misclassification penalty, a consequence of more complex models.

SVM can also become a non-linear model[19]. The idea is based on Cover's theorem. If the linear models are good enough for $n \gg N$, then the examples can be mapped to a higher-dimensional space where they are more likely to be linearly separable. So, instead of transforming the model, SVM transforms the problem. In the Equation 2.11 the example $\vec{x}$ appears in the form of a scalar product. This allows the scalar product to be replaced by the Equation 2.14

$$\kappa\left(\vec{x}, \vec{x}^*\right) = \phi\left(\vec{x}\right)^T \phi\left(\vec{x}\right)^* \tag{2.14}$$

where κ represents the kernel function and ϕ is the attribute mapping function. There are a number of standard kernel functions, some basic ones are 2.15, 2.16 and 2.17.

$$\kappa_{linear}\left(\vec{x}, \vec{x}^*\right) = \vec{x}^T \vec{x}^*, \tag{2.15}$$

$$\kappa_{polinomial}\left(\vec{x}, \vec{x}^*\right) = \left(\vec{x}^T \vec{x}^* + 1\right)^p, \tag{2.16}$$

$$\kappa_{radial}\left(\vec{x}, \vec{x}^*\right) = \kappa\left(\|\vec{x}^T \vec{x}^*\|\right). \tag{2.17}$$

The Equation 2.15 represents a linear kernel, 2.16 a polynomial kernel, and 2.17 a homogeneous kernel or radial basis function.

The SMO algorithm (sequential minimal optimization) is used to implement SVM [20, 21]. SMO solves the quadratic programming optimization problem. Since SMO is a binary classification algorithm, $\frac{n\times(n-1)}{2}$ binary SVM classifiers are used for multiclass classification.

2.2.3 Artificial Neural Network

Artificial neural networks (ANNs) are a class of computational systems inspired by biological neural networks. ANNs are widely used in machine learning as they can learn from examples. In an ANN, artificial neurons are arranged in multiple layers, interconnected to transmit signals.

A fundamental type of ANN is known as a perceptron. A perceptron is a linear binary classifier that operates on a feature vector to determine whether it belongs to a particular categorical class. The perceptron associates a weight w_i with each input vector component and has a threshold value q. The output of a perceptron is equal to 1 if the weighted sum of the input vector is greater than the threshold value and -1 otherwise. However, perceptron classifiers can only handle linearly separable data, meaning they can only classify data when there is a hyperplane that can separate all the positive points from all the negative points [22].

Multilayer Perceptron

We obtain a multilayer perceptron algorithm by integrating more layers into a perceptron. Multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. Unlike linear perceptron, MLP can distinguish data that are not linearly separable. MLP consists of multiple layers of nodes in a directed graph, where each layer is connected to the next. Consequently, each node in one layer connects with a specific weight w to every node in the following layer. The multilayer perceptron algorithm consists of at least three layers: one input layer, one output layer, and one hidden layer. Those layers must consist of nonlinearly activating nodes [23]. The backpropagation algorithm is utilized for training the network, which is a generalization of the least mean squares algorithm in the linear perceptron. The gradient descent optimization algorithm uses backpropagation to adjust the weight of neurons by calculating the gradient of the loss function [22].

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent a type of neural network first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [24, 25]. From the operational perspective, CNNs are similar to ordinary neural networks (e.g., multilayer perceptron). More precisely, they consist of several layers, and each layer

is composed of neurons. CNNs use three main layers: convolutional layers, pooling layers, and fully-connected layers. Convolutional layers are linear layers that share weights across space. Pooling layers are nonlinear layers that minimize the spatial size to inhibit the number of neurons. Fully-connected layers are layers where every neuron is connected with all the neurons in the neighborhood layer (as in the MLP). For additional information about CNNs, we refer interested readers to [26].

Activation Functions

An activation function of a node is a function g defining the output of a node given an input or set of inputs from a layer of linear nodes, as denoted in Eq. (2.18). To enable replications of non-trivial functions with ANNs using a small number of nodes, one requires nonlinear activation functions:

$$y = activation\left(\sum_{i=1}^{|inputs|}(weight_i \cdot input_i) + bias\right). \tag{2.18}$$

Changes to the bias value allow the activation function to be shifted across the input domain, while changes to the weights alter the activation function's steepness. Combined with the backpropagation algorithm, this enables the ANN to model the data automatically.

There are three activation functions: binary step function, linear activation function, and nonlinear activation function. The problem with the first one is that it does not allow multi-value outputs, which prohibits multi-classification. A linear activation function suffers from two significant issues: it cannot use gradient descent to train the model because the function's derivative is a constant, and all layers of the neural network collapse into one. Note, a linear combination of linear functions is still a linear function, and such an activation function transforms the neural network into a single layer [26].

As a result, modern neural network models use nonlinear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, essential for learning and modeling complex data or data with high dimensionality. Nonlinear functions address the problems of linear activation functions. First, they allow backpropagation learning because they have a derivative function related to the inputs. Moreover, they will enable the stacking of multiple layers of neurons to create a deep neural network.

There is no clear rule for selecting an activation function. In practice, activation functions are selected by empirical results and speed of execution. Although some functions have theoretically substantiated properties, that is not the best approach. Typically, *ReLU* is often used, and for the hidden layers of recursive models, *tanh* is commonly selected, e.g., [27].

2.3 Cryptography

Cryptography is the study of mathematical techniques for all aspects of information security. Moreover, cryptography can be defined as a science of secret writing to hide the meaning of a message [28]. One uses cryptographic algorithms, commonly known as ciphers to ensure that goal. The security of information encompasses the fundamental aspects: confidentiality, data integrity, authentication, and non-repudiation. To ensure all the previous aspects, cryptographic algorithms and modes should satisfy many criteria, which can be regarded as a multi-objective combinatorial optimization problem[29].

Modern cryptography designs cryptographic algorithms that are assumed to be hard to break by an adversary. It is divided into symmetric key cryptography, asymmetric key cryptography, and hash functions [30] .

2.3.1 Asymmetric Key Cryptography

This type cryptography is also known as public key cryptography. This approach employs two distinct keys: one for encryption and another for decryption. The encryption key is public, while the decryption key is private. Public key cryptography can be classified into three categories:

•cryptosystems founded on the factorization problem,

•cryptosystems founded on the discrete logarithm problem,

•other cryptosystems.

Among these cryptosystems, the RSA algorithm is particularly well-known and derives its name from its inventors [31].Its security is rooted in the difficulty of the integer factorization problem. During encryption, the process of modular exponentiation is utilized. Assuming p and q are prime numbers, then $n = pq$, where $p, q \in \mathbb{Z}$, $de \equiv 1 \bmod \varphi(n)$, and $\varphi(n)$ is the Euler function. The public key is (n, e), and the private key is (p, q, d). The RSA algorithm is believed to be secured as long as large p and q are used. As of 2021, the National Institute of Standards and Technology (NIST) recommends key sizes of at least 3 072 bits for RSA encryption to provide a reasonable level of security for most applications[32].

Other famous cryptosystems or protocols founded on the discrete logarithm problem include Diffie-Hellman[33] or ElGamal [34]. Another large family cryptosystems are based on elliptic curve discrete logarithm problems (ECDLP). One representative of that family is Menzes-Vanstone cryptsystem [35]. Other public key cryptosystems are Merkle-Hellman[36], McEliece[37] and NTRU[38].

2.3.2 Symmetric Key Cryptography

Symmetric key cryptography, also known as secret key cryptography, uses the same key for both encryption and decryption, necessitating a mechanism to maintain the secrecy of the key [39]. This cryptography is called symmetric since communication parties share the same secret key. The primary advantage of symmetric key cryptography over asymmetric key cryptography is its significantly faster processing. Symmetric key cryptography can be categorized into block ciphers and stream ciphers.

Block ciphers accept a plaintext block and a key as input, producing a ciphertext block the same size as the plaintext. On the other hand, stream ciphers generate an arbitrarily long stream of key material, which is combined with the plaintext bit by bit. The output of a stream cipher is generated using a hidden internal state that changes as the cipher progresses [40].

The design of block ciphers rests on two fundamental principles: confusion and diffusion. Confusion means that each digit of ciphertext should depend on several parts of the key. Diffusion means that if a single bit of a plaintext is changed, then, statistically, half of the bits in the ciphertext should change and vice versa.

An S-box or a substitution box is a basic component of symmetric key algorithms. The main purpose of using an S-box is to introduce a confusion. The nonlinearity property of S-boxes is one of the most important cryptographic criteria because cryptographic algorithm should be resistant to linear cryptanalysis [41]. S-box (n,m) consists of n input variables and m output variables. These m outputs can be viewed like m Boolean functions. If nonlinearity of a Boolean function is equal to maximum then it is called bent function. The nonlinearity bound is called the covering radius bound and is strict for bent Boolean functions.

Stream ciphers usually work by producing a keystream that is added modulo two (XOR) with plaintext bits. To obtain such a keystream, one well researched way is to employ linear feedback shift register (LFSR). However, the output from an LFSR is linear and there exist easy cryptanalysis techniques against it [42]. To add nonlinearity to the cipher, and consequently make the cryptanalysis more difficult, one can for instance add one or more Boolean functions. Two well explored approaches are to use combiner or filter generators. In a combiner generator, outputs from several LFSRs serve as an input to a Boolean function. In a filter generator, the output is obtained by a nonlinear combination of a number of positions in one longer LFSR [43].

Other criteria which is considered in Boolean or vectorial functions are algebraic degree, balancedness, resiliency, algebraic immunity, and others [41]. If all criteria are used simultaneously then the problem of finding the best Boolean function or an S-box is a multi-objective problem. To inform more about definitions or mathematics background see [43, 44].

In the context of cryptography, communication between two parties, Alice and Bob, often involves transmitting sensitive information, which must be protected from unauthorized access. To achieve this, Alice encrypts the plaintext P using a cryptographic algorithm, or cipher, to

produce the ciphertext C. The process of encryption involves applying a transformation E to the plaintext P, such that only authorized parties can decipher the message. Encryption can be expressed as $E(P) = C$.

Once Bob receives the ciphertext, he must be able to decipher it to obtain the original plaintext. This process, called decryption, involves applying a transformation D to the ciphertext C such that $D(C) = P$. The transformation D is typically the inverse of the transformation E so that the original plaintext can be recovered. It is important to note that only parties authorized to access the secret key used in the encryption process can successfully decrypt the ciphertext.

A cryptographic primitive is a part of a cryptographic tool used to provide information security, i.e., a low-level cryptographic algorithm that is frequently used to build cryptographic protocols. A cryptographic algorithm (cipher) is a mathematical function used for encryption, decryption (ciphers are also used for other actions but those are outside the scope of this thesis). From the attacker side, there are several models one can consider on the basis of his access to the system under attack. For instance, the attacker can have access only to the ciphertext (commonly known as Ciphertext-only attack). However, we can consider a more powerful attacker that has access to both ciphertexts and accompanying plaintexts. Such a model is called the Known Plaintext Attack (KPA) model and is the model we use in this paper. Note, there are more powerful models than KPA but we consider them less relevant here since such models use specific attacks that Eve does not know. Additionally, we emphasize that very powerful cryptanalysis technique called Linear cryptanalysis is actually KPA.

Assuming Alice and Bob intend to communicate confidentially while utilizing an insecure channel, they may choose to encrypt their messages to preclude unauthorized reading by third parties. After encrypting her message, Alice may send it over the insecure channel to Bob. Bob, provided that he shares the same key with Alice, can then decipher and access the contents of the message, while any attempts by Eve to decode the message would prove unsuccessful in the absence of the key. To ensure the confidentiality of their communication, Alice and Bob must keep either the key or the algorithm used for encryption confidential. A. Kerchoff, as early as the 19th century, stated that a cryptosystem should remain secure even if all details of the system, except for the key, are disclosed to the public. [45].

For a computationally secure cryptosystem, C. Shannon deduced it should follow the confusion and diffusion principles [46]. The confusion principle means that the cipher output statistics should depend on the cipher input statistics in a manner too complicated to be exploited by the attacker. The confusion principle is related to the notion of nonlinearity since the attacker cannot easily approximate a cipher with a set of linear equations if the cipher possesses enough nonlinearity. More precisely, if a system is linear (i.e., S is a linear transformation) then $S(a)+S(b) = S(a+b)$ while if S is a nonlinear transformation then $S(a)+S(b) \neq S(a+b)$ and in general, even if we know the result of $S(a)$ and $S(b)$, we do not know the result of $S(a+b)$.

To measure the nonlinearity of a function, we need to measure its distance (e.g., Hamming distance) to all linear and affine functions [44]. The diffusion principle relates to the fact that each digit of the input and each digit of the secret key should influence many digits of the output. This principle can be modeled through a general concept of avalanche criterion: a single bit change at the input must change at least half of the bits of the output (in the case exactly half of the bits must change then we talk about strict avalanche criterion [47]). Symmetric key cryptography can be divided into block and stream ciphers. The main differences between those two types is that given a message M and a ciphertext C when working with block ciphers it is hard to reconstruct the encryption transformation. When working with stream ciphers, the encryption transformation is easy and the security relies on the changing of that transformation for every symbol. One-time pad (OTP) is the only cryptographic system that ensures the perfect secrecy, i.e., that no attacker can break it, provided that some rules are enforced: the keys need to be 1) at least the same size as the plaintext, 2) random, 3) kept in secrecy, and 4) never reused.

2.4 Implementation Attacks

Implementation attacks refer to attacks that exploit vulnerabilities in implementing a cryptographic system on a device. If usability, price, or efficiency parameters are considered, implementation attacks are one of the most dominant attacks on cryptographic devices [48]. Implementation attacks are divided into active and passive. Passive attacks involve a situation where the cryptographic device operates following its specifications. In such attacks, the secret key is revealed by analyzing the device's physical characteristics, such as the time it takes to execute cryptographic operations or energy consumption. Active attacks, on the other hand, involve physical modifications to the device to enable it to function outside the boundaries specified in the device's behavior. As a result, the secret key is revealed through the device's undefined behavior in active attacks.

In addition to the division into active and passive attacks, implementation attacks can also be divided according to the level of invasiveness. In a non-invasive attack, there is no mechanical action on the cryptographic device. In a semi-invasive attack, the attacker removes only the external parts of the cryptographic device, while in an invasive attack, the attacker changes the circuitry of the cryptographic device [48].

The most common types of implementation attacks are side-channel attacks (SCA), fault attacks (FA), and probing attacks [49].

SCA belongs to passive or non-invasive attacks. In such attacks, to discover the secret key stored on the device, knowledge about the consumption of electricity (power analysis attack, PAA), electromagnetic radiation, runtime, or sound produced by the device is exploited. A signal that describes the power consumption, which the attacker later uses to discover the secret

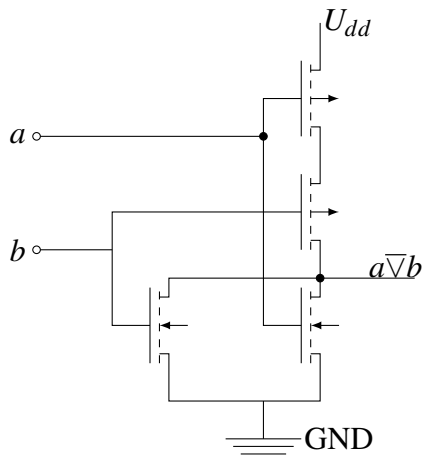


Figure 2.5: NOR logic gates implemented using CMOS technology.

key, is called a trace.

The basic idea of PAA is to take advantage of the fact that the current power consumption of a cryptographic device depends on the data that the cryptographic algorithm is currently using, as well as on the cryptographic operations that are performed. PAA attacks are divided into direct attacks and two-level attacks. Direct PAA attacks are simple power analysis (SPA), differential power analysis (DPA), correlation power analysis (CPA), and collision attacks (CA). Two-way PAA attacks are template attacks, stochastic models, and linear regression analysis (LRA)[50].

2.4.1 Electronic circuits

One of the predominant sources of leakage in cryptographic devices is attributed to their electronic circuitry. Electronic circuits, an essential component of almost all cryptographic devices, necessitate an understanding of their contribution towards the leakage of sensitive information for designing and evaluating secure cryptographic systems.

Complementary metal-oxide-semiconductor (CMOS) technology is widely used for electronic circuitry fabrication. It implements the basic logic gates that serve as the building blocks for more complex components like bistables, registers, and memories. The complementary nature of CMOS technology lies in implementing the primitive function, as demonstrated in the NOR logic gate shown in Figure 2.5. The pull-up network (PUN) implements the NOR function, while the pulldown network (PDN) implements the OR function's complement. However, during a transition of the network output from 0 to 1 or vice versa, the transistors PUN and PDN, due to delay, remain permeable to the voltage U for a brief period, leading to a short circuit. The high consumption of electricity manifests as a short circuit.

From the above, it is reasonable to conclude that the amount of energy dissipated by an electronic circuit is proportional to the number of state changes that occur. A simple mathematical

model that describes this relationship can be expressed using Equation 2.19.

$$P(t) = \sum_{g} f(g,t) + N(t) \tag{2.19}$$

where $f(g,t)$ represents the power consumption of gate g at time t, and $N(t)$ represents white noise. The noise originates from the power source or clock generator.

2.4.2 Profiled and Non-profiled attacks

A non-profiled attack refers to an attack scenario in which the attacker does not possess a copy of the cryptographic device beforehand to construct a secret key discovery model. In contrast, power consumption analysis can be employed in profiled attacks through trace classification, where the trace can describe the Hamming weight of the algorithm output or its value. In case the attacker has access to a copy of the cryptographic device, they can learn a model that differentiates traces based on their power consumption. Subsequently, this trained model can be applied to the original cryptographic device.

In the realm of side-channel analysis (SCA), two well-established models for power leakage are the Hamming weight (HW) and the Hamming distance (HD). HW assumes that the power consumption is correlated with the HW data within the cryptographic algorithm. In contrast, the Hamming distance model assumes that the power consumption is correlated with the state change on the bus.

SCA distinguishers are statistical models that make the connection between the traces and the secret key. Attacks are divided into two groups: those that assume a tight relationship between the trace and the secret key and those that infer based on a statistical model that considers the traces. Distinguishers are used in the second group of attacks. With DPA, a difference of means (DOM), T-test, variance test, Pearson's correlation, and Spearman's Rank Correlation [51] are often used.

Many traces are required to perform the differential power analysis procedure, typically on the order of 10^3. The analysis involves constructing two matrices: the trace matrix and the hypothesis matrix. Both matrices have the same number of rows, equal to the number of traces collected. The number of columns in the trace matrix is determined by the number of samples taken during the measurements. In contrast, the number of columns in the hypothesis matrix equals the number of possible values of one byte of the key, which is typically within the range of $[0,255]$ for AES and $[0,63]$ for DES.

The selection function $D(H,b)$ is used to determine the value of the target bit b of one byte of the encrypted text H using one byte of the hypothetical key Ψ. The traces are then partitioned into two groups based on whether $D(H,b) = 1$ or $D(H,b) = 0$. The group containing traces for which $D(H,b) = 1$ is denoted as G_1, while the other group is denoted as G_2. If the

hypothetical value of one byte Ψ of secret key K is correct, then the average value of traces from G_1 will be spiked above the average value of traces from G_2. On the other hand, if the value of Ψ is incorrect, then the function $D(H,b)$ will classify the traces in G_1 or G_2 with equal probability ($p = \frac{1}{2}$). Therefore, the average traces will be the same. The Equations 2.20 and 2.21 provide proof for the claims mentioned above, while Algorithm 4 outlines the secret key discovery process using differential power analysis in the Advanced Encryption Standard (AES) cryptographic algorithm. This form of DPA attack, together with a mean difference distinguisher, was initially introduced by Kocher et al. in 1999 [52].

$$\Delta_D[j] = \frac{\sum_{i=1}^{k} D_\Psi(H_i,b)\,T_i[j]}{\sum_{i=1}^{k} D_\Psi(H_i,b)} - \frac{\sum_{i=1}^{k} (1 - D_\Psi(H_i,b))\,T_i[j]}{\sum_{i=1}^{k} (1 - D_\Psi(H_i,b))}, \tag{2.20}$$

$$\Delta_D[j] \approx 2\left(\frac{\sum_{i=1}^{k} D_\Psi(H_i,b)\,T_i[j]}{\sum_{i=1}^{k} D_\Psi(H_i,b)} - \frac{\sum_{i=1}^{k} T_i[j]}{k}\right). \tag{2.21}$$

The computational complexity of the Algorithm 4 is dependent on several factors, including the size of the secret key in bytes, the maximum possible value for the key, the number of traces available, and the number of samples present in each trace.

Algorithm 4 Secret key discovery in AES using DPA with mean difference distinguisher.

Input:
M – a set of plaintext blocks $[k \times 16]$,
T – a set of traces $[k \times ts]$,
k – a number of traces,
ts – a number of samples in trace
Output:
K – a secret key $[1 \times 16]$
for $\forall \beta \in [1..16]$ **do**
  initialize the difference matrix $H\,[k \times 256]$
  initialize the difference matrix $\Delta\,[256 \times ts]$
  for $\forall \Psi \in [1..256]$ **do**
    $H_{[:,\Psi]} = M_{(:,\beta)} \oplus \Psi$
    $H_{[:,\Psi]} = SBOX\,(H_{[:,\Psi]})$
    initialize $G_1\,[1, ts]$
    initialize $G_2\,[1, ts]$
    for $\forall \Lambda \in [1..k]$ **do**
      $bit = D\left(H_{[\Lambda,k]}, b\right)$
      $G_{bit} = G_{bit} + T_{[\Lambda,:]}$
    end for
    average the values G_1 i G_2
    $\Delta_{[\Psi,:]} = |G_1 - G_2|$
  end for
  $K_{[1,\beta]} = row\,(max\,(\Delta))$
end for
return K

Chapter 3

Constructions of Boolean Functions

Boolean functions play a critical role in the design of cryptographic algorithms. To withstand linear cryptanalysis attacks, these functions must possess high nonlinearity. Bent functions are a class of Boolean functions that exhibit maximal nonlinearity for a given number of variables. Furthermore, Boolean functions must be balanced in the context of cryptographic algorithms. This chapter aims to investigate heuristic search techniques for generating Boolean functions that are both maximally nonlinear and maximally balanced. To this end, we explore various representations of Boolean functions, including binary and quaternary representations.

This chapter is organized as follows. Section 3.1 gives a motivation for research. Section 3.2 covers the necessary definitions and notions about Boolean and quaternary functions, along with the cryptographic criteria we consider. An overview of the literature concerning heuristics for finding Boolean functions with good cryptographic properties is given in Section 3.3. Section 3.4 presents details of the algorithms we use for our experiments, focusing on the representation of the candidate solutions and the genetic operators adopted. This section also describes the problem instances and the parameters we considered and discusses the results obtained by our experiments, possible transformations between binary and quaternary functions, and future work. Finally, Section 3.5 sums up the crucial contributions of the chapter.

3.1 Introduction

The role of Boolean functions is diverse and spans several research domains, such as communication, coding theory, and cryptography. Examples range from applications in combinatorics, such as the construction of Hadamard matrices [53], strongly regular graphs [54], and in coding theory, where they are used for constructing certain classes of codes such as Reed-Muller codes [55] and Kerdock codes [56].

Within the cryptography area, Boolean functions have many applications that result in a need for construction techniques able to produce Boolean functions of various sizes and properties.

Applications in cryptography are numerous: for designing hash functions [57], stream and block ciphers [58, 59] (in the form of vectorial Boolean functions for the latter case) as well as in the fully homomorphic encryption [60]. We concentrate now on stream ciphers.

For a Boolean function to be helpful in such constructions, it must satisfy many properties. One such property is nonlinearity, where the higher the value, the more nonlinearity there is. Informally speaking, the nonlinearity property tells us how far a function is from all affine functions and how difficult it is to conduct the cryptanalysis. Boolean functions that have maximal nonlinearity are called bent functions. Although they are not balanced (i.e., their truth tables do not have the same number of zeros and ones) and therefore not suitable for cryptography, there are methods to transform bent Boolean functions into balanced Boolean functions with high nonlinearity [43]. The problem of finding binary Boolean functions of n variables with the best possible combinations of cryptographic properties is of extreme difficulty. This stems from the impossibility of exhaustively exploring the corresponding search space, which grows superexponentially as 2^{2^n}, making complete enumeration unfeasible for $n > 5$. One has at his disposal three options to build Boolean functions: algebraic constructions, heuristics, and random search [61].

The main strength of algebraic constructions is that they can be proved to generate functions with specific properties and, in general, equally accessible to construct functions of any dimension. The main drawback is that they always result in the same functions (since they are deterministic), which means one is limited in the number of different functions one can obtain. The main advantages of random search are that it produces an abundance of different Boolean functions and is a relatively fast method. However, the quality of such functions (regarding their cryptographic properties) is almost always suboptimal. Finally, when discussing heuristic methods, they are usually positioned somewhere between the two approaches, as mentioned earlier: they generate a large number of good results in a relatively short time. However, there are some drawbacks when considering the search space size and the evaluation cost.

Construction techniques can be divided into primary constructions and secondary constructions. In primary constructions, one obtains new functions without using known functions. On the contrary, in secondary constructions, one uses already known functions to construct new functions [43].

Binary Boolean functions, i.e., mappings from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, represent the Boolean functions most often investigated in the literature. Still, we can consider the more general case over $\mathbb{Z}_q$ for $q > 2$. By doing so, we change the representation of the problem, the corresponding search space size, and possible target applications. As an example, Boolean functions over $\mathbb{Z}_q$ with $q > 2$ were first introduced to find codes for multicode code-division multiple access (MC-CDMA) systems [62].

In this chapter, except for binary Boolean functions, we concentrate on q-ary Boolean func-

tions and their application in cryptography (note that the cryptographic application is only one of the several possibilities). As we demonstrate, this problem is fascinating both from an evolutionary computation benchmark perspective and as a tool for constructing cryptographic primitives. We study the cryptographic properties of q-ary Boolean functions, i.e., mappings from $\mathbb{Z}_q^n$ to $\mathbb{Z}_q$, focusing in particular on the case of quaternary functions, where $q = 4$. This line of research is always motivated by the search for new algebraic constructions of cryptographically significant binary Boolean functions: the idea is to define quaternary functions with suitable properties and then derive the associated binary functions through projection mappings (such as the Gray map) [63].

With quaternary Boolean functions, the search space size equals 4^{4^n}, which is significantly more complex than the binary Boolean case. Still, from the representation point of view, one can quickly transform quaternary functions in n variables into binary Boolean functions of $2n$ variables, which means we need to work with half the number of variables.

3.2 Background

In the rest of the chapter, we denote the usual binary case with the name Boolean function. In contrast, with the name quaternary function, we consider a quaternary Boolean function.

3.2.1 Binary Boolean Functions

Let $n \in \mathbb{N}$. A Boolean function is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ where $\mathbb{F}_2$ is the Galois field with two elements. We denote the set of all n-tuples of the elements in the field $\mathbb{F}_2$ as $\mathbb{F}_2^n$. The set $\mathbb{F}_2^n$ represents all binary vectors of length n, which can be viewed as a $\mathbb{F}_2$-vector space [43]. The inner product of vectors $\vec{a}$ and $\vec{b}$ over the $\mathbb{F}_2$ field is defined as $\vec{a} \cdot \vec{b}$ and it equals $\vec{a} \cdot \vec{b} = \oplus_{i=1}^{n} a_i b_i$ with "$\oplus$" denoting addition modulo two. The Hamming weight (HW) of a vector $\vec{a}$, where $\vec{a} \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector.

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $(f(\overrightarrow{0,\ldots 0}), ..., f(\overrightarrow{1,\ldots,1}))$ that contains the function values of f, ordered lexicographically [43]. The *support* of f is the set of values $\vec{x} \in \mathbb{F}_2^n$ such that $f(x) \neq 0$. Since the vector of the output values in the truth table uniquely identifies a function f, we will refer to the Hamming weight of f as the size of its support.

The second unique representation of a Boolean function is the Walsh-Hadamard transform W_f that measures the correlation between $f(\vec{x})$ and all linear functions of the form $\vec{a} \cdot \vec{x}$, for a ranging in $\mathbb{F}_2^n$ [64]. Table 3.1 gives an example of the Walsh-Hadamard transform of a bent Boolean function with four inputs. The Walsh-Hadamard transform of a Boolean function f

Table 3.1: Walsh-Hadamard transform of a Boolean function with 4 inputs.

$\vec{x} \in \mathbb{F}_2^4$	$f(\vec{x})$	$W_f(\vec{x})$	$\vec{x} \in \mathbb{F}_2^4$	$f(\vec{x})$	$W_f(\vec{x})$
0000	0	4	1000	0	-4
0001	1	4	1001	1	4
0010	0	-4	1010	0	-4
0011	0	4	1011	0	-4
0100	1	4	1100	0	4
0101	1	4	1101	0	-4
0110	0	-4	1110	1	4
0111	1	4	1111	0	4

equals:

$$W_f(\vec{a}) = \sum_{\vec{x} \in \mathbb{F}_2^n} (-1)^{f(\vec{x}) \oplus \vec{a} \cdot \vec{x}}. \tag{3.1}$$

Algorithm 5 presents the Walsh-Hadamard transform for a Boolean function.

Algorithm 5 Walsh-Hadamard transform for a Boolean function.

Require:x is an array of 2^n Boolean binary values
Ensure:y is the Walsh-Hadamard transform of **x**

```
y ← x
for i = 1 to n do
    m ← 2^(i−1)
    for j = 0 to 2^n − 1 do
        if (j mod 2^i) < 2^(i−1) then
            s ← y[j] + y[j + m]
            t ← y[j] − y[j + m]
            y[j] ← s
            y[j + m] ← t
        end if
    end for
end for
return y
```

A Boolean function f is balanced if the Walsh-Hadamard coefficient of the null vector $\vec{0} = (0, \cdots, 0)$ equals zero [65]:

$$W_f(\vec{0}) = 0. \tag{3.2}$$

Alternatively, in the truth table representation, a Boolean function with n inputs is balanced if its Hamming weight equals 2^{n-1}, i.e. if it is composed of an equal number of zeros and ones.

A Boolean function f used in the design of stream and block ciphers should lie at a large

Hamming distance (HD) from all affine functions to resist linear cryptanalytic attacks. This distance corresponds to the nonlinearity of f, which is defined as the minimum HD between f and all affine functions [43]. The nonlinearity N_f of a Boolean function f expressed in terms of the Walsh-Hadamard coefficients of f is [43]:

$$N_f = 2^{n-1} - \frac{1}{2} max_{\vec{a} \in \mathbb{F}_2^n} |W_f(\vec{a})|. \tag{3.3}$$

A natural question is to determine what is the maximum nonlinearity a Boolean function can attain. This can be derived from Parseval's relation:

$$\sum_{a \in \mathbb{F}_2^n} W_f(a)^2 = 2^{2n}, \tag{3.4}$$

which implies that the mean of $W_f(a)^2$ equals 2^n, and $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$ is at least equal to the square root of this mean.

From Eq. (3.4), it follows that the maximal value of the Walsh-Hadamard spectrum equals at least $2^{n/2}$ which occurs in the case of bent Boolean functions. Bent functions only exist for even number of variables, and they are never balanced. The expression for the nonlinearity of bent functions is: [53, 66]:

$$N_f = 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{3.5}$$

If a Boolean function f has the correlation immunity property of order t, an even number of inputs n, and $k \leqslant \frac{n}{2} - 1$, then its nonlinearity N_f has an upper bound as follows:

$$N_f = 2^{n-1} - 2^{\frac{n}{2}-1} - 2^k, \tag{3.6}$$

where k equals $t+1$ if f is balanced or has HW divisible by 2^{t+1} and k equals t otherwise.

3.2.2 Quaternary Boolean Functions

To generalize the notion of Boolean functions, one can take into account the residual class ring (Galois ring) $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Specifically, we consider the case where $q = 4$. A quaternary function F is a mapping from $\mathbb{Z}_4^n$ to $\mathbb{Z}_4$, i.e., it is a $\{0,1,2,3\}$-valued function. The set of all n-tuples of elements in $\mathbb{Z}_4$ is denoted as $\mathbb{Z}_4^n$. Let i denote the complex number such that $i^2 = -1$. There is a group-isomorphism between the set $\{0,1,2,3\}$ and $(\pm 1, \pm i)$ under the standard isomorphism $x \to i^x$ [67, 68]. We denote addition modulo 4 with "+".

As discussed in [69], a quaternary function F can be represented using a truth table, denoted as $(F(\overrightarrow{0,\ldots 0}), ..., F(\overrightarrow{3,\ldots,3}))$. Analogous to the binary case, the support of a quaternary function F is the set of vectors u belonging to the space $\mathbb{Z}_4^n$ such that $F(u) \neq 0$, and the size of the support of F is referred to as the Hamming weight of F.

Table 3.2: Walsh-Hadamard transform of a quaternary function with two inputs.

$z \in \mathbb{Z}_4^2$	$F(z)$	$W_F(z)$
00	1	-4
01	2	4
	$\vdots$	
23	1	$-4+4i$
30	3	$4+4i$
31	0	-4
32	2	4
33	2	$4+4i$

The Walsh-Hadamard transform of a quaternary function F equals:

$$W_F(a) = \sum_{v \in \mathbb{Z}_4^n} i^{a \cdot v + F(v)}, \tag{3.7}$$

where $a \cdot v$ denotes the usual inner product in $\mathbb{Z}_4^n$ (mod 4) and $a \in \mathbb{Z}_4^n$. One example of the Walsh-Hadamard transform of a quaternary function is given in Table 3.2 (note that we do not give a complete truth table here).

A quaternary function is balanced if and only if $W_F(\vec{0})$ equals 0. Alternatively, it is balanced if and only if for all $i \in \mathbb{Z}_4$ the cardinality of the set $\eta_i(F)$ equals 4^{n-1}, where $\eta_i(F) = \left\{u \in \mathbb{Z}_4^n | F(u) = i\right\}$.

To define the nonlinearity, we can use either the Hamming metric (N_F^H) or the Lee metric (N_F^L) as the underlying distance. In this chapter, we work with the Lee metric since there is an isometry (distance preserving bijection) between $\mathbb{Z}_4^n$ equipped with the Lee distance and $\mathbb{F}_2^{2n}$ equipped with the Hamming distance when Gray mapping is used (for details see Section 3.4.2). The Lee weights LW of values $0,1,2,3 \in \mathbb{Z}_4$ are $0,1,2,1$. The Lee distance of two elements $a,b \in \mathbb{Z}_4^n$ equals $LW = (a+b)$. The nonlinearity of a function F under the Lee distance can be defined as:

$$N_F^L = 4^n - \max_{a \in \mathbb{Z}_4^n, b \in \mathbb{Z}_4} \left\{ Re(i^b W_F(a)) \right\} \tag{3.8}$$

$$= 4^n - \max_{a \in \mathbb{Z}_4^n} \left\{ |Re(W_F(a))|, |Im(W_F(a))| \right\}, \tag{3.9}$$

where $Re(z)$ and $Im(z)$ denote the real and imaginary part of the complex number z.

A quaternary function is bent if $Re(|W_F(a)|) = 2^n \ \forall a \in \mathbb{Z}_4^n$. The nonlinearity of a bent

quaternary function with n inputs equals:

$$N_F^L = 2^{2n} - 2^n. \tag{3.10}$$

Note that in $\mathbb{Z}_4^n$ a function can be bent for any dimension n while in $\mathbb{F}_2^n$ a function can be bent only if n is even.

3.3 Related Work

As mentioned in Section 3.1, there are many successful applications of heuristic techniques when constructing Boolean functions usable in cryptography [**?**]. This section focuses on several works that explore different approaches to evolving highly nonlinear Boolean functions with maximal nonlinearity.

To begin, Millan et al. utilize a genetic algorithm to evolve Boolean functions with high nonlinearity [70]. Building on this, Millan, Clark, and Dawson further enhance the genetic algorithm's strength by combining it with hill climbing and a resetting step to identify highly nonlinear Boolean functions with up to 12 variables [71]. Simulated annealing is also a promising approach, as demonstrated by McLaughlin and Clark, who experiment with generating Boolean functions with optimal values for various properties, such as algebraic immunity, fast algebraic resistance, and algebraic degree [72]. The authors extend their analysis to Boolean functions with up to 16 inputs.

Additionally, Picek, Jakobovic, and Golub explore the use of genetic algorithms and genetic programming to generate Boolean functions with multiple optimal properties [73]. The authors transform the genetic programming tree (as a genotype) into the truth table representation for evaluation purposes. Particle Swarm Optimization (PSO) is another good algorithm, as demonstrated by Mariot and Leporati, who use PSO to identify Boolean functions with good trade-offs of cryptographic properties for dimensions up to 12 inputs [74]. Furthermore, Hrbacek and Dvorak employ Cartesian genetic programming to evolve bent Boolean functions with up to 16 inputs, testing several algorithm configurations to expedite the evolution process [75]. The authors find a bent function in each run for sizes between 6 and 16 variables without limiting the number of generations.

Picek et al. compare the effectiveness of Cartesian genetic programming and genetic programming when looking for highly nonlinear balanced Boolean functions with eight inputs [76]. In that work, the authors show that Cartesian genetic programming performs favorably compared with other evolutionary approaches to cryptographically relevant Boolean functions. Picek et al. [77] investigate several evolutionary algorithms to evolve Boolean functions with different values of the correlation immunity property. In the same paper, the authors also discuss the prob-

lem of finding correlation-immune functions with minimal Hamming weight, where they experiment with Boolean functions with eight inputs. Picek et al. investigate several different evolutionary algorithms and fitness functions for Boolean functions of 8 inputs [78]. They show that genetic programming and Cartesian genetic programming outperform genetic algorithms and evolution strategies in many relevant test scenarios. Next, Picek et al. investigate the evolution of balanced Boolean functions of up to 16 inputs fulfilling several cryptographic properties and the evolution of minimal Hamming weight and different correlation immunity order Boolean functions [79]. In [80], authors search for a bent quaternary function in n variables and use a Gray mapping to obtain bent binary functions in $2n$ variables. Picek, Sisejkovic, and Jakobovic use immunological algorithms to evolve either bent or highly nonlinear Boolean functions with up to 16 inputs [81]. Finally, Picek and Jakobovic use genetic programming to evolve algebraic constructions that are then used to construct bent Boolean functions [82]. In [83], authors use a genetic algorithm and genetic programming to create vectorial bent Boolean functions with up to 6 inputs, showing practical applications in authentication codes.

Clark et al. experimented with simulated annealing to design Boolean functions using spectral inversion [84]. They observe that several cryptographic properties of interest are defined in terms of the Walsh-Hadamard transform values. Based on Parseval's theorem, one can infer what values the Walsh-Hadamard spectrum should have. Note that it is impossible to know how these values should be permuted since the inverse Walsh-Hadamard transform maps to a pseudo-Boolean function. Consequently, when generating a Walsh-Hadamard spectrum containing these values, it is necessary to verify that it corresponds to a Boolean function. Mariot and Leporati [85] also adopt the spectral inversion method, designing a genetic algorithm where the genotype consists of the Walsh-Hadamard values to permute in order to evolve semi-bent Boolean functions. Finally, in [86], authors are looking for a balanced Boolean function with maximal nonlinearity and a varying number of inputs using Estimation of Distribution Algorithms. The authors confirm the problem's difficulty, pointing to the search space size and the high level of symmetries.

3.4 Experiments and Results

3.4.1 Binary Boolean Functions

Before discussing the experimental setting and presenting results, we briefly discuss the problem's difficulty. A Boolean function f of n variables can be represented with a string of 2^n values, and the search space size is equal to 2^{2^n}, as given in Table 3.3 for several sizes of interest. Note that we always need to transform our solutions into the truth table representation to calculate nonlinearity and the Walsh-Hadamard spectrum. Next, in the same table, we give

Table 3.3: The search space size for the investigated sizes of functions and nonlinearities for bent Boolean functions.

Properties \ n	4	6	8	10	12
Search space	2^{16}	2^{64}	2^{256}	2^{1024}	2^{4096}
Bent N_f	6	28	120	496	2 016
Balanced conj. N_f [87]	4	26	116	492	2 010
Balanced best N_f [88]	4	26	116	486	1 992

optimal values for the nonlinearity property and each size of the Boolean function we consider in this section. Finally, we show the best-obtained results by algebraic constructions for a balanced Boolean function with maximal nonlinearity [87]. We emphasize the best theoretical result, according to the equations 3.5 and 3.6 is $N_{f_{bent}} - 2$.

Algorithms and Representations

We experiment with two different encodings and compare their efficiency: bitstring and tree representation.

The simplest genotype to use is the bitstring since, in that case, there is no need for mapping between it and the truth table representation of a Boolean function. However, as given in Table 3.3 we can observe that the genotype size increases exponentially with the number of variables of a Boolean function. To state it differently, with the increase of the number of variables, quite soon one can expect problems with the size of solutions if using the truth table representation.

For the algorithm, we use a simple GA with the tournament selection where its size equals 3 [89], and the population size is 100. We experiment with two mutation operators and two crossover operators. The mutation operators we use are simple mutation, where a single bit is an inverted, and balanced mutation, which preserves a function balancedness. The crossover operators are one-point and balanced crossover, performed at random for each new offspring.

As the second encoding, we use tree-based GP in which a Boolean function is represented by a syntax tree [90]. The function set for GP in all the experiments is OR, NOT, XOR, AND, XNOR, AND with one input inverted and IF, which takes three arguments and returns the second one if the first evaluates to true and the third one otherwise. The terminals correspond to n Boolean variables. Boolean functions can be represented using only XOR and AND operators, but it is pretty easy to transform the function from one notation to the other. GP also uses tournament selection with the tournament size equal to 3.

The crossover is performed with a simple tree crossover with 90% bias for functional nodes[91]. We use a single mutation type, and a maximum tree depth of 7. The population size for the GP equals 100.

Common Parameters

The experimental setup includes $N = 30$ independent trials for each configuration, and the stopping criterion for all algorithms is set to 500000 evaluations. We apply an individual mutation probability of 0.8 to all the algorithms and representations. It should be noted that the mutation probability is used to determine whether an individual will be mutated. Once selected, the mutation operator is executed only once on a given individual. For instance, if the mutation probability is set to 0.8, then on average, 8 out of every ten new individuals will be mutated, as shown in Algorithm 3.

Fitness Functions

We consider three fitness functions in our experiments. Note that all three functions are well established and used in related work in the evolution of Boolean functions.

Bent functions In the simplest version of the fitness function we aim to maximize the nonlinearity value:

$$fitness_1 = N_f. \tag{3.11}$$

The second version of fitness function improves on Eq. 3.11 and adds the second term where we aim to minimize the number of occurrences of values different from $2^{\frac{n}{2}}$ as given by the Walsh-Hadamard spectrum. Here we are again interested in the maximization of the following expression:

$$fitness_2 = N_f + \frac{1}{\tau}, \tag{3.12}$$

where τ equals the number of occurrences of the value different from $2^{\frac{n}{2}}$ in the Walsh-Hadamard spectrum. Since we still consider nonlinearity as the primary parameter, we scale the second term in the fitness function in the range $[\frac{1}{2^n}, 1]$. Note that if $\tau = 0$ then the fraction is set to 1.

Balanced highly nonlinear functions Finally, in the third fitness function we aim to simultaneously optimize balancedness and maximal nonlinearity. The function we maximize is the following:

$$fitness_3 = N_f - |2^{n-1} - HW(f)|, \tag{3.13}$$

where $|2^{n-1} - HW(f)|$ equals to 0 when the function f is balanced.

Results

We give the results in the *max value/median value* notation since the first number has greater relevance from the practical perspective (we are interested in a single function with as good as possible nonlinearity value) while the second number has more relevance from the optimization perspective to assess what is the average behavior of the algorithms and operators we tested. Note that we opted to use median rather than the average value so not to assume a normal distribution of data. To be more readable, we bold values equal to the optimal ones.

In Table 3.4 we give the results for bitstring encoding and fitness functions 3.11 and 3.12. We see that for 4 and 6 inputs, GA can find optimal results in 100% of cases. Moreover, GA can reach an optimal solution when considering the input dimension of 8 variables. On the other hand, starting with $n = 8$, GA is not able to reach optimal value for all higher dimensions. Fitness function 2 achieves better solutions because of simultaneous optimization of nonlinearity and number of occurrences maximal value in the Walsh-Hadamard spectrum. Moreover, one should keep in mind the exponential growth of the bitstring individual, and finding the optimal value gets more difficult.

Table 3.4: Results for the bitstring representation, fitness 1 and fitness 2.

f \ n	4	6	8	10	12
$fitness_1$	**6**/6	**28**/28	116/114	480/480	1 974/1 970
$fitness_2$	**6**/6	**28**/28	**120**/114	480/480	1 980/1 974

Next, in Table 3.5 we give the results for the tree representation and fitness functions 1 and 2. Note that here, we can reach optimal values for all inputs. Therefore, our results corresponds to the ones from related work for cases when comparing efficiency of solution representation. Moreover, it is noticeable that for input $n = 10$, fitness function 2 has better median value, and for both fitness functions, optimal solution can be reached in less than 10000 evaluations on average.

Table 3.5: Results for the tree representation, fitness 1 and fitness 2.

f \ n	4	6	8	10	12
$fitness_1$	**6**/6	**28**/28	**120**/120	**496**/480	**2 016**/1 984
$fitness_2$	**6**/6	**28**/28	**120**/120	**496**/488	**2 016**/1 984

Except looking for bent Boolean functions, we try to find balanced Boolean functions with maximal nonlinearity. Here, we want to emphasize the difficulty of the problem because it represents multiple criteria optimization. Moreover, in Table 3.3 the best obtained experimental results are given as well as results obtained by conjecture by using algebraic constructions.

In Table 3.6, we give the results for bitstring encoding and fitness function 3. In the table, we show results obtained by three different mutation operators. It can be seen that all mutation operators can find an optimal solution for dimensions $n = 4$ and $n = 6$.

Table 3.6: Results for the bitstring representation, fitness 3 and various mutation operators (MO).

MO \ n	4	6	8	10	12
random	**4**/4	**26**/24	114/112	478/478	1 970/1 966
balanced	**4**/4	**26**/26	**116**/112	480/478	1 976/1 968

Finally, In Table 3.7 we give the results for tree encoding and fitness function 3 and different tree depths. We see that when $n < 10$, GP can find optimal results. On the contrary, GP can not reach optimal values in all the runs when considering the input dimension of 10 or 12 variables. For inputs $n = 6$ and $n = 8$, one can observe that too shallow or too deep trees more often reach suboptimal values compared to depth 7.

Table 3.7: Results for the tree representation, fitness 3 and different tree depths.

Depth \ n	4	6	8	10	12
5	**4**/4	**26**/24	**116**/112	480/480	1 980/1 966
7	**4**/4	**26**/26	**116**/114	482/480	1 984/1 968
11	**4**/4	**26**/24	**116**/112	480/480	1 980/1 968

Interestingly, the search for a balanced Boolean function of maximum nonlinearity of 8 variables resulted in functions of nonlinearity 116, which is the same so far best-found solu-

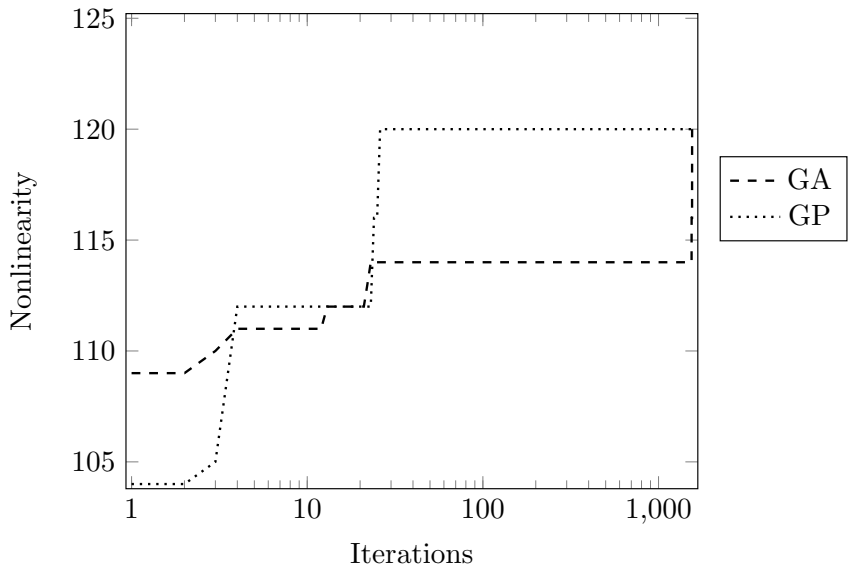


Figure 3.1: Fitness 2 maximization for Boolean function with 8 inputs. Iterations showed in logarithmic scale.

tions, but leaves open the question of the existence of a balanced Boolean nonlinearity function 118. Note that despite the theory, there exists a dose of doubt on whether such functions with nonlinearity equal to 118 exist.

Figure 3.1 shows fitness function 2 for GP and GA in obtaining bent Boolean function with 8 inputs. One can see that GP reaches optimal solution much faster, after a lower number of evaluations than GA. The reason for such behavior is the way how GP searches solution space. With one mutation in a tree representation, multiple function's outputs are simultaneously changed. On the contrary, one should make much more changes in a bitstring representation to achieve the same effect. As a result, GP much easier and faster can find a solution.

3.4.2 Quaternary Boolean Functions

Representations and Algorithms

In this section, we experiment with two different encodings and compare their efficiency.

Integer Encoding Considering both Boolean and quaternary functions, a truth table representation is probably the most obvious one. In this case, a quaternary function of n variables is represented with a string of length 4^n which corresponds to its lexicographically ordered truth table. Each element of the string can assume values in $\{0,\ldots,3\}$, which are initialized by sampling from the uniform distribution.

Appropriate crossover and mutation operators need to be defined in order to be able to use genetic algorithms with this encoding. In our experiments, we use a single mutation operator, which randomly chooses a single gene (element of the string) and generates its new value with

Table 3.8: Genetic programming functions.

Function	Definition
AND	$(a \cdot b) \mod 4$
OR	$(a + b) \mod 4$

uniform probability. There are three crossover operators in use: the first one is a simple single-point crossover, which combines the first part of one parent and the second part of the other parent into a single child. We also use a two-point variant which takes two parts from one and one part from the other parent; in both operators, crossover points are randomly chosen. The final operator is the average crossover, which constructs each gene in the child by using the average value of corresponding genes from both parents. The choice of the crossover operator is made randomly each time a crossover is performed.

Tree Encoding As opposed to truth table encoding, the other option we consider is to use a symbolic representation of a quaternary function. This is performed in a way such that genetic programming can be used to evolve a quaternary function in the form of a syntactic tree. Here, the terminal set consists of the n input quaternary variables, denoted $\{v_0, \ldots, v_n\}$. The function set (i.e., the set of inner nodes of a tree) should consist of appropriate quaternary Boolean operators that allow the definition of quaternary functions with n inputs.

In our experiments, we use binary functions AND and OR. These functions are defined as shown in Table 3.8. It could be useful to implement the NOT function as well, because all logic functions can be expressed as a linear combination of that function and AND or OR function. Additionally, with more functions it is possible to increase the GP expressiveness. The main problem is to define true and false values in quaternary logic, and moreover the definition of the NOT function in this case is not unique, which is why we opted not to use it in our experiments.

When evaluated, the same tree is parsed for every possible input combination of variables appearing in the tree. Each result (evaluated at the root node) is written in the corresponding position of the truth table, and the tree is then given its fitness based on the properties of the resulting truth table. The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point, and context preserving crossover [91].

Common Parameters and Fitness Function

Regardless of the encoding, we use the same selection process to conduct the search – steady-state selection process, shown in Algorithm 3, where in each iteration only one individual from the population is replaced with a new one. The selection of the individual to be replaced is

performed in a tournament of size three: the algorithm selects three individuals at random and eliminates the worse of those. The remaining tournament survivors are then used as parents to create a new individual using crossover.

Following the creation, the new individual immediately undergoes mutation, which depends on the mutation rate parameter. In our experiments this parameter equals 0.3, which results in three out of ten new individuals being mutated on average. This kind of algorithm is convenient since it eliminates the need for specifying the crossover mutation rate, and in our previous experience provides a steady rate of convergence.

In all the experiments the number of independent trials N for each configuration is 30 and the stopping criterion for all algorithms equals 500 000 evaluations or reaching the maximal nonlinearity. The population size in both experiments is 200 and maximum tree depth is 4. The parameters given here are selected after a tuning phase. Note that the tree size is relatively small in our experiments but we found this size to be the first one where all experimental runs results in bent quaternary functions.

The fitness function we aim to maximize is described in Eq. (3.9). We use this expression instead of Eq. (3.8) because it has a lower computational complexity. To speed up the calculation of the Walsh-Hadamard transform, we use a custom implemented inner product cache, C. Note that Eq. (3.7) can be decomposed into Eq. (3.14). The inner product calculation in $\mathbb{Z}_4^n$ (mod 4) has complexity $\mathcal{O}(\log n)$ and it is symmetric. To avoid calculating the same values repeatedly, we store once the calculated inner product values in cache C. Additionally, the cache size is decreased using the symmetry property.

$$W_F(a) = \sum_{v \in \mathbb{Z}_4^n} C\left[\min(a,v), \max(a,v)\right] \cdot i^{F(v)} \tag{3.14}$$

Results

In this section, we present results obtained in our experiments. Table 3.9, we give details on search space sizes and maximal possible nonlinearities for all quaternary function dimensions we consider. We can see that the search space size makes it impossible to conduct an exhaustive search already for dimension $n = 3$.

In Tables 3.10 and 3.11, we give results for integer and tree encodings, respectively. Note that when we are able to find bent quaternary function for a certain dimension, we denote it in bold style in tables. When considering integer encoding, we are able to find bent quaternary functions only for the smallest size we investigate ($n = 2$). All the larger sizes represent a significant problem for integer encoding with no success in obtaining bent functions. Observe we do not conduct experiments for integer encoding with dimensions larger than 6 since those sizes result in nonlinearities that are significantly smaller than the maximal ones. Additionally,

Table 3.9: Search space sizes and maximal nonlinearity values.

Size	Search space size	Maximal nonlinearity N_F^L
2	4^{16}	12
3	4^{64}	56
4	4^{256}	240
5	$4^{1\,024}$	992
6	$4^{4\,096}$	4 032
7	$4^{16\,384}$	16 256
8	$4^{65\,536}$	65 280

Table 3.10: Results for the integer representation.

Size	Min	Max	Average	Std dev
2	4	**12**	**12**	0.00
3	18	55	54.20	0.40
4	228	233	231.53	0.78
5	961	968	965.50	1.35
6	3 955	3 963	3 959.60	2.06

those quaternary function sizes are extremely computationally expensive to calculate.

Table 3.11 gives results for the tree encoding (recall, in our tree encoding there are 4 possible values, which differentiates our representation from the "standard" one as used in genetic programming). As it can be seen, for all dimensions we consider, we are able to find bent quaternary functions. Even more, we are able to find bent functions in every run: this suggests that although extremely difficult for integer encoding, the problem is easy for the tree encoding. The only difficulty for larger sizes should be the computational cost when running the Walsh-Hadamard transform and nonlinearity calculation.

Next, we first discuss how to map bent quaternary Boolean functions in n variables into bent binary Boolean functions in $2n$ variables. Then we compare such results (i.e., constructed bent binary Boolean functions) with state-of-the-art results from literature. Finally, we briefly discuss several possible future research directions.

From Quaternary to Binary Functions

Once we obtain quaternary functions, the question is how to use them. One option is to use them directly as they are, which is the approach taken for instance by Schmidt where he uses

Table 3.11: Results for the tree representation.

Size	Min	Max	Average	Std dev
2	0	**12**	**12**	0.00
3	0	**56**	**56**	0.00
4	0	**240**	**240**	0.00
5	0	**992**	**992**	0.00
6	0	**4 032**	**4 032**	0.00
7	0	**16 256**	**16 256**	0.00
8	0	**65 280**	**65 280**	0.00

quaternary constant-amplitude codes for multicode CDMA [62]. Another option is to transform quaternary functions into binary Boolean functions. Here, we follow this line of work. In order to transform quaternary into binary Boolean functions, there are several possible mappings [69]. In our experiments, we investigate the Gray mapping since it is a distance-preserving bijection between $\mathbb{Z}_4^n$ in Lee distance and $\mathbb{F}_2^{2n}$ in the Hamming distance. Gray mapping $\phi : \mathbb{Z}_4 \to \mathbb{F}_2 \times \mathbb{F}_2$ is defined as:

$$0 \to (00),\ 1 \to (01),\ 2 \to (11),\ 3 \to (10).$$

Alternatively, if $u, v \in \mathbb{F}_2$, and by denoting $w = u + 2v$ (i.e., by using 2-adic expansion), the Gray mapping equals $\phi(w) = (v, u \oplus v)$. The mapping ϕ can now be extended naturally to $\mathbb{Z}_4^n$. Observe that the same mapping can be used to transform from binary Boolean functions into quaternary Boolean functions.

Comparison with the State-of-the-art Results for Binary Boolean Functions

Since we reach 100% success in obtaining bent quaternary functions in every dimension we consider (see Table 3.11), that translates into 100% success rate in obtaining binary bent functions in $2n$ variables. When comparing those results from the state-of-the-art in the literature (when considering heuristic techniques), we see that our results represent a significant improvement. Namely, Picek, Sisejkovic, and Jakobovic investigate the performance of two immunological algorithms, as well as genetic algorithm and evolution strategy (for all algorithms they consider bitstring and floating-point representation). Their results show that only for Boolean functions with 6 variables they are able to find bent Boolean functions. For all larger sizes, the nonlinearity is significantly below the maximal attainable one [81].

Next, Picek et al. investigate a number of evolutionary algorithms to evolve bent Boolean functions with 8 inputs [92]. The results obtained there suggest that Cartesian genetic pro-

gramming and genetic programming perform the best where GP with small tree sizes can reach maximal nonlinearity in 100% of cases. Naturally, direct comparison with our work is difficult since they consider only functions with 8 inputs and it remains to be seen how their approach would perform for larger Boolean function sizes.

Hrbacek and Dvorak use Cartesian genetic programming to evolve bent Boolean functions in dimensions $[6, \ldots, 16]$. They report success (i.e., they find bent Boolean function) in each experimental run, which makes their results directly comparable with ours. Still, they use a computer cluster with 112 nodes (Intel E5-2670) and 128 Gb of RAM [75] while we use a desktop computer with Intel i5-3470 and 8 Gb of RAM. Consequently, our approach seems to be much more efficient.

Finally, Picek and Jakobovic use genetic programming to design secondary constructions that are then used to construct bent Boolean functions [82]. They can find bent Boolean functions for much larger sizes than we give here but we note that they do not evolve larger Boolean functions directly but use a clever trick that enables them to expand small bent Boolean functions into larger ones. Additionally, the secondary construction method used in that work generates a limited number of bent functions, since it relies on the initial set of bent functions of less variables. The approach presented in this chapter allows finding different bent functions in every algorithm run.

3.5 Conclusions

This chapter addressed the construction of maximal nonlinearity Boolean functions through evolutionary algorithms. To resist linear cryptanalysis attacks, Boolean functions need to have high nonlinearity. Bent functions are Boolean functions with maximal possible nonlinearity for a given number of variables. It is also crucial for functions to be balanced for usage in cryptographic algorithms.

Our results suggest that one can use evolutionary algorithms to evolve many different sizes of Boolean functions, whereas the best performing algorithm we consider genetic programming. Naturally, that success stems from representation rather than a specific selection strategy.

Even though we did not reach the optimal solutions for higher dimensions in constructing maximal nonlinear Boolean functions, we showed the critical role of choosing fitness functions and evolutionary operators.

Stepping away from binary Boolean functions, we introduce the problem of evolving quaternary bent Boolean functions. We experiment with two encodings, integer, and tree encoding, showing that the latter offers superior results. The results for quaternary tree encoding show that we can obtain bent functions for all dimensions we experiment with.

Since we find bent functions in every run of the experiments, this naturally means we have

100% success finding bent quaternary functions in n variables. We use Gray mapping to obtain bent binary functions in $2n$ variables. Since this is a deterministic procedure, we always construct bent binary functions (where we go up to 18 variables). Our results are comparable or better than those obtained with other techniques when evolving bent binary Boolean functions [82]. Finally, we note the efficiency of quaternary tree encoding, which we believe could also be competitive in other problems when Boolean functions are considered.

Chapter 4

Constructions of Vectorial Boolean Functions

The research areas of Boolean functions, their generalizations, and vectorial Boolean functions have been highly active. To the best of our knowledge, this study is the first to explore the evolution of vectorial Boolean functions where the output dimension is strictly smaller than the input dimension. The concept of differential uniformity for a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where n represents the input dimension and m the output dimension, has been the subject of several interesting problems. When $n = m$, the best differential uniformity possible is two, and heuristics can find such functions. However, when $n > m > n/2$, the differential uniformity is bounded by $2^{n-m} + 2$ from below. Unfortunately, such functions are only known for dimensions equal to $n = 4, 5$. This chapter explores the use of several evolutionary algorithms and problem sizes to find functions with a differential uniformity of 6. Our results indicate that several solution encodings can find such functions only in dimensions $(4, 2)$ and $(5, 3)$, where the notation (n, m) is used. As differentially 6-uniform functions were previously known for these sizes, our results can provide a source of new functions in those dimensions and indicate that either such functions do not exist or are challenging to find for $(6, 4)$.

The remainder of this chapter is organized as follows. Section 4.1 motivates our research. In Section 4.2, we introduce the notation used and relevant theory about vectorial Boolean functions. In Section 4.3, we discuss related work. Section 4.4 presents our experimental setup and results and discusses the obtained results and potential future research directions. Finally, Section 4.5 provides a brief conclusion.

4.1 Introduction

In cryptography, Boolean functions and their generalizations vectorial Boolean functions (also called S-boxes) have a prominent usage in symmetric key cryptography [28]. Without carefully

chosen S-boxes, such ciphers would be easier to break. Symmetric key cryptography can be divided into stream ciphers and block ciphers, where again (vectorial) Boolean functions have different usages. However, their goal is always the same: to improve the resiliency of cryptographic algorithms (commonly known as ciphers) against various cryptanalyses. For instance, when discussing block ciphers, we want to have resistance against differential [93] and linear [94] cryptanalysis. On the other hand, with stream ciphers, some attacks we want to have resilience against are fast correlation attack [95], Berlekamp-Massey attack [42], and algebraic attack [96].

The resilience against the aforementioned attacks stems from smartly chosen (vectorial) Boolean functions where in block ciphers, we mostly use S-boxes where the input size is equal to the output size, i.e., (n,n) functions like one that can be found in the AES cipher [97] or where the output size is somewhat smaller than the input size like in the DES cipher [98]. On the other hand, in stream ciphers, one usually uses Boolean functions or S-boxes where the output dimension is strictly smaller than the input dimension [43].

Naturally, since there is a plethora of different ciphers, the requirements on those Boolean functions or S-boxes differ (e.g., different sizes or cryptographic properties they need to possess). Accordingly, to produce such a diversity of functions, researchers developed over the years several construction techniques. Such techniques can be divided into algebraic constructions, random search, heuristics, and combinations of those techniques [76]. Here we are interested in investigating how one can use one type of heuristics, namely evolutionary algorithms (EA), in the evolution of (vectorial) Boolean functions with good cryptographic properties. Naturally, as already said, this area is a very active research domain, which also holds when applying heuristic techniques. Results obtained up to now (see Section 4.3) suggest that evolutionary algorithms are a perfect choice for the evolution of Boolean functions where it is possible to obtain optimal results concerning many properties and Boolean function sizes. On the other hand, when considering S-boxes, we observe that most works consider functions where the input and output dimensions are the same size (which is also the combination primarily used in practical applications). However, except for the smallest size of practical importance (4×4), the results suggest that EAs cannot compete with algebraic constructions nor achieve optimal results.

Moreover, within the block ciphers domain, one could follow several design strategies: Substitution Permutation Network (SPN), Feistel structure, and ARX structure [99]. Those structures have in common that they use mathematical operations to provide confusion and diffusion effects [46]. The confusion principle means that the cipher output statistics should depend on the cipher input statistics in a manner too complicated to be exploited by the attacker. The diffusion principle relates to the fact that each digit of the input and each digit of the secret key should influence many digits of the output.

A common way to provide confusion is to use S-boxes, e.g., [44]. S-boxes, which are

mappings between $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$, provide a nonlinear relationship between the n input bits and the m output bits in a controllable fashion for a specific secret key. When used in the SPN structure, S-boxes must be bijective (and then, $n = m$). Well-known examples of such ciphers are AES [100] and PRESENT [101]. When used in the Feistel structure, n and m sizes can differ, and both directions are possible. For instance, in the DES cipher [98], the input dimension equals 6, while the output dimension equals 4. In the CAST cipher [59], the input dimension is 8, while the output dimension is 32.

All S-boxes have in common that they need to fulfill many criteria so that the cipher using them can resist cryptanalyses. For instance, large nonlinearity will make the cipher more resilient against linear cryptanalysis [94], while small differential uniformity will make the cipher more resilient against differential cryptanalysis [93]. We can use algebraic constructions and heuristics to obtain S-boxes fulfilling those properties (among other relevant ones). Interestingly, although this problem has been an active research domain for several decades, our current knowledge is still limited, and we know only a handful of algebraic constructions [44].

At present, only a limited number of algebraic techniques are available to generate bijective S-boxes with optimal differential uniformity. Notably, the preeminent method involves utilizing monomial power functions, represented by the form $F(x) = x^d$, where d denotes the degree of the monomial power function. The best possible (and known) differential uniformity value then equals 2 for any odd n and also for $n = 6$. For n even and larger than 6, the best differential uniformity of (n,n)-permutations is an open question. Functions having differential uniformity equal to 2 are called Almost Perfect Nonlinear (APN). When considering heuristics, currently we are able to generate APN functions only for dimensions $n = 5,7$ [102]. The notion of APN function, i.e., differentially 2-uniform function, can be weakened and then we consider differentially δ-uniform (n,m)-functions. Interestingly, such functions are much less explored when $m \neq n$ and we know of even fewer algebraic constructions than for APN functions [103].

It is an open problem whether there exist differentially δ-uniform $(n,n-k)$ functions with $k \geq 2$, k significantly smaller than $\frac{n}{2}$, $\delta < 2^{k+1}$, and $n > 5$ [104] (constructions exist for k near $\frac{n}{2}$, see [105]). More concretely, if we set $k = 2$, it is still unsolved whether there exist functions $(n,n-2)$ that have differential uniformity less than 8 when $n > 5$.

This chapter investigates how to evolve vectorial Boolean functions ((n,m) functions) where the output size is strictly smaller than the input size. Moreover, we are interested in the evolution of bent (n,m) functions, which necessitates that the input dimension n is always even and that the output dimension m is smaller or equal to $\frac{n}{2}$. As far as we know, we are the first to consider this problem that has practical applications in authentication schemes [106] and secret sharing [107, 108]. However, we emphasize that this problem is also interesting from the combinatorial optimization perspective. To provide extensive results, we experiment with three fitness functions, three encodings, and some function sizes.

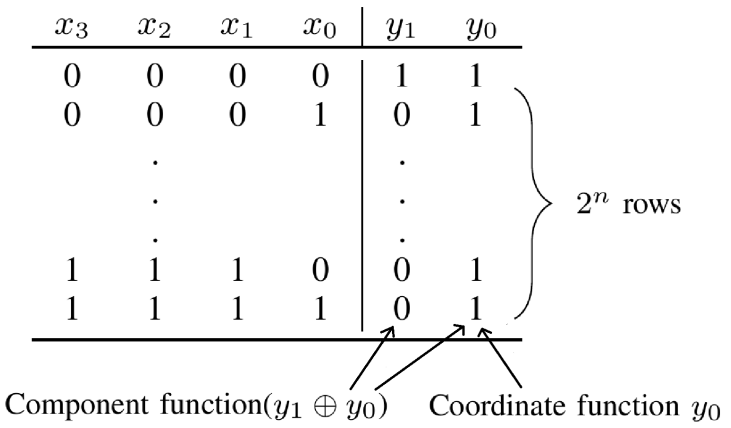


Figure 4.1: An example of $(4,2)$ function.

Additionally, we examine whether heuristics, more precisely evolutionary algorithms, can be used to find new functions fulfilling such criteria or at least bring new insights about the problem. To this end, we experiment with five representations of solutions and four problem sizes. Our experiments show that some of the representations we use can find differentially 6-uniform functions in dimensions $(4,2)$ and $(5,3)$. Although such functions were known before, we still consider our results relevant. The lack of success for higher dimensions coupled with the unsuccessful results from algebraic constructions could serve as a strong indicator that differentially 6-uniform functions do not exist in dimension $(6,4)$. This study has been disseminated in [109].

4.2 Background

Let n,m be positive integers, i.e., $n,m \in \mathbb{N}^+$. We denote by $\mathbb{F}_2^n$ the n-dimensional vector over $\mathbb{F}_2$ and by $\mathbb{F}_{2^n}$ the finite field with 2^n elements. The set of all n-tuples of elements in the field $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the Galois field with two elements. Further, for any set S, we denote $S\backslash\{0\}$ by S^*. The usual inner product of a and b equals $a \cdot b = \bigoplus_{i=1}^{n} a_i b_i$ in F_2^n. The Hamming weight (w_H) of a vector a, where $a \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector. An (n,m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. An (n,m)-function F can be defined as a vector $F = (f_1, \cdots, f_m)$, where the Boolean functions $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $i \in \{1, \cdots, m\}$ are called the coordinate functions of F. The component functions of an (n,m)-function F are all the linear combinations of the coordinate functions with non all-zero coefficients. In the rest of the section, we use a capital letter F when discussing vectorial Boolean functions and a small letter f when discussing Boolean functions. We give a small example of a $(4,2)$ function F in Figure 4.1 where one can see the difference between the coordinate and component function.

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), i.e., a vector $(f(0), ..., f(1))$ that contains the function values of f, ordered lexicographically [43].

The second unique representation of a Boolean function is the Walsh-Hadamard transform

W_f that measures the correlation between $f(x)$ and all linear functions $a \cdot x$ [43, 64]. The Walsh-Hadamard transform of a Boolean function f equals:

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}. \tag{4.1}$$

The Walsh-Hadamard transform of an (n,m)-function F is a set of values [44]:

$$W_F(a,v) = \sum_{x \in \mathbb{F}_2^m} (-1)^{v \cdot F(x) \oplus a \cdot x},\ a, v \in \mathbb{F}_2^m. \tag{4.2}$$

Algorithm 6 presents the Walsh-Hadamard transform of (n,m) Boolean function.

Algorithm 6 Walsh-Hadamard transform of (n,m) function.

Require: x is an array of 2^n Boolean values, and m is the number of outputs
Ensure: y is the Walsh-Hadamard transform of x for m outputs

```
y ← x
for k = 1 to m do
  for i = 1 to n do
    m ← 2^(i−1)
    for j = 0 to 2^n − 1 do
      if (j mod 2^i) < 2^(i−1) then
        s ← y[j][k] + y[j+m][k]
        t ← y[j][k] − y[j+m][k]
        y[j][k] ← s
        y[j+m][k] ← t
      end if
    end for
  end for
end for
return y
```

A Boolean function with n inputs is balanced if its Hamming weight equals 2^{n-1}. An (n,m)-function F is balanced if it takes every value of $\mathbb{F}_2^m$ the same 2^{n-m} number of times.

A Boolean function f should lie at a large Hamming distance (HD) from all affine functions and the nonlinearity N_f of a Boolean function is the minimum Hamming distance between the function f and affine functions [43]. The nonlinearity N_f of a Boolean function f expressed in terms of the Walsh-Hadamard coefficients equals [43]:

$$N_f = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |W_f(a)|. \tag{4.3}$$

The nonlinearity N_F of an (n,m)-function F equals the minimum nonlinearity of all its

component functions $v \cdot F$, where $v \in \mathbb{F}_2^{m*}$ [44]:

$$N_F = 2^{n-1} - \frac{1}{2} \max_{\substack{a \in \mathbb{F}_2^n \\ v \in \mathbb{F}_2^{m*}}} |W_F(a,v)|. \tag{4.4}$$

The Parseval's relation equals:

$$\sum_{a \in \mathbb{F}_2^n} W_f(a)^2 = 2^{2n}, \tag{4.5}$$

and it implies that the mean of $W_f(a)^2$ equals 2^n, and $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$ is then at least equal to the square root of this mean.

From Eq. (4.5), it follows that the maximal value of the Walsh-Hadamard spectrum equals at least $2^{\frac{n}{2}}$, which occurs with equality in the case of bent Boolean functions. From this equation we see that the nonlinearity of any Boolean function is less or equal to:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{4.6}$$

This bound is called the covering radius bound and is strict for bent Boolean functions. Furthermore, since this bound is valid for any Boolean function it is even more valid for vectorial Boolean functions. Therefore, in order for an (n,m) function to be bent, all of the component functions $v \cdot F, v \neq 0$ of a function F must be bent. Since bent n-dimensional Boolean functions can exist only when n is even, then bent (n,m) functions can exist only when n is even. However, this condition has been shown not to be sufficient. K. Nyberg showed that bent (n,m) functions can exist only when $m \leq \frac{n}{2}$ [110]. Bent (n,m) functions are also called perfect nonlinear functions [44].

Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ with $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$. We denote:

$$D_F(a,b) = \{x \in \mathbb{F}_2^n : F(x) + F(x+a) = b\}. \tag{4.7}$$

The entry at the position (a,b) corresponds to the cardinality of the difference distribution table $D_F(a,b)$ and is denoted as $\delta(a,b)$. The *differential uniformity* δ_F is then defined as [110]:

$$\delta_F = \max_{a \neq 0, b} \delta(a,b). \tag{4.8}$$

Differential uniformity is always an even number since if x is a solution of the equation $F(x)+F(x+a)=b$ when $a \neq 0$, then also $x+a$ must be a solution. Kaisa Nyberg observed that $\delta_F \geq 2^{n-m}$ only if $n > m$, and $\delta_F \geq 2$ only if $n \geq m$ [110]. Actually, she proved that $\delta_F = 2^{n-m}$ if and only if n is even and $m \leq \frac{n}{2}$. For $n \geq 5$, Nyberg proved that δ_F is bounded below by 6

for $(n, n-2)$ functions. Differentially 8-uniform $(n, n-2)$ functions are easily constructed by composing on the left any APN function by a surjective affine $(n, n-2)$ function. Still, we do not know whether it is actually possible to construct a differentially 6-uniform $(n, n-2)$ when n is larger than 5.

For further information on vectorial Boolean functions and their applications in cryptography, we refer interested readers to [43, 44].

4.3 Related Work

As stated, evolutionary algorithms have many successful applications when constructing vectorial Boolean functions.

Clark et al. use the principles from the evolutionary design of Boolean functions to evolve S-boxes with desired cryptographic properties. They use simulated annealing (SA) and hill climbing algorithm to evolve bijective S-boxes of sizes up to 8×8 with high nonlinearity values [111].

Millan et al. work with genetic algorithms to evolve S-boxes with high nonlinearity and low autocorrelation value. Moreover, the authors discuss the selection of the appropriate genetic algorithm parameters [112].

P. Tesar uses a special genetic algorithm with a total tree searching to generate 8×8 S-boxes with nonlinearity equal to 104 [113].

Picek et al. explore how to generate S-boxes of size 8×8 with better resistance against side-channel attacks as measured with the transparency order and modified transparency order properties [114, 115].

Finally, Picek, Rotim, and Cupic develop a new cost function able to reach high nonlinearity values for many different S-box sizes [116]. This fitness function eliminated the need for several parameters that usually required an extensive tuning phase.

When considering algebraic constructions, results show bounds for differential uniformity for $(n, n-2)$ functions [110]. Carlet and Alsalami are able to construct $(n, n-1)$ functions that are differentially 4-uniform [103]. They also use the same construction to find $(5, 3)$ functions that are differentially 6-uniform. Unfortunately, their construction does not generalize to higher dimensions, and they cannot find any differentially 6-uniform function for dimension $(6, 4)$. De Meyer and Bilgin conducted an exhaustive search of all quadratic $(6, 4)$ functions [117]. Their results show there are no $(6, 4)$ differentially 6-uniform functions with an algebraic degree equal to 2.

When considering heuristics, most of the work on S-boxes concentrates on bijective S-boxes. Additionally, the primary goal in these papers is usually the nonlinearity property, and only seldom do researchers consider differential uniformity.

Table 4.1: The search space size for the investigated sizes of functions.

m \ n	4	6	8	10	12
1	2^{16}	2^{64}	2^{256}	2^{1024}	2^{4096}
2	2^{32}	2^{128}	2^{512}	2^{2048}	2^{8192}
3	-	2^{192}	2^{756}	2^{3072}	2^{12288}
4	-	-	2^{1024}	2^{4096}	2^{16384}
5	-	-	-	2^{5120}	2^{20480}
6	-	-	-	-	2^{24576}

Table 4.2: Nonlinearity of a bent function of n variables.

n	4	6	8	10	12
Nonlinearity	6	28	120	496	2 106

Burnett et al. used a heuristic method to generate MARS-like S-boxes [118]. They can generate several S-boxes of appropriate size that satisfy all the requirements placed on the MARS S-box [119] and they even manage to find S-boxes with improved nonlinearity values.

Picek, Knezevic, and Jakobovic used evolutionary computation to evolve bent (n,m)-functions but do not consider differential uniformity [83].

4.4 Experiments and Results

4.4.1 Bent (n,m) functions

Before going into discussion on experimental setting and presenting results, we briefly discuss the difficulty of the problem we consider. A Boolean function f of n variables can be represented with a string of 2^n values and the search space size is equal to 2^{2^n}, as given in Table 4.1 for several sizes of interest. Note that to calculate nonlinearity and the Walsh-Hadamard spectrum, we always need to transform our solutions into the truth table representation. Next, in Table 4.2 we give optimal values for the nonlinearity property and each size of Boolean functions we consider in this section. Note that the component-wise nonlinearity for vectorial Boolean function needs to be the same as for a single Boolean function.

Algorithms and Representations

We experiment with three different representations for encoding of a (vectorial) Boolean function: bitstring, floating-point, and tree representation. The simplest genotype to use is the bitstring since in that case there is no need for mapping between it and the truth table representation of a Boolean function. However, as given in Table 4.1 we can observe that the genotype size increases exponentially with the number of variables of a Boolean function. Moreover, this number needs to be multiplied by the number of output variables when $m > 1$.

In the second encoding we use the floating-point genotype, which is defined as a vector of continuous variables. Therefore, once a solution is obtained, the first step is to transform it into the truth table representation of a Boolean function. Here, we use each continuous variable to decode a subset of bits from the truth table. In this way, we can compress the size of a solution, i.e., the number of genes we need to represent a solution. The transformation between real values and the bitstring is done as follows. First, we enforce that all real values are in the range $[0,1]$. Then, the number of bits that are represented with a single floating-point value, $decode\_by$, can vary:

$$decode\_by = \frac{2^n}{dimension}, \tag{4.9}$$

where the parameter $dimension$ denotes the floating-point vector size (number of real values). Note that this parameter can be modified as long as the size of the truth table is divisible with it, so that each real value represents the same number of bits.

In the process of transformation of real values we first convert the floating-point values into integers. Each floating-point value falls into a specific *interval* between 0 and 1. Since each real value must represent $decode\_by$ bits, the size of that interval is given as:

$$interval = \frac{1}{2^{decode\_by}}. \tag{4.10}$$

For instance, if each real value encodes 2 bits from the truth table, the interval size is 0.25. To obtain a distinct integer for a given floating-point value, every element d_i of the floating-point vector is divided by the calculated interval size, generating a sequence of integers:

$$int\_value_i = \left\lfloor \frac{d_i}{interval} \right\rfloor. \tag{4.11}$$

In the example, the corresponding integer value is obtained by dividing the real value with $2^{-2} = 0.25$ and truncating to nearest smaller integer. The second step is to decode the integer values into a binary sequence to be used in evaluation. Here, we opted to use the Gray encoding, due to the proportional solution distance in both floating-point and binary search space.

For the algorithm we use a simple GA with the tournament selection where its size equals 3 [89] and the population size is 100. The mutation is selected uniformly at random between a

simple mutation, where a single bit is inverted, and a mixed mutation, which randomly shuffles the bits in a randomly selected subset. The crossover operators are one-point and uniform crossover, performed at random for each new offspring.

As the third encoding, we use tree-based GP in which a Boolean function is represented by a tree of nodes [90]. The function set for GP in all the experiments is OR, NOT, XOR, AND, XNOR, AND with one input inverted and IF, which takes three arguments and returns the second one if the first evaluates to true, and the third one otherwise. The terminals correspond to n Boolean variables. Note that when $m > 1$, we actually use m independent trees to represent a solution. GP also uses tournament selection with the tournament size equal to 3.

The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point, and context preserving crossover [91]. We use a single mutation type, a subtree mutation, and use maximum tree depth of 5. The population size for the GP equals 200.

Common Parameters

In all the experiments the number of independent trials N for each configuration is 30 and the stopping criterion for all algorithms equals 500 000 evaluations. For each of the algorithms and representations we use the individual mutation probability of 0.5. It is important to note that we use the mutation probability to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual; e.g. if the mutation probability is 0.5, then on average 5 out of every 10 new individuals will be mutated (see Algorithm 3), and one mutation will be performed on that individual.

Fitness Functions

In the next section, we present fitness functions we consider in our experiments. Note that the first two functions are well established and used in related work while the third fitness function is, as far as we know, explored for the first time in the evolution of vectorial Boolean functions.

Fitness Function 1 In the simplest version of the fitness function we aim to maximize the nonlinearity value:

$$fitness_1 = N_F. \tag{4.12}$$

Fitness Function 2 The second version of fitness function improves on Eq. (4.12) and adds the second term where we aim to minimize the number of occurrences of values different from $2^{\frac{n}{2}}$ as given by the Walsh-Hadamard spectrum. Note that this is a version of the fitness that usually gives better results for vectorial Boolean functions where the input and output dimension

are the same (however, there it is mostly used with the permutation encoding). Here we are again interested in the maximization of the following expression:

$$fitness_2 = N_F + \frac{1}{T}, \tag{4.13}$$

where T equals the number of occurrences of the value different from $2^{\frac{n}{2}}$ in the Walsh-Hadamard spectrum. Since we still consider nonlinearity as the primary parameter, we scale the second term in the fitness function in the range $[\frac{1}{2^n}, 1]$. Note that if $T = 0$ then the fraction is set to 1.

Fitness Function 3 Finally, in the third fitness function we try to explore the relation previously not considered when evolving bent (vectorial) Boolean functions. A Boolean function is bent if all its derivatives are balanced [43]. Here, a derivative D of a Boolean function f in the direction of b equals:

$$D_b f(x) = f(x) \oplus f(x \oplus b). \tag{4.14}$$

For a vectorial Boolean function to be bent, all its linear components need to be balanced and therefore we aim to maximize the following expression:

$$fitness_3 = \sum_{i=1}^{2^n} \sum_{b=1}^{2^n} D_b f(x_i), \tag{4.15}$$

where x_i represents the value of a function f for input x at position i.

Results

We give the results in the *max value/median value* notation since the first number has greater relevance from the practical perspective. In comparison, the second number has more relevance from the optimization perspective. To assess what is the average behavior of the algorithms, we tested. Note that we opted again to use the median rather than the average value so as not to assume a normal data distribution.

In Table 4.3 we give the results for tree encoding and fitness function 1. We see that when $m = 1$ (single Boolean function case) GP is able to find optimal results for all tested input size in 100% of cases. Moreover, GP is able to reach optimal values in all the runs for when considering input dimension of 6 variables. On the other hand, starting with $n = 8$, GP is not able to reach optimal value when the output dimension is equal or greater than 3.

Next, in Table 4.4 we give the results for the bitstring representation and fitness function 1. Note that here, except for the smallest size (i.e., when $n = 4$), we cannot reach the optimal values. Therefore, our results corresponds to the ones from related work for cases when $m = 1$ or $n = m$. On the other hand, for dimensions $(10,5),(12,5),(12,6)$ bitstring representation outperforms the tree representation.

Table 4.3: Results for the tree representation, fitness 1.

m \ n	4	6	8	10	12
1	6/6	28/28	120/120	496/496	2 016/2 016
2	6/6	28/28	120/120	496/496	2 016/1 984
3	-	28/28	120/106.5	480/463	1 984/1 920
4	-	-	112/104	480/448	1 944/1 878
5	-	-	-	448/447	1 920/1 792
6	-	-	-	-	1 888/1 765

Table 4.4: Results for bitstring representation, fitness 1.

m \ n	4	6	8	10	12
1	6/6	28/26	114/113.5	478/476	1 964/1 961
2	6/6	26/25	112/110	472/470	1 956/1 952
3	-	24/24	110/108	468/466	1 948/1 946
4	-	-	108/106	464/464	1 944/1 940
5	-	-	-	462/460	1 936/1 934
6	-	-	-	-	1 930/1 929

Table 4.5: Results for the floating-point representation, fitness 1.

m \ n	4	6	8	10	12
1	6/6	26/26	112/112	476/475	1 950/1 936
2	6/6	26/24	110/109	472/470	1 934/1 920.5
3	-	24/24	108/106	468/466	1 924/1 912
4	-	-	106/104	464/464	1 918/1 900.5
5	-	-	-	462/460	1 914/1 895
6	-	-	-	-	1 912/1 893

Table 4.6: Results for the tree representation, fitness 2.

m \ n	4	6	8	10	12
1	6/6	28/28	120/120	496/496	2 016/2 016
2	6/4	28/28	120/120	496/480	2 016/1 984
3	-	28/24	120/106.5	480/460	2 016/1 920
4	-	-	112/104	480/448	1 920/1 856
5	-	-	-	448/440	1 920/1 792
6	-	-	-	-	1 856/1 768

Finally, in Table 4.5 we give results for the fitness function 1 and floating-point representation. As we can see, the results are somewhat worse than for the bitstring encoding. Such results are also in accordance with the results from [120] where the authors note that floating-point representation is in the most of the cases the worst one.

In the second scenario we improve our fitness function to give more information and consequently to reach better values. We emphasize that for the fitness function 2, we give in tables the nonlinearity values and not the fitness values (with the decimal part from the second term in Eq. (4.13)). Surprisingly, we can observe that in a number of scenarios the median value is even lower than for the fitness function 1 (cf. Table 4.3). However, we note that with the fitness function 2 we are able to reach optimal value for the $(12, 3)$ case which we could not reach with any representation and fitness function 1.

In Table 4.7 we give results for the fitness function 2 with the bitstring encoding. There, for the dimension $(6, 1)$ the results are slightly worse when compared to the fitness function 1,

Table 4.7: Results for the bitstring representation, fitness 2.

m \ n	4	6	8	10	12
1	6/6	26/26	114/114	478/476	1 968/1 962
2	6/6	26/25.5	114/110	472/470	1 956/1 952.5
3	-	24/24	110/108	468/466	1 948/1 946
4	-	-	108/106	466/464	1 944/1 940
5	-	-	-	462/460	1 938/1 934
6	-	-	-	-	1 932/1 928.5

Table 4.8: Results for the floating-point representation, fitness 2.

m \ n	4	6	8	10	12
1	6/6	26/26	112/111	477/476	1 952/1 936
2	6/6	26/24	110/108	472/470	1 928/1 913
3	-	24/24	108/106	468/466	1 968/1 910
4	-	-	106/104	464/464	1 922/1 906
5	-	-	-	462/460	1 918/1 898.5
6	-	-	-	-	1 906/1 898

but for $(8,2),(10,4),(12,1),(12,5),(12,6)$ dimensions, the fitness value improves although it does not reach optimal values. What is interesting to notice is that the bitstring representation outperforms the tree encoding for sizes $(10,5),(12,4),(12,5),(12,6)$ which is similar situation as with the fitness function 1. On the other hand, smaller dimensions reach higher fitness values when the tree encoding is used.

When working with the floating-point encoding, we can observe that the improved fitness function does not help and actually for a number of scenarios we reach even worse values when compared to the fitness function 1. Moreover, the results with the floating-point encoding and fitness function 2 are the worst ones when compared to the other encodings.

In Figures 4.2a until 4.2f we give statistics in the boxplot form for fitness functions 1 and 2 and for function input sizes $n = 8,10,12$. To denote the scenarios we use the notation: input size – output size encoding (T for tree, B for bitstring, and F for the floating-point encoding). We also use a color coding where the boxplot in red color represents the tree encoding, in green color the bitstring encoding, and finally in blue color the floating-point encoding. From the

Table 4.9: Theoretical maximal values for fitness 3.

m \ n	4	6	8
1	15	63	255
2	45	189	765
3	-	441	1 785
4	-	-	3 825

Table 4.10: Results for the tree representation, fitness 3.

m \ n	4	6	8
1	15/15	63/63	255/255
2	45/45	189/189	765/765
3	-	438/435	1 779/1 776
4	-	-	3 813/3 795

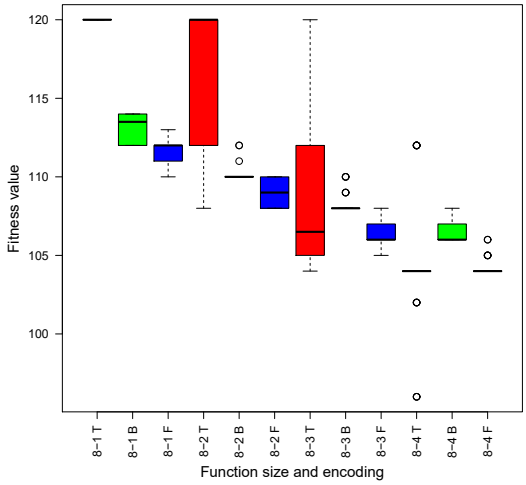
graphs it is easy to see that when the output dimension is small (e.g., $m = 1$), the tree encoding is by far superior when compared to the other two encodings. Furthermore, for input sizes 10 and 12, tree encoding performs the best on average with a clear advantage over other encodings while for the input dimension 12 the results are much more similar over all encodings (especially when $m > 1$).

Next, we present results when experimenting with the fitness function where the objective is to obtain all balanced derivatives of linear combinations of a vectorial Boolean function. First, in Table 4.9 we give the theoretical best results. Note that here the higher the value, the more derivatives are balanced. In Table 4.10 we give the results for tree encoding and input dimension in the range $[4,8]$ and the output dimension in the range $[1,4]$. We do not display the results for larger sizes nor for the other encodings since the results obtained are inferior when compared to the first two fitness functions.

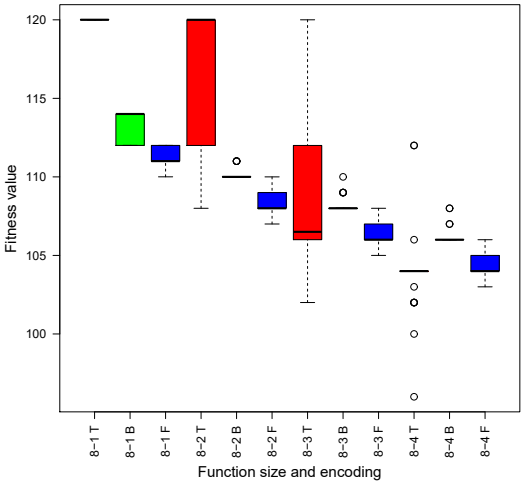
4.4.2 Differentially-6 Uniform $(n, n-2)$ Functions

Solution Representation

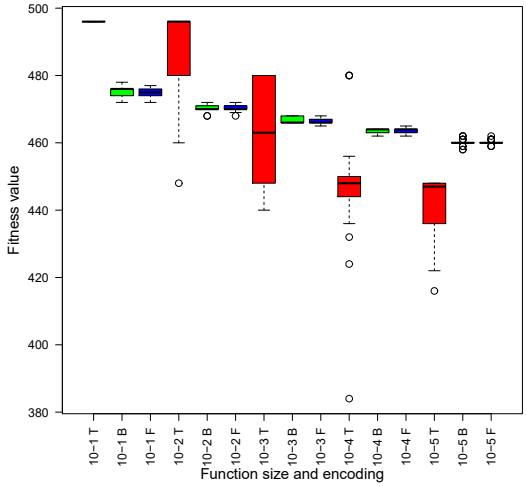
We used several representations to obtain more reliable results. Some of the representations, such as genetic programming, integer and floating-point, cover all of the possible search space.



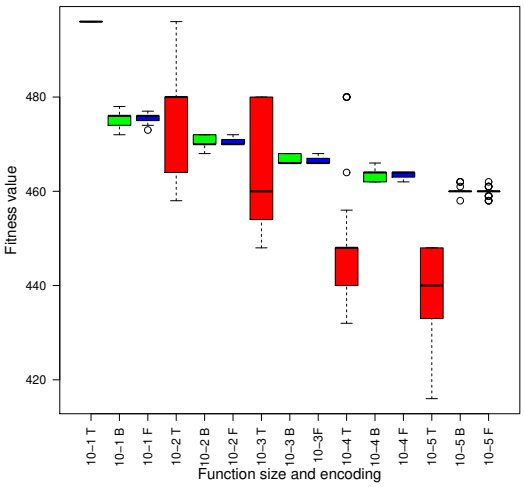
(a) Fitness function 1, $n = 8$



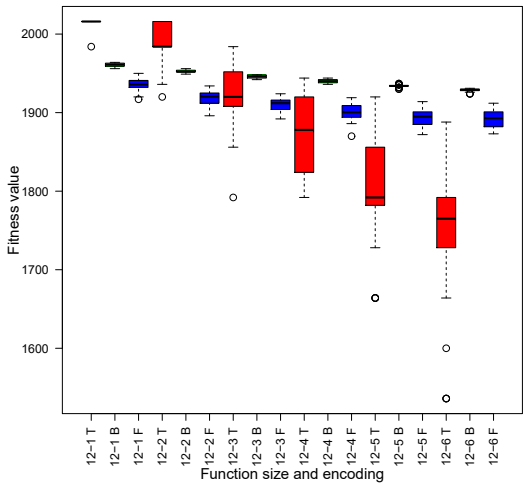
(b) Fitness function 2, $n = 8$



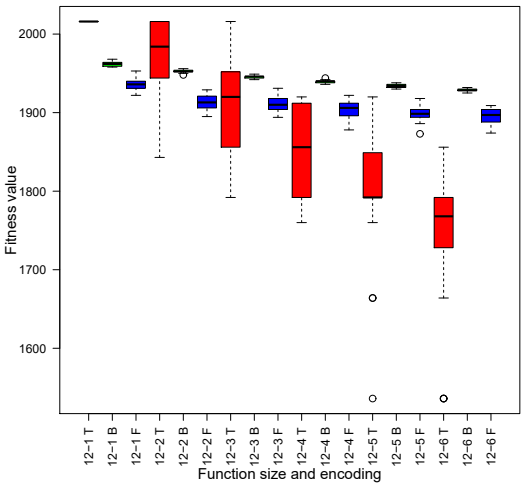
(c) Fitness function 1, $n = 10$



(d) Fitness function 2, $n = 10$



(e) Fitness function 1, $n = 12$



(f) Fitness function 2, $n = 12$

Additionally, we employ representations that only map to a portion of the search space, such as permutation based, in the hope of increasing the probability of finding the optimal solution. We opted to use multiple genetic operators with every encoding since our previous experience showed such a strategy giving the best results. Additionally, since there are no previous works considering the problem of finding differentially 6-uniform $(n, n-2)$ functions, we had no point of reference to select a single best operator.

Genetic Programming - Tree Encoding As the first encoding, we use tree-based genetic programming (GP) with the same algorithm setting described in this chapter's previous experiment. We use $n-2$ independent trees in an individual to represent a single solution.

Integer Encoding In this representation, we use the truth table of the underlying $(n, n-2)$ function; the truth table is represented with an integer array of length 2^n with each element in the range $[0, 2^{n-2}-1]$. Before the fitness calculation, it can be easily transformed into the binary form and evaluated accordingly. In this encoding, the mutation and crossover operators are based on their binary counterparts: the mutation selects a random gene and modifies its value. The crossover operators are single-point and two-point crossover, which only concatenate parts of different individuals. The average crossover for which the child's genes assume mean values of parents' genes.

Permutation Encoding To restrict the search space and make it easier for the algorithm to form a solution, permutation encoding is employed rather than an array of arbitrary values. In this representation, we use the permutation array of size 2^n with elements in the range $[0, 2^n - 1]$ where each value occurs precisely once. Before evaluation, we decode it into a $(n, n-2)$ function by truncating each value with modulo 2^{n-2}. It will represent a $(n, n-2)$ function truth table. For instance, an example input permutation of size 2^n with beginning elements $[7, 0, 9, 11, 2, \ldots]$ will be transformed into $[3, 0, 1, 3, 2, \ldots]$. After that, the individual is evaluated as in the previous encoding. The permutation genotype uses the OX, PBX, PMX, and cyclic crossover, whereas the mutation operators are inserted, inverse, and toggle. For each crossover and mutation, a single operator is chosen at random.

Quadruple Permutation The permutation encoding poses constraints on the search space size, but for larger function sizes, it is still tremendous. Consequently, we limit it further by employing the representation in which each individual is encoded with four permutations, each of size 2^{n-2}. This encoding uses the same genetic operators as the previous one. The total length of the individual genotype is the same as in the previous case. Still, the crossover and mutation operators are performed separately on each containing four permutations. By default, an individual will have only one of them affected with mutation or crossover.

Floating-point Encoding Finally, we use the floating-point encoding in the following manner: an individual is an array of floating-point values in the range $[0,1]$. Each floating-point value represents one or more integer values in the range $[0,2^{n-2}-1]$ as in the integer encoding. Based on the number of different integer values (2^n) and the number integers per single floating-point (k), the range $[0,1]$ is divided into $(2^n)^k$ intervals. Depending on the interval in which each floating-point value falls, we decode k elements of the truth table from a single floating-point variable. In our experiments, we use $k = 1,2,4$ so the size of the floating-point array is either the same (for $k = 1$), two times, or four times smaller than the corresponding integer array size. For instance, if $k = 2$ and we are searching for the $(4,2)$ function, the individual is an array of 8 floating-point values where each gene presents two integer values. If the first few floating-point numbers are $[0.54,0.41,0.96,\ldots]$ they will get decoded into integer array $[2,0,1,2,3,3,\ldots]$ of size 16. This way, we can use a wide variety of floating-point-based crossover and mutation operators, which usually are not applicable in the discrete domain. In our experiments, we used the arithmetic, heuristic, average, one point, and SBX crossover; the mutation consists of a single operator which changes a randomly selected gene.

Considering the expressiveness of the above representations, we note that the GP, integer, and floating-point array allow mapping to all the possible solutions, while the permutation and quadruple permutation map only a portion of the search space.

Fitness Function

In order to obtain the target differential uniformity value, the evaluation of each potential solution includes only the calculation of the differential uniformity property. Based on that, the fitness function to be minimized is simply the absolute distance from the desired value of 6:

$$fitness_1 = |6 - \delta_F|. \tag{4.16}$$

We noticed that when using $fitness_1$, the used algorithms had difficulties in finding the optimal solutions. Since the values in the difference distribution table (Eq. (4.7)) depend on each other, looking only at the maximal value can be counterintuitive. Indeed, minimizing the maximal value can easily result in increasing some other value in the table. Consequently, we also conducted experiments with the second fitness function defined as:

$$fitness_2 = \sum_{a \in \mathbb{F}_2^n} \sum_{b \in \mathbb{F}_2^m} |6 - \delta(a,b)|. \tag{4.17}$$

The motivation for the fitness function defined as in Eq. (4.17) was to minimize the number of values different from 6 occurring in the difference distribution table, or alternatively, to make as much as possible values close to 6.

Table 4.11: Parameters for GP representation

Parameter	Initial value	Tested values	Final value
Max tree depth	4	{4, 5, 6}	**5**
Population size	200	{100, 200, 300}	200

Table 4.12: Parameters for quadruple permutation

Parameter	Initial value	Tested values	Final value
Population size	200	{100, 200, 500}	200
Mutation rate	0.5	{0.1, 0.3, 0.5, 0.7, 0.9}	**0.9**

Algorithm and Parameters

Regardless of the representation, all the variants use the same evolutionary algorithm, which is a steady-state process presented in Algorithm 3.

In all the experiments, the number of independent trials for each configuration is 30 and the stopping criterion for all algorithms equals 10^6 evaluations or reaching the differential uniformity equal to 6. In order to evaluate the differences between different representations, we conducted a tuning phase. For each encoding, we selected the most influential parameters (based on our previous experience) and run the tests with different parameter values on the problem size of $(5,3)$. Unfortunately, the problem at hand provided no clear guidelines, since in most cases all the experiments converge to the same value in all the repetitions. In other words, for most of the configurations there were no statistically significant differences between different parameter values. In cases where the difference was not visible we have kept the initial parameter values, whereas the different final parameter values are shown in boldface. The results of the tuning phase are shown in Tables 4.11 till 4.15.

The tuning phase was conducted with both $fitness_1$ and $fitness_2$ functions. The results showed that $fitness_1$ significantly outperforms $fitness_2$ and we consequently present the results for only the first fitness function.

Table 4.13: Parameters for permutation encoding

Parameter	Initial value	Tested values	Final value
Population size	200	{100, 200, 500}	200
Mutation rate	0.5	{0.1, 0.3, 0.5, 0.7, 0.9}	0.5

Table 4.14: Parameters for integer encoding

Parameter	Initial value	Tested values	Final value
Population size	200	{100, 200, 500}	200
Mutation rate	0.5	{0.1, 0.3, 0.5, 0.7, 0.9}	0.5

Table 4.15: Parameters for floating-point encoding

Parameter	Initial value	Tested values	Final value
Population size	50	{50, 100, 200}	50
Mutation rate	0.5	{0.3, 0.5, 0.7, 0.9}	0.5
Integers encoded with single FP value	1	{1, 2, 4}	**2**

Results

In total, the search space size of possible (n,m)-functions equals 2^{m2^n}. When considering $(n,n-2)$ functions, when $n=4$ the search space size equals 2^{32}, which is possible to exhaustively search. Already for $(5,3)$ functions, the search space size equals 2^{96}, which is far beyond current computing capabilities.

Exhaustive Search in $(4,2)$ Dimension

We conduct the exhaustive search in $(4,2)$ dimension to investigate the properties of the functions with differential uniformity equal to 6. We find 0.38% of functions with such differential uniformity. Only 1.367% of those functions have the same number of occurrences of each value (e.g., 4 values 0, 4 values 1, etc.). The distribution of S-box values is uniform, i.e., we notice every possible value on every possible position. Next, we investigate the number of functions in which, at least, one value is missing. There are 1.366% of functions with that property. When considering a single value repeating itself a number of times consecutively, we find the longest such subset to be of length 6. Finally, when considering the results that could be obtained with the quadruple permutation encoding, out of 0.38% solutions with differential uniformity equal to 6, there are 6.70% that can be represented with quadruple permutations.

Heuristic Results

We conduct experiments with 5 representations and 4 different problem sizes: $(4,2)$, $(5,3)$, $(6,4)$, and $(7,5)$. The performances of different representations are shown as box-plots for each

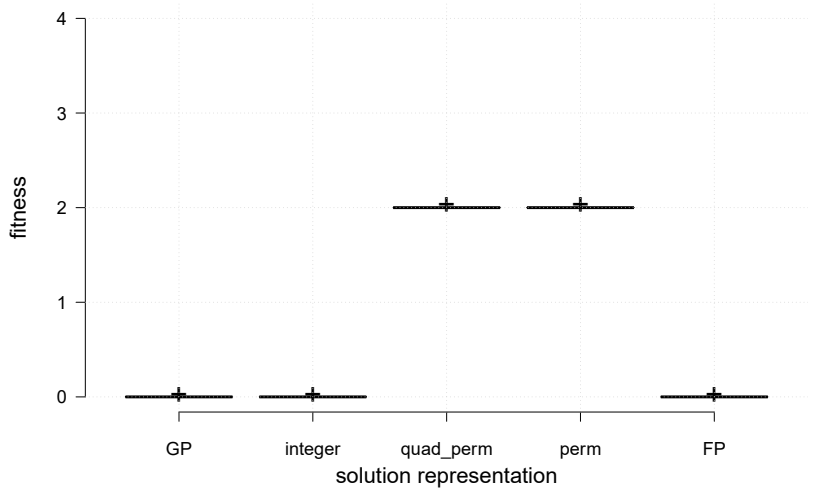


Figure 4.3: Results for $(4,2)$ S-box size

Table 4.16: Results for $(5,3)$ size and quadruple permutation encoding.

Min	Max	Average	Std dev
0	2	1.68	0.74

problem size in Figures 4.3 until 4.6. The combinations in which we were able to obtain the optimal solution (with differential uniformity 6) are denoted with the fitness value of zero.

In the most simple $(4,2)$ case, the GP, integer, and floating-point representations are always able to reach the optimal solution (see Figure 4.3). On the other hand, the permutation and quadruple permutation encoding were unsuccessful in every run. This is an interesting behavior since we know from the exhaustive search it is possible to obtain differentially 6-uniform functions encoded with quadruple permutations. Continuing to that fact, it is intriguing that only the quadruple permutation encoding succeeded in obtaining the optimal solution for the $(5,3)$ size.

When considering $(5,3)$ dimension as given in Figure 4.4, we see that GP encoding works the worst, while quadruple permutation is the only one reaching the global maximum. Floating-point, integer, and permutation encoding all reach the same value and there is no statistically significant difference in their behavior.

In Table 4.16, we give results for $(5,3)$ size and quadruple permutation representation, with population size equal to 200 and mutation of 0.9.

For the $(6,4)$ size we depict results in Figure 4.5. Here, the integer encoding is the only one reaching differential uniformity equal to 8, while all the other encodings are stuck on differential

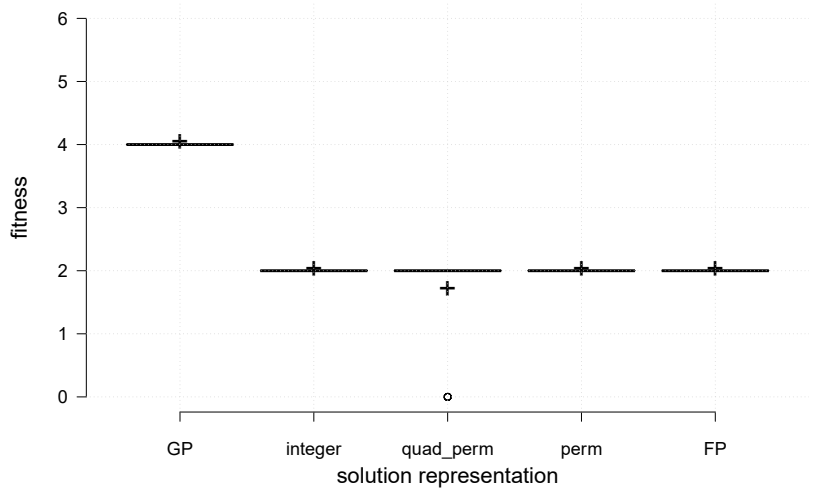


Figure 4.4: Results for $(5,3)$ S-box size

uniformity of 10.

When examining $(7,5)$ dimension, we observe even worse results than in the previous cases. This behavior is somewhat expected since even for the smaller dimension we could not find optimal solutions. Interestingly, again integer encoding gives the best results with differential uniformity going down to 10. Quadruple permutation, permutation, and floating-point behave the same and reach differential uniformity of 12. Finally, GP encoding works the worst with differential uniformity solutions in the range $[12,14]$. Figure 4.7 gives the convergence results for the $(7,5)$ dimension when using the permutation encoding where we depict the averaged Min, Max, and Average values over all experimental runs. As it can be seen, despite having relatively rough grained fitness values, the evolution process still needs more than 150 000 evaluations to find the best solution and more than 800 000 evaluations before is starts stagnating.

The best obtained solutions over all the experiments are then given in Table 4.17. When a certain encoding is able to reach the global optimum of 0, we depict it in bold style.

The relation between the representations and the obtained differentially 6-uniform solutions is illustrated in Figure 4.8. Note that the sizes are not given in the actual ratio.

We can observe that evolving differentially 6-uniform $(n,n-2)$ functions is an extremely difficult problem when $n > 5$. What is more, this could be impossible since we have no proofs that such functions even exist. Unfortunately, as is the case with all heuristic techniques, failure to produce optimal results does not mean there are no such results. From that perspective, our experiments did not yield any new information since we still do not know such functions nor do they exist. Since the differential uniformity property can have only even values (i.e., it changes

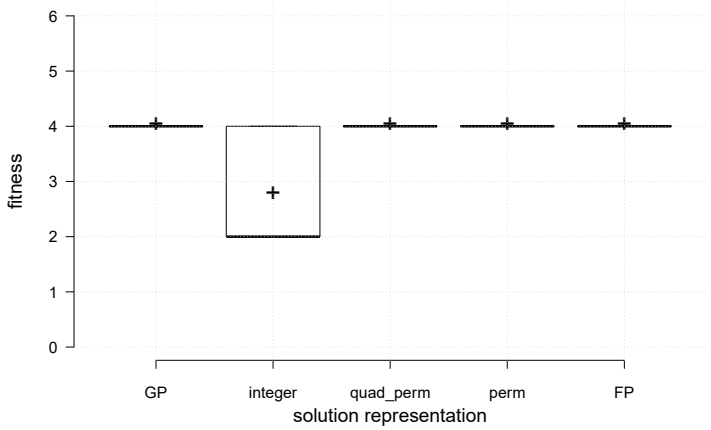


Figure 4.5: Results for $(6,4)$ S-box size

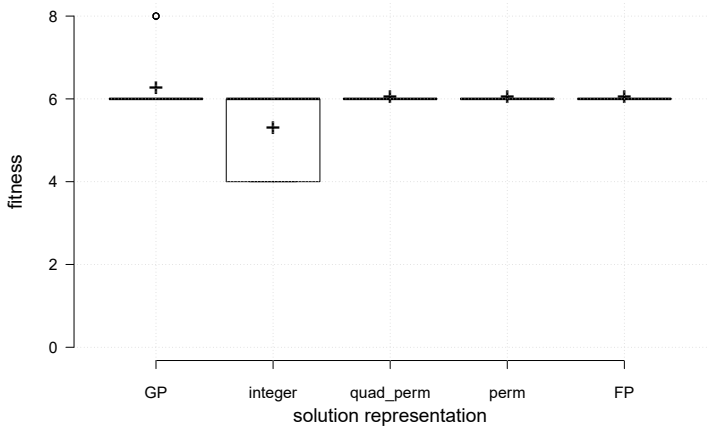


Figure 4.6: Results for $(7,5)$ S-box size

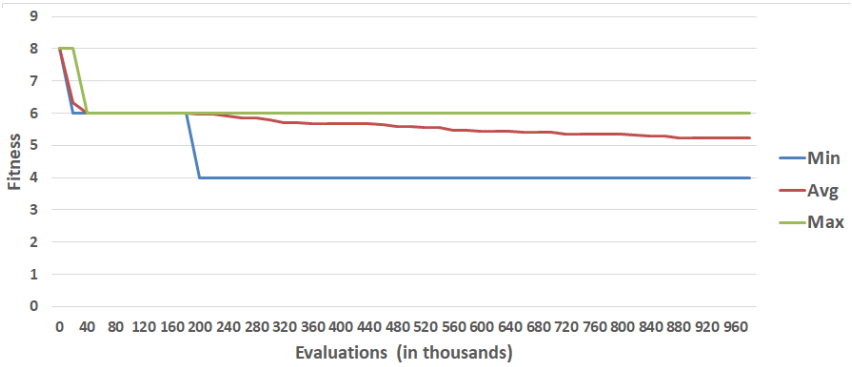


Figure 4.7: Convergence for $(7,5)$ dimension with permutation encoding

Table 4.17: Best obtained results for all encodings

Encoding \S-box size	(4,2)	(5,3)	(6,4)	(7,5)
Genetic programming	**0**	4	4	6
Permutation	2	2	4	6
Quadruple perm.	2	**0**	4	6
Integer array	**0**	2	2	4
Floating-point array	**0**	2	4	6

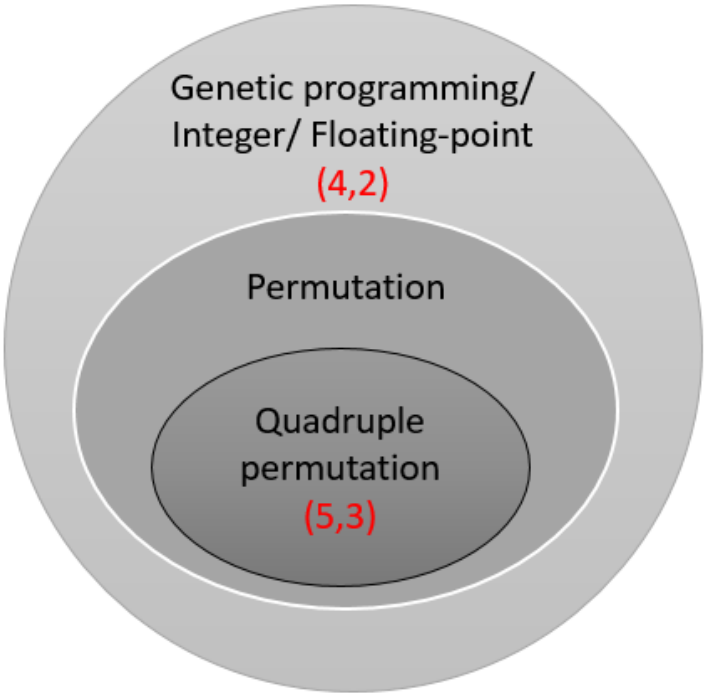


Figure 4.8: Illustration of the mapping between representations and problem space

in jumps of two), it is easy to observe that there is not much gradient that could lead the search process.

Still, for the $(5,3)$ S-box size, we are able to find a number of differentially 6-uniform functions. Since the corpus of currently known such functions is very limited, it could be expected that at least some of the found functions are new (i.e., not equivalent to the previously known functions).

4.5 Conclusions

In this chapter, we investigate the evolution of bent vectorial Boolean functions. Differing from the case when considering single Boolean functions (i.e., where the number of outputs equals 1) or S-boxes as used in block ciphers (where the input and output dimensions are usually the same), here we investigate S-boxes where the output is strictly smaller than the input. Such S-boxes have practical applications in authentication codes or secret sharing schemes but are also interesting as combinatorial optimization problems and could be used as benchmarks.

The results confirm that evolutionary computing algorithms can create S-boxes of different sizes. As in the previous chapter, the genetic algorithm achieves the best results.

Finally, the results show that when the number of outputs is strictly smaller than the number of inputs, this problem is more reminiscent of a Boolean function case than the S-box case. Moreover, differing from results usually obtained for S-boxes with $m = n$, evolutionary algorithms can be a viable option when several different functions are needed.

Moreover, we investigate whether heuristics can be used to generate differentially 6-uniform $(n, n-2)$ functions. Our little theoretical work on this topic points us to the conclusion that this problem is challenging. Our results confirm that the problem is intricate and evolutionary algorithms are of limited success.

We are able to obtain global optimums for only dimensions $(4,2)$ and $(5,3)$. Interestingly, different encodings are successful for those two dimensions. Since there exist algebraic constructions able to reach the same results for those two dimensions, we cannot report any new findings. Finally, as it can be observed from the presented box plots, there is not much variety in the obtained solutions. Unfortunately, since the differential uniformity values are always even valued integers, obtaining a more fine-grained fitness function seems difficult. This could indicate that evolutionary algorithms are not the best option for this problem, but still, the results are highly competitive with the state-of-the-art. The failure of the evolutionary approach for larger dimensions cannot, of course, be considered proof that such functions do not exist when $n \geq 6$. Still, considering that the algebraic constructions also cannot find differentially 6-uniform when $n \geq 6$, we believe it could serve as a strong indication of the nonexistence of such functions.

Chapter 5

Automatic Construction of Cryptographic Algorithms

In this chapter, we investigate the possibility of utilizing evolutionary algorithms to generate cryptographic algorithms without specific design criteria automatically. To this end, we employ Cartesian Genetic Programming in a bi-level setting where multiple populations simultaneously evolve a cryptographic algorithm, and an attacker attempts to break it. To challenge our design paradigm, we explore various scenarios with different criteria on the system and its security. Our results demonstrate promising outcomes in several scenarios where the attacker cannot comprehend the text with more than a random chance. While our system may not be practical, it presents an interesting approach, producing human-readable results. Moreover, our system can generate multiple versions of one-time pads, the only systems ensuring perfect secrecy.

The remainder of this chapter is structured as follows. In Section 5.1, we provide an overview of cipher design and the motivation for our research. Section 5.2 surveys related work, and Section 5.3 details our experimental setup. We describe the general principles of cipher design and bi-level optimization and the cost functions utilized in the evolutionary algorithm. Section 5.4 discusses the results and outlines potential avenues for future research. Finally, Section 5.5 presents a brief conclusion.

5.1 Introduction

Designing a cipher is usually very complex since the designers need to follow several principles to create a strong cipher. In the design phase, it is necessary to consider the properties of individual components (commonly known as cryptographic primitives) and the complete cipher. At the same time, that cipher needs to be tested against many possible attacks (e.g., differential cryptanalysis [93] or linear cryptanalysis [94]) to gain confidence in its strength. Although computers are extensively used in the design process to test specific parts of the cipher, in

modern ciphers, the design is exclusively done by human experts. Here, by design, we consider the choices on how to combine lower-level primitives into a cipher. Naturally, those lower-level primitives are often obtained/validated via computer investigation.

In this chapter, we pursue the goal of the automatic design of ciphers. To evolve ciphers, we use evolutionary algorithms, more precisely, Cartesian Genetic Programming (CGP) [14]. CGP is used in the optimization procedure due to interpretability, parallel construction of multiple solutions, and the inherently solved problem of excessive individual growth - bloat. We believe such automatic cipher design is exciting 1) as an exercise to see the limits of evolutionary algorithms in modern cryptography and 2) as a source of inspiration for new ciphers or their components. The automatic evolution of ciphers is a difficult task. That difficulty stems from the fact that we aim to develop a cipher easily usable by legitimate parties (commonly denoted as Alice and Bob). At the same time, the malicious third party (commonly known as Eve) should not be able to eavesdrop on that communication unless she has the key. To be as generic as possible (i.e., to allow the evolutionary algorithm to design a cipher freely), we must impose no (or as limited as possible) criteria on how the communication should happen. This constitutes a vast search space of solutions where 1) one side (we denote our cipher designer as Alice) generates a cipher, and 2) Eve must not be able to understand the message since she does not have a key. Note, Eve knows both plaintexts and corresponding ciphertexts, which puts our setting into a well-known attack model called the Known Plaintext Attack (KPA) [99]. Ideally, Eve could use that information to develop some attacks better than just random guessing.

Abadi and Anderson used two neural networks (Alice and Bob) to construct a cipher and a third network (Eve) to attack it, thus having an adversarial environment between the first two networks and the third one [121]. The only constraint they imposed in the design process is that legitimate parties need to find a cipher so that they can communicate while Eve cannot decipher it. This should be possible since Eve has a much more difficult task because she does not know the secret key, while the legitimate parties know it. Abadi and Anderson obtained relatively good results, i.e., they found a way for Alice and Bob to communicate. At the same time, Eve could not decipher that communication, but their approach had issues. While Eve could not decipher the communication with significantly better chances than random guessing (i.e., 50% of plaintext characters correctly guessed), Alice and Bob communicated "successfully," but their error was slightly lower than 50%. Next, since the cipher is a neural network architecture, no proper design analysis is possible. In the same way as it is difficult (impossible) to interpret the way Alice and Bob communicate, it is difficult to understand what is Eve doing so there is no guarantee the system is successful simply because Eve was not able to learn good attack strategies (i.e., the cipher could be bad but Eve is so primitive she still cannot break it).

Our design principles differ in several ways to evolve a cipher that ensures a certain level of security while being at least partially interpretable. Instead of neural networks, we use Cartesian

Genetic Programming since that allows us to have solutions in the form of graphs, which are (potentially) understandable by human designers. We do not use the scenario where Alice and Bob develop a cipher since we do not see a practical justification. Indeed, it is sufficient that one side generates a cipher and then shares it with all parties. Our setting uses the bi-level optimization for Alice and Eve. Finally, since the design of a cipher also depends on its intended use, we set this as a constraint in our process. Indeed, the evolutionary algorithm cannot guess what we want to use in the cipher. We can show our design strategy works over many different settings and produces solutions (i.e., ciphers) where the attacker's best strategy is simply random guessing. At the same time, our solutions are short enough, so it is possible to analyze them. Interestingly, even when not imposing any specific design constraints, our approach still finds some general paradigms of good cipher design (e.g., it is important to use the key – although this sounds trivial, it is still a result obtained solely by the evolution process). Naturally, once we can automatically design a cipher, the question is why to use it. Due to a lack of proper cryptanalysis, there must be a severe concern about the strength of such evolved ciphers that would prohibit them from being used. Still, there are many motivations for this work:

1. Testing the limits of what evolutionary algorithms can do.
2. Finding new building blocks for ciphers – since we do not impose criteria on how a cipher should look, new primitives could be found. The same logic applies to Eve, where we could find new techniques to attack a cipher.
3. Since Alice generates a cipher and Bob must find a way to understand it (regardless of whether it is the same cipher or some functionally equivalent one), one could use our setting to find smaller, functionally equivalent ciphers.
4. In scenarios where some fault (for instance, in satellites caused by cosmic radiation) renders one side in communication non-operational, our setting could be used to evolve a different (but equivalent) cipher with the still functional circuitry.
5. Finally, one can consider, e.g., an evolvable hardware setting where the system could adapt to new threats or design paradigms by simply modifying the fitness functions.

In this chapter, we realized the original scientific contribution of the automatic construction of cryptographic algorithms by using attacker and defense dynamics in the security domain as follows[122]:

- to the best of our knowledge, we are the first to consider such an open-ended cipher evolution process with evolutionary algorithms,
- we design an entire system named C^3PO where Alice can design ciphers considering various criteria and cipher characteristics,
- we show that bi-level CGP can construct ciphers that Eve cannot break.

5.2 Related Work

When discussing heuristics in the design of ciphers, there are two directions in the literature. The first deals with the design of cryptographic primitives, i.e., parts of ciphers, while the second tackles the problem of the full cipher design. Since the design of a part of a cipher is easier than the design of the whole cipher, the first direction has been much more explored, and the results are better.

Among the various heuristic techniques adopted for the problem of evolving Boolean functions (often used in stream ciphers) one can find simulated annealing [123], genetic algorithms [71], genetic programming and Cartesian genetic programming [76], particle swarm optimization [124], and immunological algorithms [81]. All those approaches follow the same line of reasoning: they define important cryptographic criteria that Boolean functions need to fulfill and incorporate them into fitness functions. Next, researchers use heuristics to generate Substitution boxes (S-boxes) to be used in block ciphers. Examples use simulated annealing with hill climbing [111], genetic algorithms [112], genetic programming [125], Cartesian genetic programming [126], and gradient descent method[127]. Like the Boolean functions, fitness functions contain specific properties that an S-box should possess. When considering the design of full ciphers, we distinguish two options: in the first one, the design follows precisely devised criteria defining the behavior of the cipher, while in the second direction, the process is more open ending since there are no specific constraints. Examples in the first avenue encompass the design of pseudorandom number generators where the fitness function uses various types of randomness testing to evaluate whether the constructions offer sufficient randomness. The approaches use genetic programming [128] and Cartesian genetic programming [129]. A block cipher called Wheedham is designed by genetic programming where a fitness function ensures sufficient nonlinearity in the cipher [130].

Aside from the evolutionary computation, a number of papers belong to the neural cryptography domain. A usual goal is to develop a key exchange protocol [131]. Still, such systems do not offer security as, for instance, shown by Klimov et al. [1]. Finally, adversarial neural networks are used to design a cipher where the only constraints are that Alice and Bob need to exchange messages. At the same time, Eve should not be able to eavesdrop on them [121]. As far as we know, this is the first attempt to build a whole cipher in an "open" design style where Eve is also considered.

5.3 Experimental Setting and Results

In this section, we start by a short discussion about general properties we require from our evolved ciphers. Next, we discuss Cartesian genetic programming (CGP) as the algorithmic

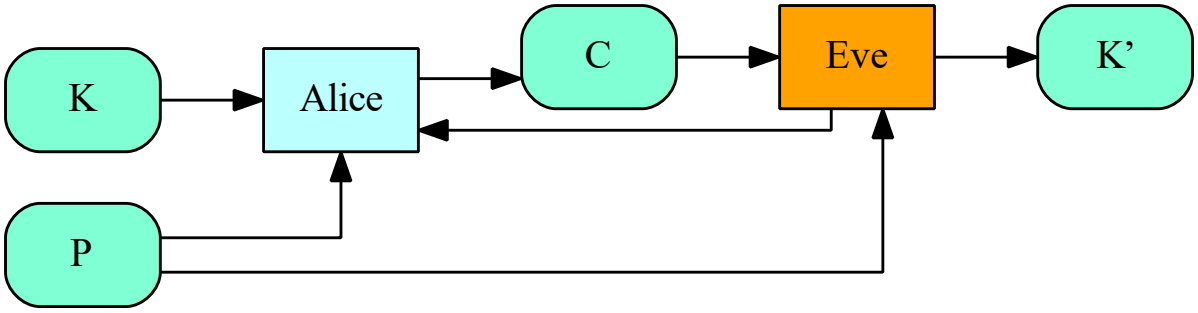


Figure 5.1: Alice and Eve in bi-level optimization.

paradigm used in experiments. Finally, we present fitness functions for all parties – Alice, Bob, and Eve.

5.3.1 General Cipher Design Principles

As already said in Section 5.1, our goal is the automatic design of ciphers where we do not impose criteria on how the cipher should be designed. Indeed, described as briefly as possible, we want a cipher that legitimate parties can use to communicate and that the attacker cannot break. Figure 5.1 shows the design scheme we used in our experiments. Everything else should be (in the ideal case) designed by the evolutionary process. Yet, it is not difficult to see that some additional constraints need to be given since EA does not know what kind of cipher it needs to generate:

1. In the evolution process, we do not use two parties (Alice and Bob) to evolve a cipher but only Alice. We see no need to strain the evolution process and make Bob guess what Alice finds. First, guessing the cipher correctly is extremely difficult and adds errors to legitimate parties communication. Additionally, there is no reason to limit the communication to only two legitimate parties. In a system where one side develops a cipher, and the other sides need to guess it to communicate, adding more parties means even more errors in legitimate communication. Second, we do not want the cipher to be a secret but only the key following Kerckhoff's principle so Alice can develop a cipher and send it over an insecure network.
2. What kind of a cipher do we require, public-key or symmetric key algorithm? In this chapter, we consider only symmetric-key algorithms.
3. Does our algorithm use the key or not? Both options are possible: for instance, block-/stream ciphers use keys while hash functions do not use them. We require that our designs use the key.
4. We consider symmetric-key algorithms that use keys but still give several options. Will the cipher operate on bits of blocks of bits? Although this seems like a detail that could be left to EA to decide, our experiments show that if we allow EA to operate on a bit level, it

will not group those bits into blocks. We choose to work with blocks and, consequently, to design block ciphers.

5. What is the size of plaintext, ciphertext, and keys? We consider a scenario where all have the same size, either 4 bits or 8 bits.

Note, all decisions given above are our design choices and not something devised due to some constraints on EA. Indeed, we could have decided to work on stream ciphers or hash functions, and EA would still be able to produce results. While 4 or 8 bits can look too small to be practical, applying the same cipher to any number of bits in parallel and arriving at more practical sizes is easy. Of course, since we consider each block separately, there is no diffusion between the blocks, but we do not consider this a problem at this phase. Finally, although it is common for block ciphers to be iterated [99], i.e., to operate in several rounds to improve their security, we use only a single round here. Trivially, each new round adds a certain amount of security to the cipher, but we can measure the security only concerning Eve. More precisely, if Eve cannot break a single round cipher, there is no reason to add more rounds since Eve will naturally be unable to break them. Adding rounds will make the attacks more difficult if Eve can break a single round. Still, our experiments show many designs with a single round strong enough so Eve cannot break them. Consequently, we see no need to consider multiple rounds at this point. Naturally, we do not limit EA, so it cannot produce multiple rounds, but the analysis of the obtained solutions did not indicate this to be the case.

Besides these general constraints, our experiments indicated arguments for several additional constraints that are more specific. If we require our cipher to be bijective, i.e., invertible, we need to encode this constraint. This may seem like a limitation, but it is a natural situation since EA does not know whether there is another side (or sides) that needs to be able to decrypt the ciphertext by inverting the encryption procedure. We note this is also not a must; any function can become invertible if it is set into the Feistel structure [99].

Once the evolution process is finished and we have the cipher that Eve could not break, we would require some assurance that our cipher is strong. Indeed, if Eve cannot break the cipher, there are two reasons: 1) the cipher is strong, and that is the reason Eve cannot break it, or 2) Eve did not learn any good way to analyze the cipher, so although the cipher is weak, she cannot break it (colloquially said, Eve, is "stupid"). To gain some assurance that the cipher is strong and that Eve is a capable attacker, we measure the confusion and diffusion as given by the nonlinearity and avalanche criterion, respectively.

5.3.2 Bi-level optimization

Bi-level optimization is a special kind of optimization where one problem is embedded within another one [132]. The outer optimization task is commonly referred to as the upper-level optimization task, and the inner optimization task is commonly referred to as the lower-level

optimization task. In our case, Alice does the upper-level optimization task referring to Eve's lower-level task.

When evaluating each individual in the upper-level population (each Alice), a new lower-level population (of Eves) is created. The chosen Alice individual's cipher is used to generate training set pairs of plaintext and ciphertext; this training set is used at the lower level to evaluate lower-level individuals. After the lower-level evolution is terminated, the best solution from the lower level is used to estimate the fitness of Alice, i.e., the upper-level individual.

The same evolutionary algorithm and the same representation is used at both levels, but the fitness functions and termination criteria are different. In our case, the size of the lower-level population is $\sigma = 3$, and the lower-level termination criteria is many evaluations. This process is illustrated in Algorithm 7. Train and test datasets are denoted with $\{P,K\}^N$ and $\{P',K'\}^M$, where P and P' are plaintexts, and K and K' are keys. The size of the datasets is marked with N and M; C and C' are ciphertexts obtained from the train and test dataset; Π stands for the population of candidate attackers, and π denotes the iteration counter.

Algorithm 7 Alice and Eve evolution by bi-level optimization.

Input: $\mathbb{X}_{train} = \{P,K\}^N$ – train dataset, $\mathbb{X}_{test} = \{P',K'\}^M$ – test dataset, I – iterations
$\pi = 0$
repeat
  $Alice =$ build model $(\mathbb{X}_{train}, Eve)$
  $C =$ encrypt $(\mathbb{X}_{train}, Alice)$
  $\Pi^{\sigma}_{Eve} = Eve_1, ..., Eve_{\sigma}$
  for $\forall Eve$ in Π^{σ}_{Eve} **do**
    $Eve_i =$ build model $(\{P,C\}^N)$
  end for
  $Eve =$ best model in Π^{σ}_{Eve}
  $C' =$ encrypt $(\mathbb{X}_{test}, Alice)$
  $\hat{K}' =$ break secret key $(\{P',C'\}^M, Eve)$
  if $\hat{K}' = K'$ or $\hat{K}' = \neg K'$ **then**
    secret key K' broken
  end if
  $inc(\pi)$
until $\pi < I$

5.3.3 Common Parameters and Datasets

The table 5.1 shows the parameters for CGP. The function set consists of three binary and three unary functions. Binary functions are AND, OR, XOR, and unary functions are NOT, ROR and ROL. ROR and ROL are rotation functions that rotate a bit string by one bit to the right or left. In all experiments, the number of independent trials for each configuration is 30. We use two artificial datasets depending on the message length that can be $n = 4$ and $n = 8$ bits.

Table 5.1: Parameters for CGP.

Parameter	Value
Genotype length	20/40 nodes (4/8 bits)
Input/Output nodes	2/1
Shortcut connections	disabled
Evolutionary strategy	(1+4)
Mutation type	single active gene
Functions	AND, OR, NOT, XOR, ROR, ROL
Maximum evaluations	25 000/50 000 (Alice/Eve)
Runs per experiment	30

Here, both the message and the secret key are of the same length. The message and the secret key consist of uniformly distributed binary values. After a tuning phase, we set the number of nodes in CGP for 4-bit messages to 20 and for 8-bit messages to 40 nodes. To encrypt a message block consisting of n bits with a key consisting of n bits as well, we use vectorial functions that produce n bit ciphertext. Consequently, Alice consists of 2 input nodes and 1 output node. Eve has information about the pairs of plaintext and ciphertext so she also has 2 input nodes and 1 output node, where output represents the secret key used in the encryption. Note, our encryption algorithm always outputs ciphertexts of the same size as is the plaintext. For the 4-bit messages, the training dataset contains $P = 150$ messages and $K = 10$ keys ($N = 150$, training set size), while testing dataset contains $P' = 50$ messages and $K' = 6$ keys ($M = 50$, testing set size). The 8-bit messages the training dataset contains $P = 400$ messages and $K = 50$ keys ($N = 400$) while testing dataset contains $P' = 100$ messages and $K' = 10$ keys ($M = 100$). All pairs plaintext/key are selected uniformly at random. We divide our evolution process into a number of evaluations E and iterations I. A single iteration I represents a process where all parties undergo E evaluations. Training and testing set is regenerated after each iteration. In all our experiments, the evolution does $I = 50$ iterations. We set the number of evaluations E to 25 000 for Alice and 50 000 for Eve. By larger number of evaluations, we give Eve an advantage over Alice in order to build as powerful as a possible attacker. In total, each of 30 experimental runs has at least 1 250 000 evaluations for each of the populations.

5.3.4 Cost Functions

Before we explain the cost functions, we explain Alice and Eve's tasks. Alice must build such a cryptographic algorithm where the attacker will have an average probability of guessing the secret key equal to random guessing. Additionally, it is required that the constructed cryptographic algorithm has high nonlinearity and diffusion propertie and is bijective. On the other

hand, the task of the attacker, Eve, is to discover as much information as possible about the secret key, or the inverted secret key. Eve has pairs of plaintexts and associated ciphertexts because she follows KPA.

We use the L1 distance to measure the difference between the key K and the guessed key K'. The L1 distance is defined as $d(K,K') = \sum_{i=1}^{n} |K_i - K'_i|$, where n equals the message length. We work in the multi-objective setting since we simultaneously aim to maximize Eve's decryption error and to satisfy the good properties of a cipher: high nonlinearity, high diffusion, and bijectivity. The last property enables decryption with an inverse function.

Alice's $cost_A$ expression represents Eve's error, where the global optimum is reached when half (on average) of the decrypted message bits are wrong. Suppose Eve is correct in only half the bits. In that case, that means she is doing random guessing (independent coin toss for each message bit), which is the optimal scenario from Alice's perspective. Note, if Eve would be wrong in significantly more than half the bits, we could invert her guesses (swap all 0s for 1s and vice versa), which would make her wrong in significantly less than half of the bits. The second component $crypto_{Eve}^{Alice}$ describes the simultaneous fulfillment of cryptographic properties. We emphasize that Alice's cost function is measured by Eve and then, in bi-level optimization, sent to Alice.

$$cost_{Alice} = \sum_{i=1}^{N} \left| \frac{n}{2} - d(K, K'_{Eve}) \right| + crypto_{Eve}^{Alice}. \tag{5.1}$$

$$cost_{Eve} = \sum_{i=1}^{N} min(d(K, K'_{Eve}), n - d(K, K'_{Eve})). \tag{5.2}$$

$$crypto_{Eve}^{Alice} = \widehat{NL} + \widehat{diffusion} + bijection. \tag{5.3}$$

The bijection is determined by creating unique ciphers with different messages and the same key. The values of this fitness component are 0 if the Alice constructs a non-bijective cipher and 1 otherwise. Nonlinearity NL and diffusion are statistically measured with a subset of secret keys from the training set. To reduce the NL and $diffusion$ time complexity, we randomly choose 3 keys and calculate cipher's nonlinearity and diffusion, where $\widehat{NL} = min\{NL_{key^i}\}$ and $\widehat{diffusion} = min\{diffusion_{key^i}\}$. Since the parameters values have different co-domains, we scale $\widehat{NL}$, $\widehat{diffusion}$, and bijection to the interval $[0,N]$, using transformation $scaled = N\frac{optimal-value}{optimal}$. Optimal nonlinearity for 4 bit messages is 4 and for 8 bit messages is 112. On the other hand, optimal diffusion for each bit of the output is equal to $\frac{n^2}{2}$ because for each message, there exists n messages different in only 1 bit which should have cipher difference in $\frac{n}{2}$ bits.

5.4 Results

We divide our experiments into five scenarios when considering cryptographic cost on the Alice side. The first scenario includes only Eve's L1 measure, which Eve tries to minimize, and which for Alice needs to be as close to half of the bits as possible. In the second scenario, we consider Alice's diffusion. The third scenario considers nonlinearity, while the fourth scenario measures Alice's bijection property. Finally, we combine the second, third, and fourth scenario into the fifth scenario. The aggregate results for all scenarios are given in Tables 5.2–5.3 and Figures 5.2–5.7.

Table 5.2: Alice, average results for all scenarios.

N	Scenario	average ± standard deviation						
		$cost_{Alice}$	active nodes	diffusion	nonlinearity	bijectivity	$cost_{Eve}$	$crypto_{Eve}^{Alice}$
4 bits	1	11.20 ± 2.43	4.60 ± 2.36	106.13 ± 24.95	0.40 ± 1.20	0.13 ± 0.33	0.81 ± 0.17	10.38 ± 2.40
	2	8.91 ± 0.46	5.16 ± 2.33	**4.80 ± 12.49**	0.00 ± 0.00	0.00 ± 0.00	0.76 ± 0.24	8.15 ± 0.39
	3	7.14 ± 1.29	5.30 ± 1.91	62.53 ± 8.16	3.60 ± 1.20	0.00 ± 0.00	0.78 ± 0.15	6.35 ± 1.29
	4	7.03 ± 0.46	3.23 ± 2.01	72.53 ± 14.15	0.00 ± 0.00	**1.00 ± 0.00**	0.76 ± 0.13	6.26 ± 0.44
	5	**1.09 ± 0.88**	10.00 ± 1.50	9.00 ± 5.76	**3.86 ± 0.71**	**1.00 ± 0.00**	**0.68 ± 0.16**	**0.41 ± 0.89**
8 bits	1	21.68 ± 3.78	7.73 ± 3.35	6779.73 ± 1044.84	6.40 ± 19.20	0.20 ± 0.40	1.11 ± 0.15	20.56 ± 3.76
	2	16.95 ± 0.82	9.96 ± 2.40	**17.06 ± 91.90**	2.13 ± 11.48	0.00 ± 0.00	1.08 ± 0.11	15.86 ± 0.82
	3	11.76 ± 1.37	15.20 ± 2.48	2314.67 ± 925.86	**106.13 ± 7.71**	0.00 ± 0.00	1.08 ± 0.11	10.67 ± 1.36
	4	15.35 ± 0.32	5.70 ± 3.11	6400.00 ± 288.11	0.00 ± 0.00	**1.00 ± 0.00**	1.10 ± 0.10	14.25 ± 0.28
	5	**10.40 ± 1.13**	13.16 ± 2.92	1366.40 ± 1148.21	0.00 ± 0.00	**1.00 ± 0.00**	**1.06 ± 0.11**	**9.33 ± 1.12**

Table 5.3: Eve, average results for all scenarios.

N	Scenario	average ± standard deviation			
		$cost_{Eve}$	active nodes	keys broken	secret key wrong bits
4 bits	1	0.78 ± 0.51	7.40 ± 3.08	**4.20 ± 1.83**	1.97 ± 1.14
	2	1.23 ± 0.21	8.10 ± 2.79	2.80 ± 1.72	2.02 ± 0.26
	3	1.14 ± 0.34	7.93 ± 2.93	2.63 ± 1.68	1.91 ± 0.59
	4	**0.86 ± 0.54**	6.96 ± 2.62	3.56 ± 2.06	**1.80 ± 1.03**
	5	1.17 ± 0.33	6.63 ± 2.18	2.40 ± 1.94	1.96 ± 0.22
8 bits	1	1.98 ± 1.05	14.16 ± 6.58	3.76 ± 3.66	4.14 ± 2.09
	2	2.66 ± 0.72	17.26 ± 4.28	1.66 ± 2.64	**3.67 ± 1.01**
	3	2.82 ± 0.20	17.50 ± 3.98	1.10 ± 1.42	3.96 ± 0.42
	4	**1.62 ± 1.39**	12.36 ± 5.55	**5.06 ± 4.48**	4.00 ± 2.63
	5	2.77 ± 0.32	18.60 ± 3.65	1.53 ± 1.82	3.81 ± 0.72

5.4.1 Scenario 1

In this scenario, $crypto_{Eve}^{Alice}$ is set to 0 in Eq. (5.1) and only Eve's L1 measure is optimized. For Eve, the aim is to guess as many bits as possible (or as *little* bits as possible, see Eq. 5.2). Then, the best solution from Eve lower-level population is used as the fitness for the current Alice candidate. When considering 4 bits scenario, Alice evolves a cipher that is small (as can be observed by the number of active nodes) but the cipher is relatively weak since Eve is able to guess many bits. We see that the diffusion is bad (since we aim to minimize it) as well as the nonlinearity (since we aim to maximize it). Both properties indicate that our ciphers are not providing a lot of security. Additionally, we see that the evolved ciphers are usually not bijective. The scenario with 8 bits displays a similar behavior, where values are slightly larger but also their range is much larger. At the same time, Eve's cost is relatively large, which means that she is able to do better than random guess. In Table 5.2, we show the situation after Eve's evaluation where we see that she is able to break several keys and that she makes mistake in only a few bits of the secret key.

5.4.2 Scenario 2

In this scenario, besides Eve's L1 measure, the fitness component $crypto_{Eve}^{Alice}$ includes the diffusion which is also optimized. We can see that this has caused a significant change when comparing with Scenario 1. The cost of Alice is reducing, which means she is developing better ciphers (i.e., those that are more difficult for Eve) but she also needs more active nodes on

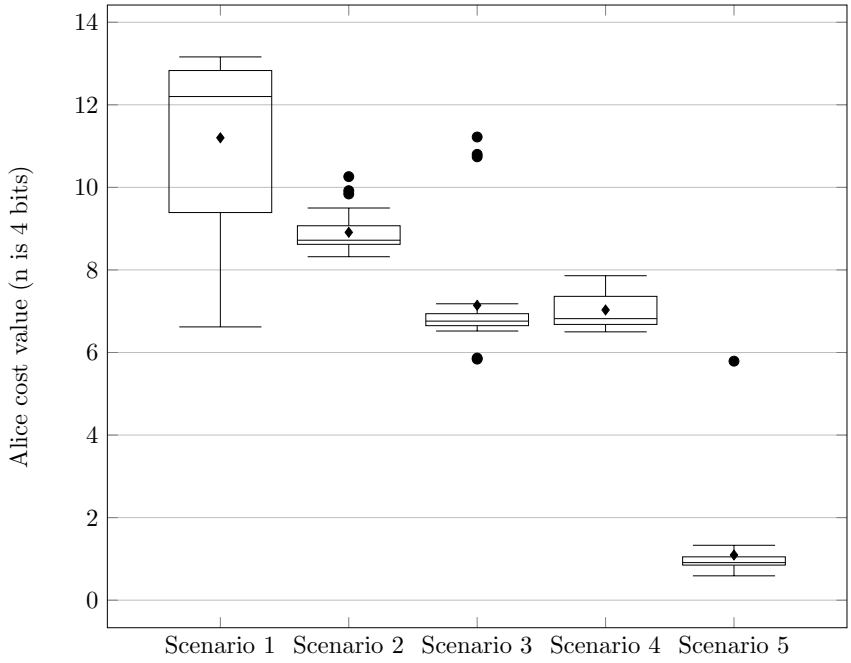


Figure 5.2: Alice cost function values in all scenarios for $N = 4$.

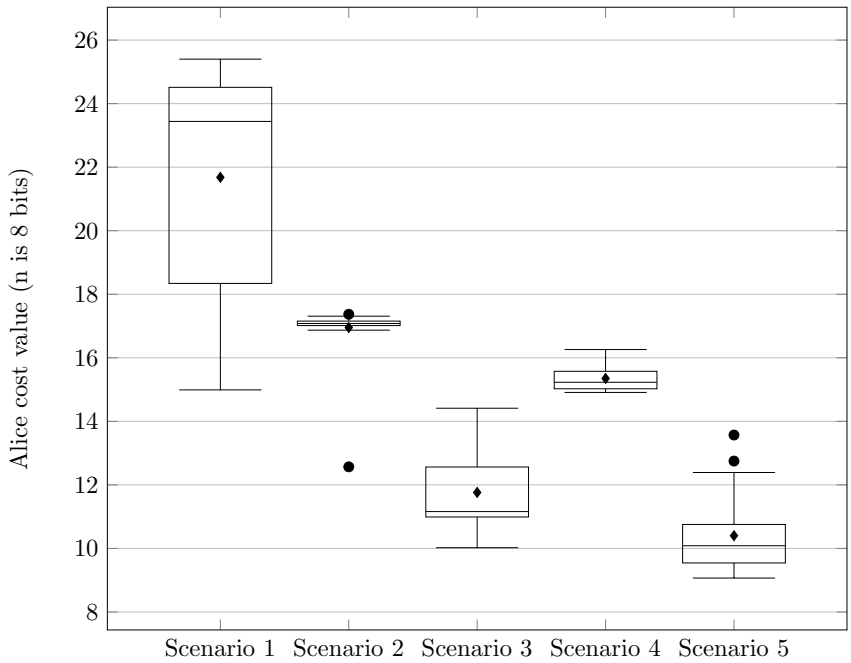


Figure 5.3: Alice, cost function values in all scenarios for $n = 8$.

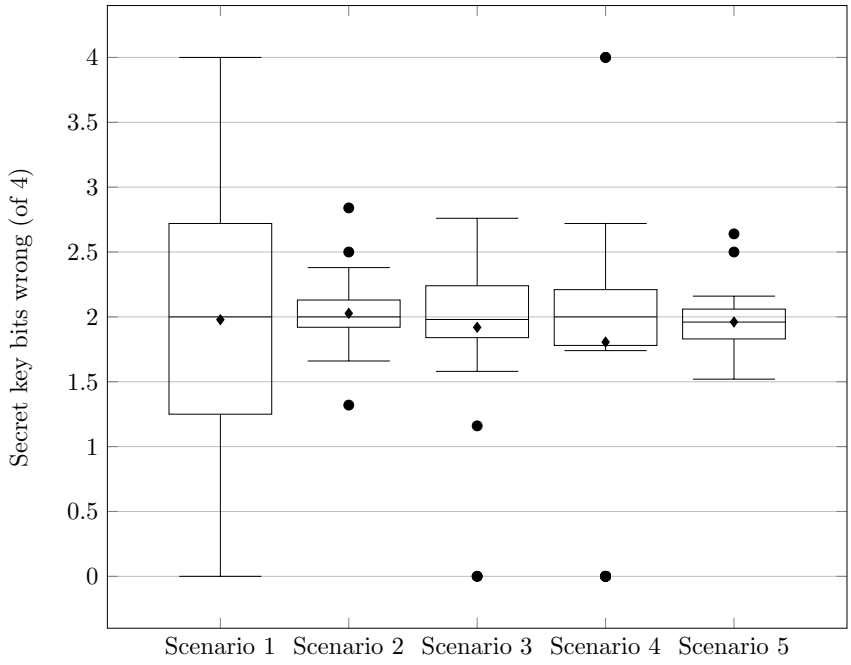


Figure 5.4: Eve, average key bits error in all scenarios for $n = 4$.

average. Since we optimize for diffusion, we see that for both 4 and 8 bits, the diffusion significantly improves. The ciphers are never bijective (both for 4 and 8 bits) and the nonlinearity decreases. This indicates that if we want to optimize for diffusion, we loose on nonlinearity. Considering Eve, we see that her results indicate that she is closer to random guessing and consequently, she guesses less keys.

5.4.3 Scenario 3

This scenario optimizes Eve's L1 measure and in the term $crypto_{Eve}^{Alice}$ the nonlinearity is optimized. We can observe that in this case, nonlinearity is significantly better than in the previous scenarios. At the same time, apart from the diffusion, the other components are largely not affected. As in the previous cases, bijectivity is not obtained, which again indicates that this property is not intrinsic to the design of Alice's cipher. The sizes for ciphers that Alice evolves are very similar for Scenario 2 and 3, which means that the properties are similar in complexity to add to the cipher design. When considering the results from Alice's perspective, it is difficult to say whether Eve has more problems for Scenario 2 or Scenario 3 since the results are very similar. When considering the results from Eve/s perspective, we see that she needs less active nodes for nonlinearity and she is able to guess the keys better.

5.4.4 Scenario 4

In this scenario, the component $crypto_{Eve}^{Alice}$ in Alice's cost includes only the bijection term, while optimizing Eve's L1 measure. When explicitly included as a criterion, we observe that

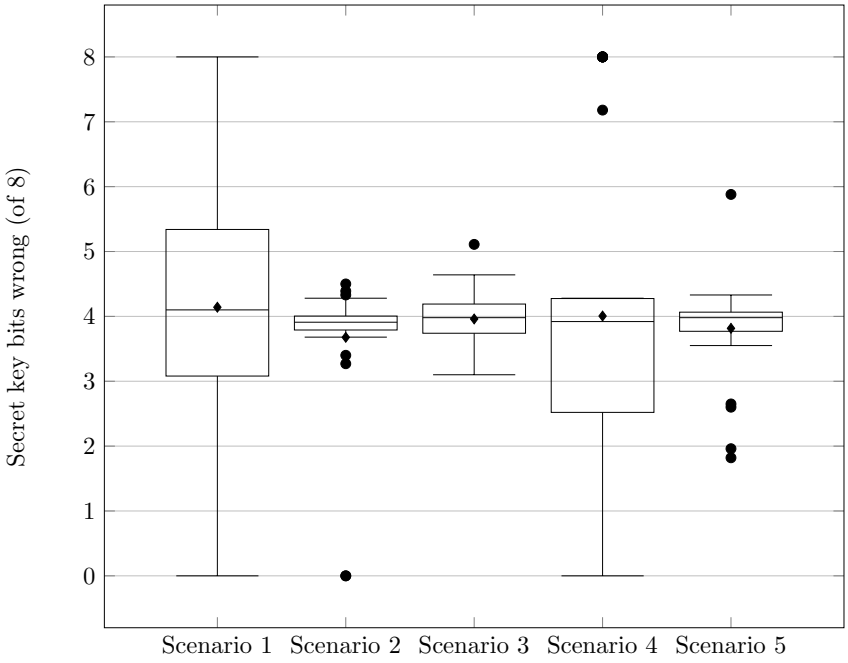


Figure 5.5: Eve, average key bits error in all scenarios for $n = 8$.

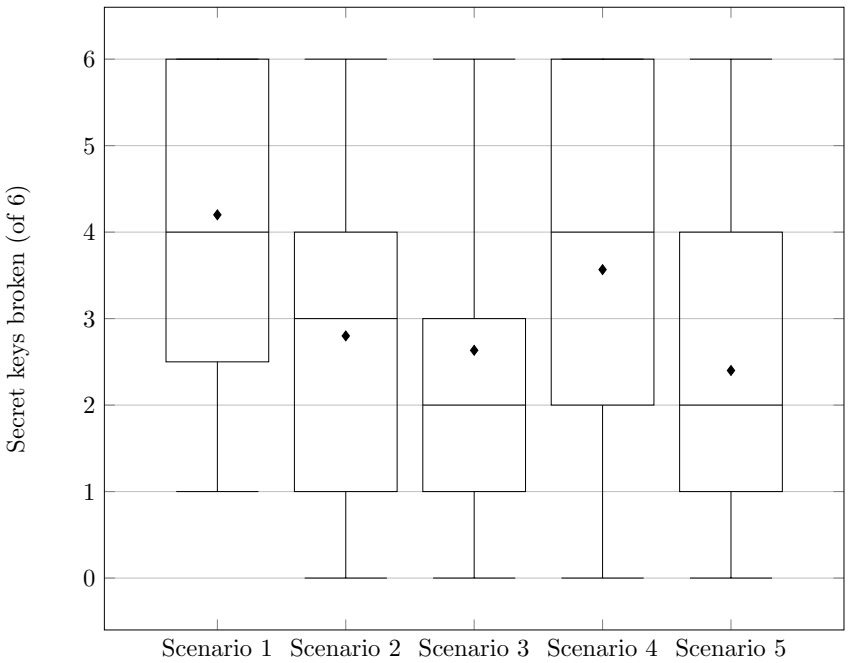


Figure 5.6: Eve's broken keys in test set for all scenarios for $n = 4$.

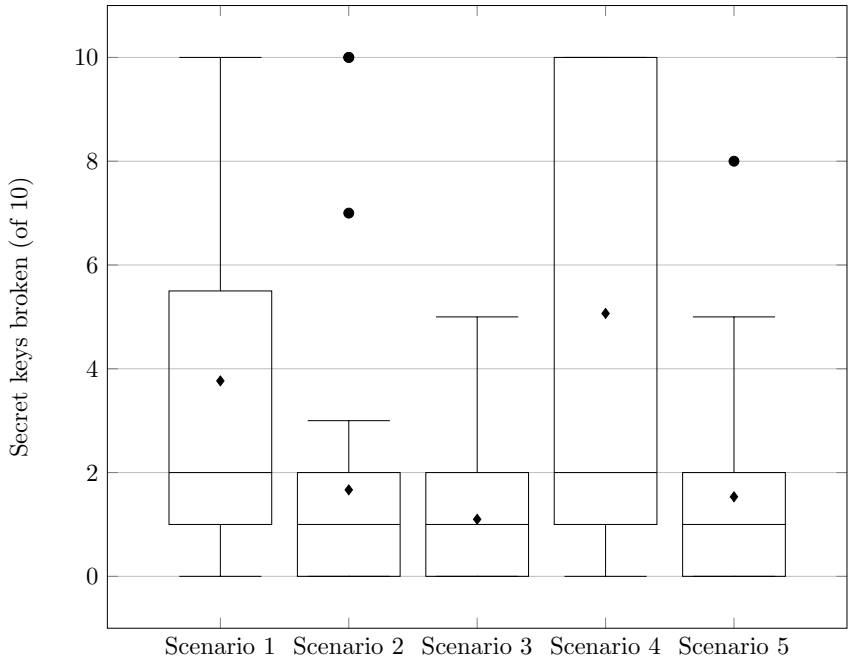


Figure 5.7: Eve, broken keys in test set for all scenarios for $n = 8$.

the evolutionary algorithm can easily adapt and provide a bijective cipher. It is important to remember that the bijectivity is not tested and guaranteed on the whole domain, but only on the generated plaintext dataset. Interestingly, making a cipher bijective is still causing a similar amount of problems for Eve despite the fact that diffusion and nonlinearity are not as good as for Scenarios 2 and 3. This constraint does not seem to make a significant change in Eve's ability to obtain better solutions in competition with Alice. From Eve's perspective, making this scenario enables her to guess the most of the bits and she requires less active nodes when compared to the first 3 scenarios. Additionally, when considering 8-bit scenario, we see that Eve is able to brake most of the keys.

5.4.5 Scenario 5

In this scenario, besides optimizing Eve's L1 measure, $crypto_{Eve}^{Alice}$ includes all the three previous criteria. We see that our ciphers are bijective and with good diffusion (although not as good as when considering only diffusion). When considering scenario with 4 bits, we see we are also able to obtain the best possible nonlinearity while for 8 bits we cannot obtain nonlinear functions. This means that the problem is simply too difficult for CGP when considering all conditions. Still, even with such differing results for two cases, we see that this scenario is the most difficult one for Eve and she is making mistakes the most. Interestingly, for 4-bit scenario, Eve uses the least active nodes while for the 8-bit scenario she uses the most active nodes. This indicates that she is somewhat "deceived" for 4-bit case to think the problem is easy. As one could expect, we consider this scenario to be the best for Alice (and consequently the worst for

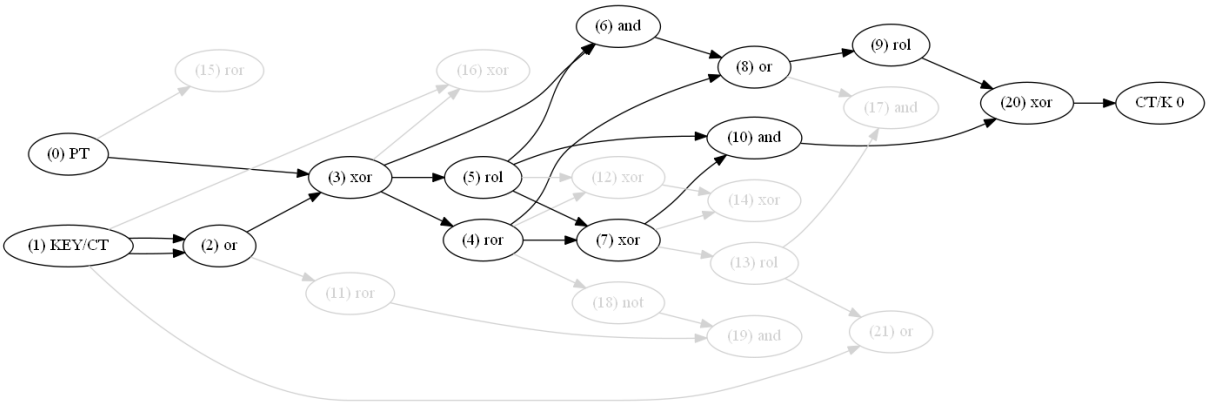


Figure 5.8: Example of an evolved cipher obtained by Alice.

Eve) since all statistical indicators for cryptographic properties are balanced. At the same time, Eve is making mistakes for many bits, which means that this scenario is difficult fo her. Finally, we depict one cipher working on 4 bits evolved by Alice in Figure 5.8 and corresponding Eve's attack in Figure 5.9.

On a general level, we see that improvements in the cost function for Alice are making a big difference since she is able to obtain dramatically different results. Eve, on the other hand, is always making errors in around half of the bits. What is especially interesting is to note that we obtained for several scenarios one-time pad cipher and its variants, which are the only ciphers providing perfect secrecy.

5.5 Conclusions

In this chapter, we investigate how to automatically evolve ciphers with CGP and bi-level optimization. Our results show that we are able to develop ciphers that are (relatively) resilient against attacks and use only a small number of active nodes, which makes them easier to interpret. Once we add more properties that a cipher needs to fulfill, the results are naturally improved. Eve as the attacker is not able to be significantly more successful than if she would be random guessing. We consider this to be only a proof of a concept that shows that EA has potential as a automatic cipher builder. Naturally, to obtain something useful in practice, more refinements will be necessary.

In future work, we will consider more variations in the number of evaluations and/or size of CGP graph. Besides that, we notice that our cost function is often too strict, which results in getting stuck in local optima. To remedy that, we aim to design cost functions that gradually add constraints during the evolution process and not impose all of them from the beginning of the evolution process.

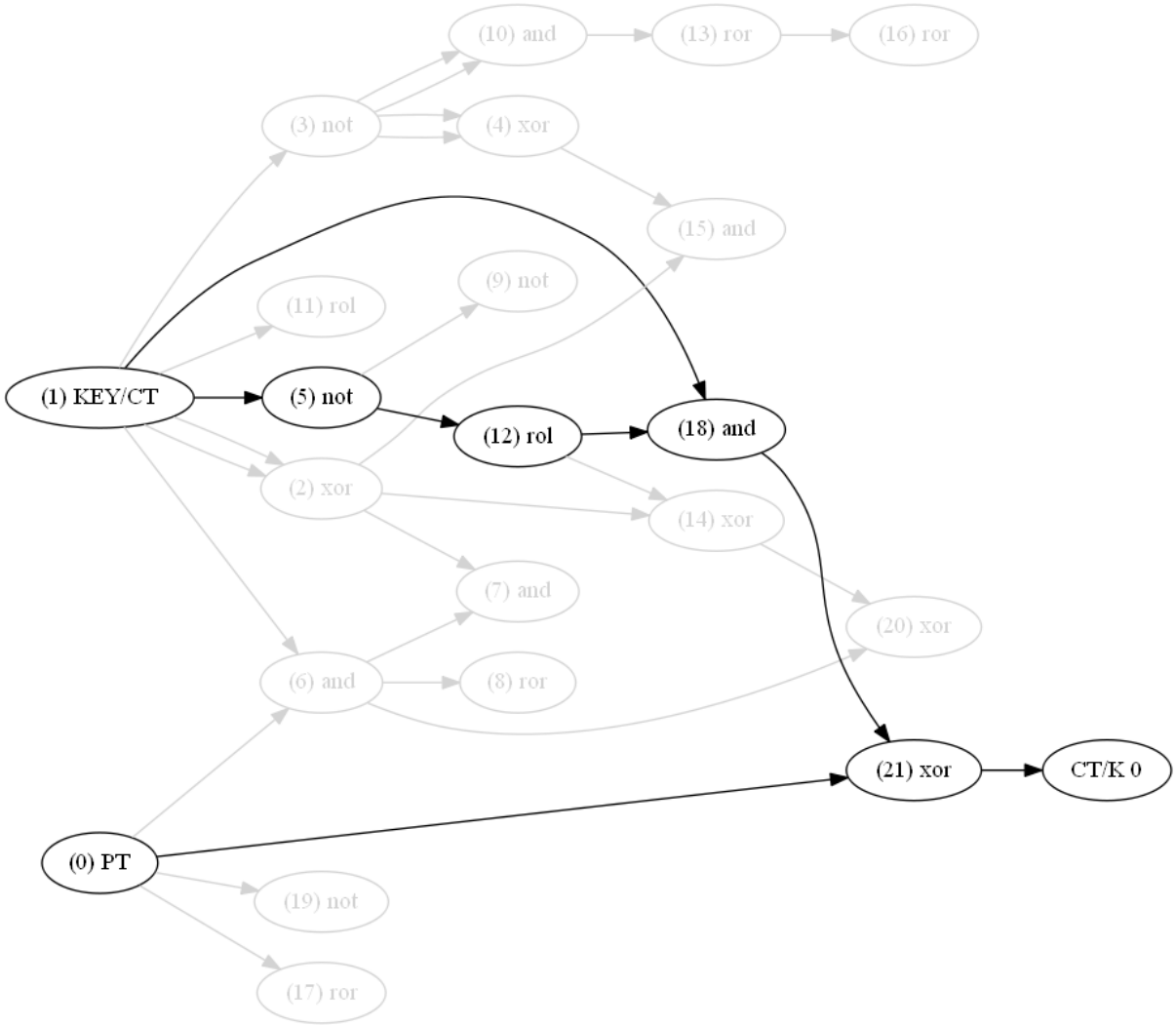


Figure 5.9: Example of an evolved attack obtained by Eve.

Chapter 6

Machine Learning Algorithms in Side-channel Attack

The profiled side-channel analysis represents the most powerful category of side-channel attacks. In this context, the security evaluator (i.e., attacker) gains access to a profiling device to build a precise model which is used to attack another device in the attacking phase. It is assumed that the attacker has significant capabilities in the profiling phase, whereas the attacking phase is very restricted. We step away from this assumption and consider an attacker restricted in the profiling phase, while the attacking phase is less limited. We propose the concept of semi-supervised learning to side-channel analysis, in which the attacker uses the small number of labeled measurements from the profiling phase and the unlabeled measurements from the attacking phase to build a more reliable model. Our results show that the semi-supervised concept significantly helps the template attack. For machine learning techniques and pooled template attack, the results are often improved when only a smaller number of measurements is available in the profiling phase. At the same time, there is no significant difference in scenarios where the supervised set is large enough for reliable classification.

The rest of this chapter is organized as follows. Section 6.1 gives an overview of the profiled attacks and motivation for research. In Section 6.2, we discuss the semi-supervised paradigm, how one can use it to boost classification results, and the classes of algorithms we consider. Next, in Section 6.3, we give details about the datasets we consider, the algorithms we use, and our experimental evaluation procedure. Section 6.4 presents the experimental results for both semi-supervised and supervised experiments, which serve as a baseline case. Finally, Section 6.5 offers a brief conclusion.

6.1 Introduction

Side-channel analysis (SCA) involves extracting secret data from (noisy) measurements. It is made up of a collection of miscellaneous techniques combined in order to maximize the probability of success, for a low number of trace measurements and as low computation complexity as possible. The most powerful attacks are based on a profiling phase, where the link between leakage and the secret is learned, assuming the attacker knows the secret on a profiling device. This knowledge is subsequently exploited to extract another secret using fresh measurements from a different device. To run such an attack, one has a plethora of techniques and options to choose from, where the two main types of attacks are based on 1) template attack (relying on probability estimation) and 2) machine learning (ML) techniques. When working with the typical assumption for profiled SCA that the profiling phase is not bounded, the situation becomes relatively simple if neglecting computational costs. Suppose the attacker can acquire an unlimited (or, in the real world, huge) amount of traces. In that case, the template attack (TA) is proven to be optimal from an information-theoretic point of view (see, e.g., [133, 134]). In the context of the unbounded and unrestricted profiling phase, ML techniques seem not needed.

Stepping away from the assumption of an unbounded number of traces, the situation becomes much more interesting and of practical relevance. Many results in recent years showed that in those cases, machine learning techniques can actually significantly outperform template attack (see e.g., [135, 136, 137]).

Still, the attacks described above work under the assumption that the attacker has a (significantly) a large amount of traces from which a model is learned. The opposite case would be to learn a model without any labeled examples. Machine learning approaches (mostly based on clustering) have been proposed, for instance, for public key encryption schemes where only two possible classes are present – 0 and 1 – and where the key is guessed using only a single trace (see, e.g., [138]). To the best of our knowledge, unsupervised machine learning techniques have not been studied in the case of differential attacks (using more than one encryption) and using more than two classes.

In this chapter, we aim to address a scenario between supervised and unsupervised learning, the so-called semi-supervised learning, in the context of SCA. Figure 6.1 illustrates the different approaches of supervised (on the left) and semi-supervised learning (on the right). Supervised learning assumes that the security evaluator first possesses a device similar to the one under attack. With this additional device, he can build a precise profiling model using a set of measurement traces and knowing the plaintext/ciphertext and the secret key of this device. In the second step, the attacker uses the beforehand profiling model to reveal the secret key of the device under attack. For this, he measures a new, additional set of traces, but as the key is secret, he has no further information about the intermediate processed data and thus builds

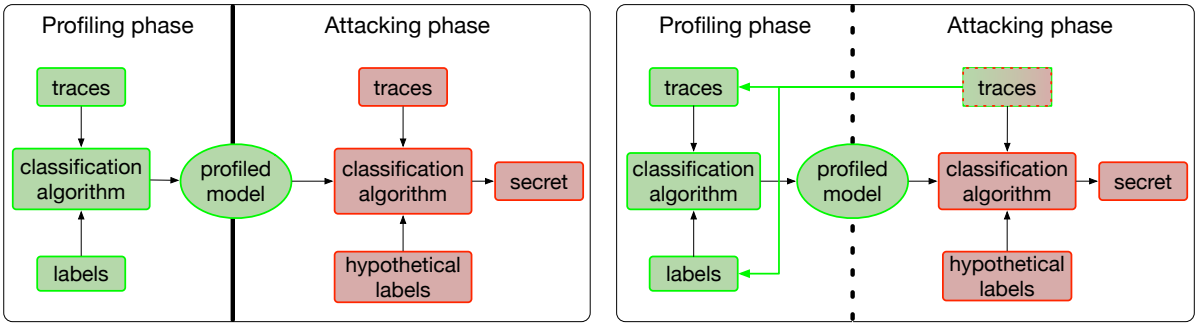


Figure 6.1: Profiling side-channel scenario: traditional (left), semi-supervised (right)

hypotheses. Accordingly, the only information the attacker transfers between the profiling and attacking phases is the profiling model he builds.

In realistic settings, the attacker is not obliged to view the profiling phase independently from the attacking phase. He can instead combine all available resources to make the attack as effective as possible. In particular, he has at hand a set of traces for which he precisely knows the intermediate processed states (i.e., labeled data) and another set of traces with an unknown secret key and thus no information about the intermediate variable (i.e., unlabeled data). To take advantage of both sets at once, we propose a new strategy of conducting profiled side-channel analysis to build a more reliable model (see Figure 6.1 on the right). This new view is of particular interest when the number of profiling traces is (very) low, and thus any additional data is helpful to improve the model estimation. An instance of practical application arises when the security analyst's access to the target device is limited, rendering it challenging to construct an accurate profiling model using a limited number of measurement traces. In such cases, using an analogous device to the target device in the profiling phase can furnish the required data to generate a more precise model. By integrating the additional data in the profiling phase, the semi-supervised approach can produce more accurate outcomes in the attacking phase. This approach can prove particularly valuable in real-world scenarios where the expense or feasibility of acquiring multiple devices for profiling is prohibitive.

To show the efficiency and applicability of semi-supervised learning for SCA, we conduct extensive experiments where semi-supervised learning outperforms supervised learning if certain assumptions are satisfied. More precisely, the results show many scenarios where the accuracy of the test set is significantly higher if semi-supervised learning is used (compared to the "classical" supervised approach). We start with the scenario that we call "extreme profiling", where the attacker has only a minimal number of traces to learn the model. From there, we increase the number of available traces, making the attacker more powerful, until we reach a setting where semi-supervised learning is no longer needed. Still, our results show that using semi-supervised learning even in these settings is not deteriorating the efficiency of attacks.

As far as we know, the only example currently implementing a semi-supervised analysis in

SCA is [139], where the authors conclude that the semi-supervised setting cannot compete with a supervised setting. Unfortunately, the assumed scenario is hard to justify; consequently, their results are expected (without much implication for SCA). More precisely, the authors compared the supervised attack with $n+m$ labeled traces for all classes with a semi-supervised attack with n labeled traces for one class and m unlabeled traces for the other unknown classes (i.e., in total $n+m$ traces). Based on such experiments, they concluded that the supervised attack is better, which is intuitive and straightforward. A proper comparison would be between the supervised attack with n traces and the semi-supervised attack with $n+m$ traces, where m is mostly smaller than n, which is the direction we take in this chapter. Also, our analysis is not restricted to only one labeled class in the learning phase.

We primarily focus on improving the accuracy if the profiling phase is limited. Since we are considering challenging scenarios, the improvements one can realistically expect are often not too significant (i.e., in the range of only a few percent). Still, we consider any progress relevant since it makes the attack easier while not requiring additional knowledge or measurements[140].

6.2 Semi-supervised Learning Types and Notation

Semi-supervised learning (SSL) is positioned between supervised and unsupervised learning. The basic idea is to take advantage of a large quantity of unlabeled data during a supervised learning procedure [141]. This approach assumes that the attacker can possess a device to conduct a profiling phase but has limited capacities. This may reflect a more realistic scenario in some practical applications, as the attacker may be limited by time and resources and face implemented countermeasures that prevent him from taking an arbitrarily large amount of side-channel measurements while knowing the secret key of the device.

Let $\vec{x} = (x_1, \ldots, x_n)$ be a set of n samples where each sample x_i is assumed to be drawn i.i.d. from a common distribution $\mathscr{X}$ with probability $P(x)$. This set $\vec{x}$ can be divided into three parts: the points $\vec{x}_l = (x_1, \ldots, x_l)$ for which we know the labels $\vec{y}_l = (y_1, \ldots, y_l)$ and the points $\vec{x}_u = (x_{l+1}, \ldots, x_{l+u})$ for which we do not know the labels. Additionally, the third part is the test set $\vec{x}_t = (x_{l+u+1}, \ldots, x_n)$ for which labels are also not known. We see that differing from the supervised case, where we also do not know labels in the test phase, unknown labels appear already in the training phase. As for supervised learning, the goal of semi-supervised learning is to predict a class for each sample in the test set $\vec{x}_t = (x_{l+u+1}, \ldots, x_n)$. One can discuss two learning paradigms for semi-supervised learning: transductive and inductive learning [142]. In transductive learning (a natural setting for some semi-supervised algorithms), predictions are performed only for the unlabeled data on a known test set. The goal is to optimize the classification performance. More formally, the algorithm makes predictions $\vec{y}_t = (y_{l+u+1}, \ldots, y_n)$ on $\vec{x}_t = (x_{l+u+1}, \ldots, x_n)$. In inductive learning, the goal is to find a prediction function defined on

the entire space $\mathscr{X}$, i.e., to find a function $f : \mathscr{X} \to \mathscr{Y}$. This function is then used to make predictions $f(x_i)$ for each sample x_i in the test set. Transductive learning is more straightforward since no general rule needs to be inferred, and, consequently, we opt to conduct it whenever possible. From the algorithm class perspective, we will use two approaches in order to achieve successful semi-supervised learning, namely: self-training [142] (Section 6.2.1) and graph-based algorithms [142, 143] (Section 6.2.2).

Although, on an intuitive level, semi-supervised learning sounds like a compelling paradigm (after all, humans learn through semi-supervised learning), the results show that it is not always the case. More precisely, it is not always possible to obtain more accurate predictions when comparing semi-supervised learning with supervised learning. Consequently, we are interested in the cases where semi-supervised learning can outperform supervised learning. For that to be possible, the following needs to hold: the knowledge on $p(x)$ one gains through unlabeled data has to carry useful information for inference of $p(y|x)$. If this is not true, semi-supervised learning will not be better than supervised learning and can lead to worse results. To assume a structure about the underlying distribution of data and to have helpful information in the inference process, we use two assumptions that should hold when conducting semi-supervised learning [142].

Smoothness Assumption. If two points x_1 and x_2 are close, then their corresponding labels y_1 and y_2 are close. The smoothness assumption can be useful for semi-supervised learning: if two points x_1 and x_2 in a high-density region are close, then so should the corresponding labels y_1 and y_2.

This assumption tells us that if two samples (measurements) belong to the same cluster, their labels (e.g., their Hamming weight or intermediate value) should be close. Note that this assumption also implies that if a low-density region separates two points, their labels need not be close. The smoothness assumption should generally hold for SCA, as the power consumption (or electromagnetic emanation) is related to the device's activity. For example, low Hamming weight or a low intermediate value should result in a low side-channel measurement.

Manifold Assumption. The high-dimensional data lie on or close to a low-dimensional manifold. If the data lie on a low-dimensional manifold, then the classifier can operate in the space of the corresponding (low) dimension.

Intuitively, the manifold assumption tells us that a set of samples is connected in some way: e.g., all measurements with the Hamming weight four lie on their manifold, while all measurements with the Hamming weight five lie on a different but nearby manifold. Then, we can try to develop representations for each of these manifolds using just the unlabeled data while assuming that the different manifolds will be represented using different learned features

of the data.

6.2.1 Self-training

In self-training (or self-learning), any classification method is selected, and the classifier is trained with the labeled data. Afterward, the classifier is used to classify the unlabeled data. From the obtained predictions, one selects only those instances with the highest output probabilities (i.e., where the output probability is higher than a given threshold σ) and then adds them to the labeled data. This procedure is repeated k times.

Self-training is a well-known semi-supervised technique and probably the most natural choice to start with [142]. The biggest drawback with this technique is that it depends on the choice of the underlying classifier and that possible mistakes reinforce themselves as the number of repeats increases. Naturally, one expects that the first step of self-learning will introduce errors (wrongly predicted classes). It is, therefore, essential to retain only those instances for which the prediction probability of the class is high. Unfortunately, a high-class prediction probability (even 100%) does not guarantee that the actual class is correctly predicted. The assumption taken by the self-training algorithm is the same as that of the underlying supervised classifier. I.e., when we use Support Vector Machine (SVM) as the classifier, we work with the manifold assumption. In contrast, if we use Naive Bayes, we use the semi-supervised smoothness assumption (alongside the independence assumption, which is a standard for Naive Bayes).

Our experiments use Naive Bayes or SVM (with RBF kernel) as classifiers. The labeling threshold is set to the value obtained by cross-validation, where a ratio between training set classification accuracy and the size of the labeled samples from the unlabeled set is optimized.

We repeat the labeling process as long as the classification accuracy on the testing set is increasing or if the samples exist where the output probability of the classifier is higher than the threshold. The second readjustment is essential because we noticed that even wrong labeling could improve the classifier generalization on the testing set. Here, we consider how well the classifier will behave on a yet unseen dataset by classifier generalization.

6.2.2 Graph-based Learning

In graph-based learning, the data are represented as nodes in graphs, where a node is both labeled and unlabeled example. The edges are labeled with the pairwise distance of incident nodes. If an edge is not labeled, it corresponds to the infinite distance. Most graph-based learning methods depend on the manifold assumption and refer to the graph using Laplacian. Let $G = (E,V)$ be a graph with edge weights given by $w : E \to \mathbb{R}$. The weight $w(e)$ of an edge e corresponds to the similarity of the incident nodes, and a missing edge means no similarity.

The similarity matrix W of graph G is defined as:

$$W_{ij} = \begin{cases} w(e) \text{ if } e = (i,j) \in E \\ 0 \text{ if } e = (i,j) \notin E \end{cases} \tag{6.1}$$

The diagonal matrix called the degree matrix D_{ii} is defined as $D_{ii} = \sum_j W_{ij}$. To define the graph Laplacian, two well-known ways are to use:

•normalized graph Laplacian $\mathscr{L} = I - D^{-1/2} W D^{-1/2}$,

•unnormalized graph Laplacian $L = D - W$.

We use a graph-based learning technique called label spreading based on normalized graph Laplacian. In this algorithm, node's labels propagate to neighbor nodes according to their proximity. Since the edges between the nodes have certain weights, some labels bear easier. Consequently, nodes close (in the Euclidean distance) are more likely to have the same labels. As the classifier within the label spreading, we use k-nearest neighbors (k-NN) (i.e., the technique of how to assign labels) since it produces a sparse matrix that one can calculate very quickly. k-nearest neighbors is the basic non-parametric instance-based learning method. The classifier has no training phase; it just stores the training set samples. In the test phase, the classifier assigns a class to an instance by determining the k instances that are the closest to it concerning Euclidean distance metric: $d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$. Here, a_r is the r-th attribute of an instance x. The class is assigned as the most commonly occurring one among the k-nearest neighbors of the test instance. This procedure is repeated for all test set instances. This simple method is advantageous and accurate in practice, especially in the cases when there are many more instances than the number of attributes and in the presence of noise (for k>1) [144]. The method is saddled with the problem of irrelevant attributes - as all of them are used in determining the distance, some irrelevant attributes may contribute to false decisions. Although one may use feature selection (and dimensionality reduction) methods to eliminate the majority of irrelevant attributes for k-NN, this usually slows the whole procedure and is not examined in detail in this work [145]. We use label spreading as implemented in Python [146], but we wrote a custom wrapper around it to better suit our requirements. Instead of using all measurements obtained from semi-supervised learning, we use only those samples with the highest classification probabilities (similar to self-training).

6.3 Experimental Setting

6.3.1 Classification algorithms

We use template attack and its pooled version, Support Vector Machines (SVM), and Naive Bayes (NB) algorithms. Those algorithms are used both in supervised and semi-supervised

scenarios. Table 6.1 presents the time and space complexities for the classification algorithms we use.

Template Attack The template attack (TA) relies on the Bayes theorem such that the posterior probability of each class value y, given the vector of N observed attribute values x:

$$p(Y = y|\vec{X} = \vec{x}) = \frac{p(Y = y)p(\vec{X} = \vec{x}|Y = y)}{p(\vec{X} = \vec{x})}, \tag{6.2}$$

where $\vec{X} = \vec{x}$ represents the event that $\vec{X}_1 = \vec{x}_1 \wedge \vec{X}_2 = \vec{x}2 \wedge \ldots \wedge \vec{X}_N = \vec{x}_N$. When used as a classifier, $p(\vec{X} = \vec{x})$ in Eq. (6.2) can be dropped as it does not depend on the class y. Accordingly, the attacker estimates in the profiling phase $p(Y = y)$ and $p(\vec{X} = \vec{x}|Y = y)$ which are used in the attacking phase to predict $p(Y = y|\vec{X} = \vec{x})$. Note that the class variable Y is discrete while the measurement X is continuous. So, the discrete probability $p(Y = y)$ is equal to its sample frequency where $p(X_i = x_i|Y = y)$ displays a density function.

Mostly in state of the art, TA is based on a multivariate normal distribution of the noise, and thus the probability density function used to compute $p(\vec{X} = \vec{x}|Y = y)$ equals:

$$p(\vec{X} = \vec{x}|Y = y) = \frac{1}{\sqrt{(2\pi)^D|\Sigma_y|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_y)^T\Sigma_y^{-1}(\vec{x}-\vec{\mu}_y)}, \tag{6.3}$$

where $\vec{\mu}_y$ is the mean over $\vec{X}$ for $1,\ldots,D$ and Σ_y the covariance matrix for each class y. The authors of [147] propose using only one pooled covariance matrix to cope with statistical difficulties resulting in low efficiency. We will use both versions of the template attack, where we denote pooled TA attack as TA_p.

Naive Bayes The Naive Bayes (NB) classifier [18] is also based on the Bayesian rule but is labeled "Naive" as it works under a simplifying assumption that the predictor features (measurements) are mutually independent among the D features, given the class value. The existence of highly-correlated features in a dataset can influence the learning process and reduce the number of successful predictions. Also, NB assumes a normal distribution for predictor features. NB classifier outputs posterior probabilities due to the classification procedure [18]. The Bayes' formula is used to compute the posterior probability of each class value y given the vector of N observed feature values x.

Support Vector Machines Support Vector Machine (SVM) is a kernel-based machine learning technique used to accurately classify linearly separable and linearly inseparable data [19]. The SVM algorithm is parametric and deterministic. When the data are not linearly separable, the basic idea is to transform them into a higher dimensional space using a transformation ker-

Table 6.1: Time and space complexities. N is the number of samples in the training set, M is the number of samples in the test set, D is the number of attributes, $|\mathscr{Y}|$ is the number of classes of the target attribute, and v is the average number of values for a feature.

Alg.	Training		Testing	
	Time	Space	Time	Space
TA	$O(ND^2)$	$O(\|\mathscr{Y}\|D^2v)$	$O(\|\mathscr{Y}\|D^2)$	$O(\|\mathscr{Y}\|D^2v)$
k-NN	$O(1)$	$O(ND)$	$O(M(ND+kN))$	$O(ND+MD)$
NB	$O(ND)$	$O(ND)$	$O(\|\mathscr{Y}\|D)$	$O(MD)$
SVM	$O(N^3D)$	$O(N^2D)$	$O(MND)$	$O(N^2D)$

nel function. In this new space, the samples can usually be classified more accurately. Many kernel functions have been developed, with the most used being polynomial and radial-based.

6.3.2 Datasets

We use two datasets that mainly differ in the amount of noise and the side-channel leakage distribution – DPAcontest v2 [148] and DPAcontest v4 [149]. We do not consider the variations in the number of available points of interest (features) since the number of scenarios would become quite large in such a case. We select 50 points of interest with the highest correlation between the class value and data set for all the analyzed data sets and investigate scenarios with a different number of classes: 9 classes and 256 classes. Points of interest represent the attributes of traces.

Calligraphic letters (e.g., $\mathscr{X}$) denote sets, capital letters (e.g., X) denote random variables taking values in these sets, and the corresponding lowercase letters (e.g., x) denote their realizations. Let k^* be the fixed secret cryptographic key (byte) and the random variable T the plaintext or ciphertext of the cryptographic algorithm, which is uniformly chosen. The measured leakage is denoted as X, and we are particularly interested in multivariate leakage $\vec{X} = X_1, \ldots, X_D$, where D is the number of time samples or features (attributes) in machine learning terminology.

Considering a powerful attacker with a device with knowledge about the secret key implemented, a set of N profiling traces $\vec{X}_1, \ldots, \vec{X}_N$ is used to estimate the leakage model beforehand. Note that this set is multi-dimensional (i.e., it has a dimension equal to $D \times N$). In the attack phase, the attacker then measures additional traces $\vec{X}_1, \ldots, \vec{X}_Q$ from the device under attack to break the unknown secret key k^*.

DPAcontest v2 [148] DPAcontest v2 provides measurements of an AES hardware implementation. Previous works showed that the most suitable leakage model (when attacking the last

round of an unprotected hardware implementation) is the register writing in the last round, i.e.:

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[C_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}}, \tag{6.4}$$

where C_{b_1} and C_{b_2} are two ciphertext bytes, and the relation between b_1 and b_2 is given through the inverse ShiftRows operation of AES. In particular, we choose $b_1 = 12$ resulting in $b_2 = 8$ as it is one of the easiest bytes to attack*. In Eq. (6.4) $Y(k^*)$ consists of 256 values. As an additional model, we applied the Hamming weight (HW) to this value resulting in 9 classes. These measurements are relatively noisy and the resulting model-based signal-to-noise ratio $SNR = \frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$, lies between 0.0069 and 0.0096. We use the measurements from the "template" part of the database.

DPAcontest v4 [149] The 4th version provides measurements of a masked AES software implementation. However, as the mask is known, one can quickly turn it into an unprotected scenario. Though it is a software implementation, the most leaking operation is not the register writing but the processing of the S-box operation, and we attack the first round. Accordingly, the leakage model changes to

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}, \tag{6.5}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Again we consider the scenario of 256 classes and nine classes (considering $HW(Y(k^*))$). Compared to the measurements from version 2, the model-based SNR is much higher and lies between 0.1188 and 5.8577.

6.3.3 Dataset Preparation

We experiment with randomly selected 20 000 measurements (profiled traces) from DPAcontest v2 and DPAcontest v4 datasets. These measurements are divided into 2:1 ratio for training and testing sets (i.e., 13 000 in total for training with or without semi-supervised learning and 7 000 for testing). When using supervised learning, the training datasets are divided into ten stratified folds and evaluated by a 10-fold cross-validation procedure. For semi-supervised learning, we divide the training dataset into a labeled set of size l and an unlabeled set of size u, as follows:

- (100+12.9k): $l = 100$, $u = 12\,900 \rightarrow 0.77\%$ vs 99.23%
- (250+12.75k): $l = 250$, $u = 12\,750 \rightarrow 1.93\%$ vs 98.07%
- (500+12.5k): $l = 500$, $u = 12\,500 \rightarrow 3.85\%$ vs 96.15%
- (1k+12k): $l = 1\,000$, $u = 12\,000 \rightarrow 7.69\%$ vs 92.31%

*see e.g., in the hall of fame on [148]

•(3k+10k): $l = 3\,000$, $u = 10\,000 \rightarrow 23.08\%$ vs 76.92%
•(5k+8k): $l = 5\,000$, $u = 8\,000 \rightarrow 38.46\%$ vs 61.54%
•(7k+6k): $l = 7\,000$, $u = 6\,000 \rightarrow 53.85\%$ vs 46.15%
•(10k+3k): $l = 10\,000$, $u = 3\,000 \rightarrow 76.92\%$ vs 23.08%

6.4 Experimental Results

As the primary performance measure, we use the accuracy, i.e., the percentage of correctly classified instances: $ACC = \frac{TP+TN}{TP+FP+TN+FN}$.

In supervised learning, the classifiers are built on the labeled sets and estimated on the unlabeled sets. We give results here only for the results obtained from the testing phase. When discussing semi-supervised learning, we first learn the classifiers on the labeled sets. Then, we learn with the labeled set and unlabeled set in several steps, where in each step, we augment the labeled set with the most confident predictions from the unlabeled set. Once we cannot add more measurements, we finish the learning phase. Finally, we conduct the estimation phase on a different unlabeled set.

For machine learning techniques that have parameters to be tuned, we conducted a tuning phase on labeled sets and used such tuned parameters in consequent experimental phases. For SVM with radial kernel, we select C equal to 10 and γ equal to 0.6 for DPAcontest v4 and C equal to 2 and γ equal to 0.05 for DPAcontest v2. A low cost of the margin parameter C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. The radial kernel parameter γ defines how much influence a single training example has, where the larger γ is, the closer other examples must be to be affected. When using k-NN with label spreading, we select k as equal to 7. Naive Bayes and template attacks do not have parameters to tune.

For semi-supervised learning, we tune the σ parameter. Table 6.2 states all the threshold levels σ for different classifiers for all scenarios we consider, which were set at the labeled set training accuracy. In all experiments, when there is a single result with the best accuracy, we depict it in bold style.

6.4.1 DPAcontest v2 Dataset Results

In Table 6.3, we give the results for the testing phase for supervised learning vs. semi-supervised learning methods for the DPAcontest v2 dataset.

Overall, the accuracies are low for all scenarios, which is expected due to the high amount of noise. Supervised learning results serve as a baseline when compared with semi-supervised learning. Using only 100 traces in the profiling phase can be considered the worst-case scenario while using all 13 000 in the profiling phase can be viewed as the best-case scenario. A natural

Table 6.2: Threshold levels. When considering SVM threshold level σ for DPAcontest v4, both 9 and 256 classes scenarios use the same value. This is because the problem is "simple" for 9 classes and the threshold can be set to a higher value but we noticed no difference in performance. The same behavior is not observed for DPAcontest v2.

	DPAcontest v4		DPAcontest v2	
Classifier	9 classes	256 classes	9 classes	256 classes
NB, k-NN	0.99	0.99	0.99	0.99
SVM	0.22	0.22	0.01435	0.004

assumption is that adding more measurements in the semi-supervised phase helps when having a tiny number of traces in the profiling phase. Still, as we assume there will be at least some portion of measurements incorrectly classified during semi-supervised learning, we cannot expect semi-supervised learning to be more successful than supervised learning with all traces.

For the nine classes scenario, we can notice an interesting behavior for the smaller numbers of measurements, e.g., up to 1 000 measurements with Naive Bayes and TA_p techniques. We see that the accuracies are higher than those with more measurements. Namely, since there is a minimal number of traces in the profiling phase, some classes do not have correctly trained representatives. For the scenario with 100 measurements, we see no instances of HW 0, HW 1, HW 7, and HW 8 classes present in the profiling phase. Consequently, the classifiers do not work anymore with nine classes but only five classes, making it a much simpler classification problem. Although such results look good, they are not very helpful in the SCA context to reveal the secret key.

The most extreme cases behave worse than supervised learning when considering ML techniques. This is somewhat expected, since the noise level is high, and it is challenging to form "good" clusters with very few labeled measurements. For the cases where the amount of labeled measurements is higher than 5 000, we see improvements with SSL, which clearly indicates that more measurements are necessary for SSL if the data is noisy. Unfortunately, not all cases for TA become stable. The analysis shows that some classes are underrepresented, which makes covariance matrices unstable.

The behavior for label spreading is similar to that of self-training, but we see somewhat worse accuracies throughout all scenarios. A smaller number of measurements for nine classes have higher accuracies than more measurements. This is due to a lack of labeled examples of all classes, which corresponds to a more manageable problem since the classification process has fewer classes to choose from. When considering 256 classes, interestingly, TA works better with SSL than with supervised learning in many cases. Still, the accuracies are very low, as expected, since we work with a highly noisy scenario and many classes.

Table 6.3: Testing results, supervised learning vs. semi-supervised learning approaches, DPAcontest v2, (ACC, %).

Size	NB	SVM	TA	TA_p
	9 classes, supervised learning / SSL:self-learning / SSL:label spreading			
100/+12.9k	**20.6**/14.5/11.8	**21.6**/18.7/18.1	0.4/**7.3**/5.9	**17.7**/17/15.6
250/+12.75k	10.4/**12.3**/11.3	**21**/20.8/19.8	**10.2**/0.5/0.4	15.8/**16.2**/15.1
500/+12.5k	10.8/**12.9**/11.8	**22**/21.4/21	**5.5**/1.4/1.1	15.3/**17.6**/16.7
1k/+12k	11.9/**12.7**/12.3	23.8/**25.2**/25.1	**3.8**/0.4/0.4	13.8/**14.9**/13.1
3k/+10k	**7.3**/7/6.8	24.5/**25**/24.9	**8.7**/1.5/1.4	10.4/**12.4**/12
5k/+8k	9.3/**10.3**/9.4	24.6/**25.5**/25.1	0.9/**15**/14.1	8.9/**11.8**/10.9
7k/+6k	8.8/**11**/10.4	25.5/**26**/26	**2.1**/1.7/1.3	8.4/**11.1**/10.9
10k/+3k	8.8/**10.2**/10	25.3/**26.2**/26	7.5/**8.3**/6.9	8.2/**8.6**/8
13k	8.3	26.2	15	7.6
	256 classes, supervised learning / SSL:self-learning / SSL:label spreading			
100/+12.9k	0.3/**0.6**/0.5	0.4/**0.5**/0.4	0.3/**0.5**/0.3	0.4/0.4/0.4
250/+12.75k	0.4/**0.7**/0.5	0.4/0.5/0.5	**0.6**/0.5/0.4	0.4/0.4/0.4
500/+12.5k	0.4/**0.6**/0.5	0.4/**0.5**/0.4	0.4/**0.6**/0.5	0.4/0.4/0.4
1k/+12k	0.5/0.5/0.5	0.4/0.4/0.3	0.4/**0.6**/0.5	0.5/0.5/0.5
3k/+10k	0.4/0.4/0.4	0.4/**0.5**/0.4	0.3/**0.4**/0.3	0.4/0.4/0.4
5k/+8k	**0.6**/0.5/0.4	**0.5**/0.4/0.4	0.4/0.4/0.4	**0.5**/0.4/0.4
7k/+6k	**0.7**/0.6/0.5	0.5/0.6/0.6	**0.5**/0.4/0.4	0.4/0.4/0.4
10k/+3k	**0.6**/0.5/0.5	0.5/0.5/0.5	0.4/0.4/0.4	0.4/0.4/0.4
13k	0.6	0.5	0.4	0.3

6.4.2 DPAcontest v4 Dataset Results

In Table 6.4, we give the results for the testing phase for supervised learning vs. semi-supervised learning approaches for DPAcontest v4 dataset.

Here, we see far better results compared to DPAcontest v2. We observe that for both 9 and 256 classes, machine learning techniques work well. SVM performs better than Naive Bayes, which is expected since SVM is a more powerful classification technique. When considering template attacks, we see that the pooled version is significantly better since it does not have a problem with covariance matrix instability. As it can be seen, for 13 000 measurements and nine classes, TA and TA pooled perform similarly, which is a strong indication that the covariance matrices got stable and that if there is a further increase in the number of measurements, TA may outperform the pooled version.

Particularly interesting, we highlight the efficiency of ML techniques even in scenarios with only 100 to 500 measurements. This leads us to conclude that ML is a potent option to be used even in the most extreme profiling cases, provided that the noise level is not too high. When considering nine classes and SSL, only for the case with 100 measurements, the results for ML are slightly worse when compared with supervised learning. The other results are either better or comparable. What is most interesting, TA and TA_p results are significantly better than those obtained with supervised learning. In particular, the accuracy for TA and TA_p increases for all scenarios regardless of the number of added unlabeled measurements. For TA, the explanation is simple but with profound consequences. By adding more measurements, we can resolve instabilities in estimating the covariance matrices, and consequently, the accuracy of TA is significantly increasing. The highest increase (more than 73.3%) can be observed for TA using 10k labeled measurements and 3k unlabeled ones. Interestingly, we see that for TA and TA_p, using 10k+3k is approximately as efficient as using 13k labeled traces. The highest increases for TA_p can be observed in the first four scenarios (up to 12 000 additional unlabeled measurements). Afterward, the accuracy is higher than in the supervised scenario, but the margin gets smaller. For 256 classes, the results for Naive Bayes are better for the most extreme cases (i.e., up to 500 labeled measurements) but slightly worse for the other scenarios. Similar behavior can be seen for TA_p. With label spreading, we see that the results are generally worse than for self-training. When considering nine classes, the first case with 100 labeled measurements significantly drops in accuracy compared to the supervised case or self-training. The rest of the results for nine classes are comparable with the results obtained with self-training. What is important to notice are the cases $1k+12k$ and $5k+8k$, where TA is not stable, and, consequently, the results are much worse than for self-training. Scenarios with 256 classes are again similar, but we note that there are no cases where label spreading outperforms self-training.

On a more general level, one could ask if a slight increase in accuracy is significant from a

Table 6.4: Testing results, supervised learning vs. semi-supervised learning approaches, DPAcontest v4, (ACC, %).

Size	NB	SVM	TA	TA_p
	9 classes, supervised learning / SSL:self-learning / SSL:label-spreading			
100/+12.9k	**61.5**/59/30	**69.1**/69/25	0.3/**58.9**/18.8	45.4/**67.6**/21.1
250/+12.75k	64.3/64.6/**65.3**	78.4/**78.2**/77.5	0.3/12.6/**61.4**	53/**75.2**/71.3
500/+12.5k	65.9/**66.2**/65.5	82.7/**82.8**/81.1	0.3/56.6/**58.8**	68.9/**76.9**/74.5
1k/+12k	64.8/**68.1**/67.7	86.6/**87.1**/84.1	1.3/**44.2**/7.1	73.1/**78.3**/76.6
3k/+10k	67.2/68.3/**68.7**	90.8/90.5/**91.8**	5.2/53/**66.6**	74.9/**78.1**/77.4
5k/+8k	67.9/68.1/**68.8**	92/**92.3**/91.8	2.8/**46.4**/3.2	75.8/**78.4**/78
7k/+6k	68/**68.4**/68.6	**92.8**/92.7/92.5	11.2/**75.6**/14.8	76.5/**78**/77.9
10k/+3k	68.1/68.7/68.7	93.3/**93.6**/93.5	0.4/**73.8**/49.6	77.2/77.9/**78**
13k	68.4	93.7	75.3	77.7
	256 classes, supervised learning / SSL:self-learning / SSL:label-spreading			
100/+12.9k	1.5/**2.7**/1.7	**5.1**/4.2/3.7	0.3/0.3/0.3	0.4/**3.4**/2.6
250/+12.75k	2.2/**3.1**/3	**6.8**/6.4/6.1	0.3/0.3/0.3	3.3/**3.7**/3.5
500/+12.5k	4.9/5.7/5.7	**10.3**/8.5/7.9	0.4/**0.5**/0.4	6.4/**7.1**/7
1k/+12k	**10.5**/9.3/8.5	**13.6**/12.8/11	0.4/**0.5**/0.4	**10.2**/9.5/9
3k/+10k	**16.5**/15.6/15	**22.4**/21.7/18.7	0.1/**0.4**/0.3	**16.3**/15.5/14.8
5k/+8k	**18**/17.3/16	**27.4**/25.7/24.8	**0.2**/0.1/0.1	**19.2**/18.7/17.2
7k/+6k	**19.5**/18.4/17	**30**/29/26.9	**0.3**/0.1/0.1	20.6/**21**/20.1
10k/+3k	**20.1**/19.6/18.1	**33.3**/32.8/28.8	0/0.2/0.2	**22.5**/22.4/21.9
13k	20.2	34.9	0.1	23.7

practical (attack) perspective. We believe it does, since 1) with SSL, it requires no additional knowledge except the one already used in profiling attacks, and 2) as shown in [150], even a slight difference in accuracy can translate to a significant difference is guessing entropy or success rate.

6.5 Conclusions

In this chapter, we aim to explore the application of semi-supervised learning (SSL) to profiled side-channel analysis (SCA), which has traditionally been considered a two-step process involving the transfer of the profiled model between the profiling and attacking phases. Our investigation focuses on scenarios where the attacker is restricted in the profiling phase but has access to additional information from the attacking measurements to build the profiled model. We examine two approaches to SSL under varying noise levels, the number of prediction classes, and measurement counts in the profiling phase. The two machine learning techniques of Naive Bayes and SVM, template attack and its pooled version, are used as side-channel attack techniques.

Our results demonstrate the efficacy of SSL in many scenarios. In particular, the template attack and its pooled version significantly improved the low-noise scenario. Furthermore, the addition of unlabeled samples from the attacking phase enhanced the estimation of the covariance matrices, resulting in improvements of over 70%. We also observed that the higher the number of samples in the profiling phase, the less significant the impact of the added unlabeled samples from the attacking phase. While the improvements were minor in the scenario with high noise, this can be attributed to the inherent difficulty of such scenarios, and the results do not significantly deteriorate compared to standard profiling.

This chapter sheds light on the potential of SSL in enhancing the accuracy of profiled SCA in scenarios where access to multiple devices for profiling is limited. It also highlights the importance of considering SSL as a viable alternative to standard profiling in designing SCA attacks.

Chapter 7

Neuroevolution in Side-channel Analysis

The impact of activation functions on neural network performance is a topic of great importance in machine learning. While there have been efforts to develop novel activation functions, Rectified Linear Unit (*ReLU*) is the most widely used in practice. This chapter proposes using evolutionary algorithms to discover new activation functions for side-channel analysis (SCA) that outperform *ReLU* when using the same network architecture. The proposed method uses Genetic Programming (GP) to define and explore candidate activation functions, representing the first attempt to develop custom activation functions for SCA. Results from experiments conducted on the ASCAD database show that the proposed approach is highly effective compared to state-of-the-art neural network architectures. Furthermore, the evolved activation functions demonstrate the property of generalization, exhibiting high performance across different SCA scenarios.

The remainder of this chapter is structured as follows. Section 7.1 provides an overview of the chapter's contributions and the motivation for this research. Section 7.2 presents the necessary definitions and concepts related to activation functions and evolutionary algorithms. Section 7.3 discusses related works in the field. Section 7.4 describes the datasets and parameters considered in our experiments. Section 7.5 presents the results obtained from our experiments. Finally, Section 7.6 summarizes the main contributions of this chapter and outlines possible directions for future research.

7.1 Introduction

Modern digital systems are commonly equipped with cryptographic primitives, acting as the foundation of security, trust, and privacy protocols. While such primitives are proven to be mathematically secure, poor implementation choices can make them vulnerable to attackers. Such vulnerabilities are commonly known as side-channel leakage [50]. Side-channel leakage exploits various sources of information leakage in the device where some common examples

of leakage are timing [151], power [152], and electromagnetic (EM) emanation [153]. The researchers proposed several side-channel analysis (SCA) approaches to exploit those leakages in the last few decades. One standard division of side-channel analyses is into non-profiling and profiling attacks. Non-profiling attacks like Simple Power Analysis (SPA) [154] or Differential Power Analysis (DPA) [52] require fewer assumptions but could need thousands of measurements (traces) to break a target, especially if it is protected with countermeasures. On the other hand, profiling attacks are considered one of the strongest possible attacks [155]. The attacker has complete control over a clone device, which can build its profile. The attacker then uses this profile to target other similar devices to recover the secret information. The deep learning approaches represent a powerful (and more recent) option for profiling SCA. Indeed, the results in the last few years show the potential of such an approach where neural networks like multilayer perceptron (MLP and convolutional neural networks (CNNs) can break targets protected with countermeasures [156, 157]. Still, finding high-performing neural network architectures is often not accessible due to a large number of hyperparameters to consider. We can distinguish between two rather different approaches in the hyperparameter tuning phase. The first approach considers various techniques to select the best-performing hyperparameters [158, 159, 160, 161, 162, 163]. Common techniques include gradient descent, Bayesian hyperparameter optimization, reinforcement learning, and evolutionary algorithms [159, 164, 165]. The second direction considers the design of custom neural network elements. Along with the topology and loss function, the activation function's choice is essential in determining how a neural network learns and acts. A well-defined activation function can be any nonlinear function that transforms a layer's output in a neural network.

Several different activation functions are widely used in modern neural network architectures. For instance, the Rectified Linear Unit, $ReLU(x) = max\{x, 0\}$, is popular because it is simple and effective. Other activation functions such as $tanh(x)$ and $\sigma(x) = 1/(1+e^{-x})$ are commonly used when it is useful to restrict the activation value within a certain range, such as in recurrent neural networks for language modelling [166]. There have also been attempts to engineer new activation functions with certain properties. For example, Leaky ReLU [167] allows information to flow when $x < 0$. On the other hand, $Softplus$ [168] is positive, monotonic, and smooth. Many hand-designed activation functions exist [169], but none achieved widespread adoption like $ReLU$. Still, there is a significant (unused) potential in designing custom activation functions for neural network design.

There are a lot of attempts to optimize network architectures, hyperparameters, learning rates, or loss functions [158, 159, 160, 161, 162, 163], rather than directly optimize just a single model in machine learning. Several techniques for meta-learning have been proposed, including gradient descent, Bayesian hyperparameter optimization, reinforcement learning, and evolutionary algorithms [159, 164, 165]. Among these, evolutionary algorithms are the most

versatile and can be applied to several aspects of neural network design.

This chapter investigates an evolutionary approach to evolving activation functions for side-channel analysis. We build upon recent results considering deep learning architectures. We ask whether it is possible to make deep learning-based SCA even more efficient if activation functions in a neural network are optimized for a particular problem (neural network architecture, side-channel leakage model, and dataset). More precisely, we use Genetic Programming (GP), where we represent activation functions as syntactic trees, and we evolve custom expressions.

The resulting functions are unlikely to be discovered manually, yet they perform (surprisingly) well, surpassing traditional activation functions like *ReLU* on common side-channel measurements, like those in the ASCAD database. To the best of our knowledge, this is the first time that neuroevolution has been used for SCA or evolutionary algorithms have been used to develop activation functions for SCA.

In this chapter, we realized the original scientific contribution of the neuroevolutionary procedures for optimization of neural network architecture in the security domain as follows[170]:

1. We evolve novel activation functions used in multilayer perceptron and Convolutional Neural Network. Neural networks with those activation functions perform better than existing relevant research. This shows that the newly developed activation functions have their place in the future designs of neural network architectures for SCA.
2. We replace popular activation functions with evolved activation functions in previously developed neural network topologies and demonstrate better network performance after this substitution. This shows that optimization of the activation function has relevance even when considering already developed neural networks.

We consider experiments on two datasets, two leakage models, two types of neural networks, and many specific scenarios.

7.2 Background

7.2.1 Notation

Let calligraphic letters ($\mathscr{X}$) denote sets and the corresponding upper-case letters (X) random variables and random vectors $\mathbf{X}$ over $\mathscr{X}$. The corresponding lower-case letters x and $\mathbf{x}$ denote realizations of X and $\mathbf{X}$, respectively. We denote the key candidate as k where $k \in \mathscr{K}$, and k^* represents the correct key.

We define a dataset as a collection of traces (measurements) $\mathbf{D}$. Each trace $\mathbf{x}_i$ is associated with an input value (plaintext or ciphertext) $\mathbf{i}_i$ and a key $\mathbf{k}_i$. To access a specific trace or input value, we use the index i. We divide the dataset into three parts: profiling set consisting of N traces, validation set consisting of V traces, and attack set consisting of Q traces.

We denote the vector of learnable parameters in our profiling models as θ and the set of hyperparameters defining the profiling model as $\mathscr{H}$. We consider the supervised machine learning task (classification), where the goal is to predict the class value $v \in V$ for an input x. The size of the set V equals c.

7.2.2 Machine Learning-based SCA

We consider a typical profiling side-channel analysis setting with two phases: training (profiling) and testing (attack). A powerful attacker has a device (clone device) with knowledge about the secret key. The attacker can obtain a set of N profiling traces $x_1, \ldots, x_N$ (where each trace corresponds to the processing of plaintext or ciphertext i).

- The profiling phase aims to learn θ' that minimizes the empirical risk represented by a loss function L on a profiling set of size N.
- The goal of the attack phase is to make predictions about the classes:

$$y(x_1, k^*), \ldots, y(x_Q, k^*),$$

where k^* represents the secret (unknown) key on the device under the attack.

We consider an attack on a block cipher (the AES cipher) and conduct the multi-class classification task. More precisely, we learn a function f that maps an input to the output ($f : \mathscr{X} \rightarrow Y$)) based on examples of input-output pairs, where the number of classes c is determined by the leakage model. The function f is parameterized by $\theta \in \mathbb{R}^n$, where n denotes the number of trainable parameters.

Based on the class predictions, we estimate the effort required to reveal the secret key k^*. A common result of predicting with a model f on the attack set is a two-dimensional matrix P with dimensions equal to $Q \times c$. Every element $\mathbf{p}_{i,v}$ of matrix P is a vector of all class probabilities for a specific trace $\mathbf{x}_i$. The probability $S(k)$ for any key byte candidate k is used as a log-likelihood distinguisher:

$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{7.1}$$

The value $\mathbf{p}_{i,v}$ denotes the probability that for a key k and input i_i, the result is class v (derived from the key and input through a cryptographic function and a leakage model l).

Finally, to estimate the effort required to break the secret key, it is common to use the guessing entropy (GE) [171] metric. An attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathscr{K}|}]$ in decreasing order of probability given Q traces in the attack phase. Here, g_1 is the most likely key candidate and $g_{|\mathscr{K}|}$ is the least likely key candidate. Guessing entropy is the average position of k^* in $\mathbf{g}$. Our work only considers attacks on specific key bytes, formally using the partial guessing entropy metric. Still, due to simplicity, we denote it as guessing entropy.

7.2.3 Activation Function as a Tree

In our work, each activation function is represented as a tree consisting of unary and binary operators. The terminals in the tree represent the function input values, while the root node's value represents the output function value. Each individual in the GP population is a potential candidate activation function. An individual is evaluated using the activation function it embodies in a neural network. The individual's fitness is then defined as the neural network's performance applied to a specific task - in this case, the key prediction efficiency.

7.3 Related Work

One perspective of hyperparameter tuning is finding better (custom) activation functions for SCA. In SCA, most works have considered hyperparameter tuning but using standard options (i.e., not designing new neural network elements). There, we can enumerate several phases in profiling SCA and hyperparameter tuning. The first approaches in profiling SCA like template attack [155]), or machine learning-based attacks (random forest [172], support vector machines [135, 173], Naive Bayes [174]) had only a few or even none hyperparameters to tune.

In 2016, Maghrebi et al. introduced convolutional neural networks for profiling SCA [175]. The authors also reported they used genetic algorithms to tune the hyperparameters. While it is difficult to know whether this is the first time deep learning was used in SCA (many works omitted details about neural network architectures), this work represented a significant turning point in SCA research. Indeed, the SCA community moved its attention from other profiling methods to (almost exclusively) deep learning from this moment.

As a result, multiple research works report outstanding attack performance even in the presence of countermeasures [176, 177]. Interestingly, the first work did not discuss hyperparameter tuning, while the second conducted manual hyperparameter tuning. Kim et al. constructed VGG-like architecture that performs well over several datasets, but they did not discuss the hyperparameter tuning involved in checking the performance of such an architecture [156]. Benadjila et al. made an empirical evaluation of different CNN hyperparameters for the ASCAD dataset [178]. Perin et al. used a random search in predefined ranges to build deep learning models to form ensembles [179]. Both works reported excellent results, despite relatively simple methods for choosing hyperparameters. Wu et al. proposed to use Bayesian optimization to find optimal hyperparameters for MLP and CNN architectures [180]. Their results indicated it is possible to find excellent architectures and that even a random search can find many architectures that exhibit top performance. Rijsdijk et al. explored how reinforcement learning can be used for hyperparameter tuning for CNNs [181]. They reported very good results (attack performance with small neural network architectures), but their approach requires significant computational resources.

Besides improving the neural network performance by conducting efficient hyperparameter tuning, several works aim to provide a methodology to build neural networks for SCA. Zaid et al. proposed a method to select hyperparameters related to the size (number of learnable parameters, i.e., weights and biases) of layers in CNNs. The authors considered the number of filters, kernel sizes, strides, and the number of neurons in fully-connected layers [157]. Wouters et al. [182] improved upon the work from Zaid et al. [157] and discussed several problems in the original work. Wouters et al. showed how to reach similar attack performance with significantly smaller neural network architectures.

Finally, several works investigate improving the performance of deep learning-based SCA by designing custom neural network elements. Pfeifer and Haddad developed a new type of layer called "Spread", and they claimed it reduces the number of layers required and speeds up the learning phase [183]. Zheng et al. proposed a new metric function called Cross Entropy Ratio (CER), where they adapted it to a new loss function designed specifically for deep learning in SCA [184]. Zaid et al. introduced a new loss function derived from the learning to rank approach that helps to prevent approximation and estimation errors [185].

On the other hand, several papers investigate the evolution of activation functions by using evolutionary algorithms and machine learning. One can find a first attempt to learn activations in a neural network in [186], where the authors propose randomly adding or removing logistic or Gaussian activation functions using genetic programming. In [187], the authors developed a method to choose an activation function for each neural network layer automatically. Hagg et al. augmented the NEAT algorithm [188] to evolve simultaneously, except for the overall network topology, and per-neuron activation functions [189]. Both works used predefined lists to select activation functions. Ramachandran et al. automatically designed novel activation functions using reinforcement learning [190]. There, the authors discovered several new, high-performing activation functions, but they analyzed just one in detail: $x \cdot \sigma(x)$, which they call Swish. Bingham et al. augmented previous research by introducing an evolutionary algorithm to design novel activation functions [191]. The authors showed that it is possible to evolve specialized activation functions that perform well for the CIFAR-10 and CIFAR-100 datasets. In [192], the authors use a hybrid genetic algorithm to evolve a function defined differently on the positive and negative domains. Parts of the function are represented by trees and crossed by special operators that separately change the positive and negative sides. The nodes comprise basic arithmetic operations, and leaves are popular activation functions without constants. The authors also presented the new activation functions ELiSH and Hard ELiSH, which they built manually, intending to combine smaller functions' good properties. On three datasets, they showed that their functions perform the best.

Finally, we discuss related works investigating activation functions designed manually. Nair and Hinton introduced rectified linear unit (*ReLU*) and argued it to be a better model than the lo-

gistic sigmoid activation function [193]. Slight modifications of *ReLU* have been proposed over the years, such as leaky *ReLU* (*LReLU*), which deals with the issues with dead neurons [194]. More variations of *ReLU* can be found in [195, 196]. Clevert et al. experimented with exponential linear unit function (*ELU*), which reduces the vanishing gradient problem [197]. Furthermore, Klambauer et al. extended properties of *ELU* with scaled exponential linear unit function (*SELU*) [198]. The authors in [199, 200] proposed to use a combination of different activation functions in the same layer. However, this approach has memory issues, which is typically a critical parameter in real-world scenarios.

7.4 Experimental Setup

In this section, we first discuss the datasets and leakage models we consider. Afterward, we give details about the investigated approaches to design activation functions.

7.4.1 Datasets and Leakage Models

In our experiments, we consider two versions of the ASCAD database [178]. This database contains the measurements from an 8-bit AVR microcontroller running a masked AES-128 implementation. This database is publicly available from `https://github.com/ANSSI-FR/ASCAD`.

The first version of the ASCAD database has a fixed key and consists of 50 000 traces for profiling and 10 000 for the attack. The traces in this dataset have 700 features (preselected window when attacking the third key byte). We use 45 000 traces for training and 5 000 for validation from the original training set. The second version of the ASCAD database has random keys, and the dataset consists of 200 000 traces for training and 100 000 for testing. Each trace in this database has 1 400 features (preselected window for the third key byte). We use 5 000 traces from the original training set for validation. We normalize the input features to a Gaussian distribution (zero mean and unit variance) for both datasets by calculating the training set's distribution parameters.

This chapter considers the Hamming Weight (HW) and Identity (ID) leakage models. In the HW leakage model, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. This leakage model results in nine classes when considering a cipher that uses an 8-bit S-box. Since this induces a heavy imbalance in the label distribution, we additionally calculate the imbalance weights that balance the estimated model loss. We follow the guidelines for calculating the imbalance weights in [177]. For the ID leakage model, the attacker considers the leakage as an intermediate value of the cipher. Considering an 8-bit S-box, this leakage model results in 256 classes (values between 0 and 255).

7.4.2 Architecture Search Strategies

Our experiments consider two neural network types: CNN and MLP. The specific CNN architecture is described in [157] for the ASCAD synchronized dataset, and we use it with all its reported hyperparameters. The procedure's seed values were not defined, so we translated their original architecture and training procedure from Keras to PyTorch and ran the process with several seed values until we observed an equal or better result than the original paper. The exact hyperparameters are reported in Section 7.5. We used the same seed value throughout our experiments for this architecture. Following their hyperparameter setup, the authors implemented a one-cycle learning rate schedule, which we replaced with the implementation in PyTorch. The network architecture consists of a convolutional layer with four output channels, followed by batch normalization, activation function, and average pooling. The output of this block is flattened and fed to an MLP tail to produce the final prediction. The width and depth of the MLP tail were optimized per dataset with the grid search. In the CNN training procedures, we use the one-cycle policy for learning rate with reported hyperparameters of learning rate 0.005, with 40% of the cycle incrementing the value using a linear annealing strategy.

Since CNN inference time can be quite long compared to MLP, we apply and compare both architectures. The MLP architecture is defined with consecutive blocks consisting of a dense linear layer, batch normalization layer, and a nonlinear activation function.

We first employ architecture search to find the representative architecture for each dataset (and neural network types). Two different algorithms were used to explore the space of network architectures and their hyperparameters: grid search and random search. Both techniques include evaluating a multitude of points of search space to find the optimal one. The points are evaluated on the test set to obtain a distribution of representative solutions that will later serve for further optimization with evolution. The techniques are also easily parallelized, allowing a much faster search than sequential evaluation. We consider grid search for CNNs as the number of hyperparameters is large, making it more difficult for a random search. For MLP, we use random search as related works reported good results even with such a simple tuning setup [179, 180].

Grid Architecture Search for CNNs

Grid search evaluates all possible combinations of parameter values for a given search space, where the continuous variables are sampled along with fixed steps. We employ this search strategy following [157] to find an optimal CNN architecture for a given dataset. Due to time constraints, we slightly truncated this search space by removing hyperparameter values that were not expected to provide good results (such as very shallow or narrow architectures). The considered hyperparameter space is described in Tables 7.1 and 7.2 and resulted in 2160 grid

samples to obtain the best convolutional model (CNN - GS_{best}) for each of the datasets.

Random Architecture Search for MLP

The random search strategy samples the given space by randomly selecting points in the search space. The search space can be defined with a multitude of features, both discrete and continuous. It is similar to grid search but without a structured sampling of continuous spaces, which might introduce bias by selecting only a subset of possible values. We find this bias unwanted as we do not use a learning rate schedule for our MLP models, and the model might be more sensitive to a fixed learning rate's exact value. We define the search space as the collection of hyperparameters that affect the shape of MLP architecture and its train parameters, listed in Tables 7.1 and 7.2. The search space slightly differs from the grid search strategy by offering more resolution for the layer widths, learning rate, and over several seed values to compensate for the possibility of bad initialization. We sample 600 random points from this space to obtain an approximate distribution of the solution space for MLP architectures for each dataset. From this we can obtain the best model (MLP - RS_{best}) and the median model (MLP - RS_{median}).

Table 7.1: Definitions of the architecture grid search subspace. The values in square brackets represent a continuous range, while the curly brackets denote a set of discrete values.

Parameter	Type	Grid search subspace
Seed	int	36
Number of layers	int	{2, 3, 4}
Layer width	int	{10, 15, 20, 25, 100}
Learning rate	float	5e-3
Optimizer	operator	{SGD, RMSProp, Adam}
Activation function	function	{ReLU, ELU, SELU, tanh}
Train epochs	int	{20, 25, 50, 75}

Table 7.2: Definitions of the architecture random search subpace. The values in square brackets represent a continuous range, while the curly brackets denote a set of discrete values.

Parameter	Type	Random search subspace
Seed	int	[0,100]
Number of layers	int	[2, 8]
Layer width	int	[100, 1000]
Learning rate	float	[1e-4, 1e-2]
Optimizer	operator	{SGD, RMSProp, Adam}
Activation function	function	{ReLU, LReLU, ELU, SELU, tanh, Sin}
Train epochs	int	50

7.4.3 Evolving Activation Functions

In this section, we describe in detail the setup for evolving activation functions.

Search space and Solution Encoding

The space of all feasible solutions is called a search space. Each activation function in the search space represents one possible solution. Every activation function is represented as a tree consisting of unary and binary operators with leaves corresponding to function inputs $\vec{x}$, namely the outputs of a dense linear layer. We consider the following operators:

- •Unary: $\vec{x}$, $-\vec{x}$, $|\vec{x}|$, $sin(\vec{x})$, $cos(\vec{x})$, $e^{\vec{x}}$, $erf(\vec{x})$, $\vec{x}^2$, $1//\vec{x}$, $\sigma(\vec{x})$, $\sigma_H(\vec{x})$, $ReLU(0,\vec{x})$, $ELU(\vec{x})$, $Softsign(\vec{x})$, $Softplus(\vec{x})$, $tanh(\vec{x})$
- •Unary, multidimensional: $normalized(\vec{x})$, $Softmax(\vec{x})$, $Softmin(\vec{x})$
- •Binary: $\vec{x_1}+\vec{x_2}$, $\vec{x_1}-\vec{x_2}$, $\vec{x_1}\cdot\vec{x_2}$, $\vec{x_1}//\vec{x_2}$

The operator $//$ denotes protected division, where the denominator value is replaced with $\varepsilon = 10^{-4}$ if the denominator's absolute value is smaller than ε. *ReLU* denotes the rectified linear unit, *ELU* the exponential linear unit, erf the Gaussian error function, σ the sigmoid function, σ_H the hard sigmoid function, and *normalized* the L_2 vector normalization. The initial tree depth is between 2 and 5, and the maximal tree depth is limited to 12. Moreover, there is no constraint of tree balancedness as discussed in [190] and [191].

Evolutionary Process

The selection process used in this work is presented as Algorithm 3 and employs a variant called steady-state tournament selection with the tournament size (k) equal to 3.

Evolution offers a more efficient approach to space sampling since it uses information from previous samplings to guide the search. We use this technique to find an activation function that optimizes the generalization of our models. The search space of functions is huge and requires some assumptions to make the search feasible. First, we assume the function can be represented as a tree, making the genetic programming technique a natural choice. Here, we constrain leaves to only be the function input, without constants or learnable parameters. Next, we limit the maximal depth of candidate trees to restrict the search to a fast-to-evaluate subspace of functions since they need to be evaluated for each layer. Finally, we limit the representation's expressivity by defining a set of possible unary and binary operations that can be used as function nodes (see Section 7.4.3). We use a population of 20 individuals and run the algorithm with a budget of 2000 evaluations.

Fitness Function

A neural network is trained with each function on a training dataset, starting with a population of P activation functions. Recall guessing entropy denotes the average key rank, i.e., the correct key position in the guessing vector after processing Q attack traces. As such, we aim to minimize the guessing entropy for any number of attack traces. Thus, it is natural to consider the number of attack traces required to reach the GE of 0, which we denote as $\overline{Q}_{t_{GE}}$. Each candidate function is assigned a fitness value F:

$$F = \overline{Q}_{t_{GE}} + (1 - accuracy). \tag{7.2}$$

The goal is to minimize fitness value F, where the optimal value is 1 (this would require only a single trace to break the target, representing the optimal scenario). The guessing entropy is averaged over 100 attacks on randomly selected data subsets. The maximum subset sizes were selected depending on a particular experiment to balance between differentiation of result qualities and computation time. This induces similar results between individuals in initial iterations and slows the EA convergence. To remedy this, we add the accuracy error $(1 - accuracy)$ to the fitness, which is also subject to minimization and adds additional information for differentiation between individuals. In preliminary experiments, we observed faster convergence by using this fitness function.

We note that it could be somewhat counterintuitive to use accuracy for SCA. This problem can be especially pronounced for the HW leakage model as it results in highly imbalanced data [177]. Still, as the second part of the fitness function is bounded in the range $[0,1]$, for neural networks that perform well (i.e., those that reach GE of 0 in a small number of traces), added information about accuracy can help by providing more search space gradient. When the number of required traces is high, the accuracy term's contribution is small, so the number of

traces remains the primary objective.

Mutation and Crossover

In mutation, one node in an activation function tree is randomly selected. The selected subtree is replaced at that point by generating a new subtree, respecting the maximal depth limit. The probability of mutation was selected from the preliminary experiments on the ASCAD fixed key dataset to provide reasonable exploration and exploitation properties and was kept at 70% through all of the experiments. In the crossover, two-parent activation functions exchange randomly selected subtrees, producing new children activation functions. The crossover is performed with a simple tree crossover with 90% bias for functional nodes being selected as crossover points.

7.4.4 Learning System

All experiments were performed on a machine using a single GeForce GTX 1080 Ti graphics card, i7-6700 CPU, and 32 GB of RAM, running Ubuntu 16.04. We implemented our experiments in Python 3.7 with the usage of the DEAP[201] framework (v1.3.1) for evolutionary algorithms and PyTorch framework [202] (v1.7.0) for deep learning with the CUDA (v11.2) backend.

As our model architectures do not require a significant GPU memory, we fully utilize the hardware by parallelizing individuals' evaluation step via multiprocessing. This proved crucial as our experiments are incredibly time-consuming, in the order of 7 days per architecture search and 14 days per evolution without parallelization. Note that the upper limit of the number of processes is dictated by the available GPU memory and physical CPU cores.

To ensure our experiments' reproducibility, we carefully set the seed value on both the CPU and GPU sides. Python's *random*, *numpy*, and *torch* libraries provide methods for managing the state of a random generator. By implementing a simple context manager, entire blocks of an experiment can be run under the standardized setup while maintaining the code clean and automatically restoring the context afterward.

7.5 Results

This section presents the experimental results, demonstrating that evolved activation functions can outperform commonly used activation functions. We first search for the optimal network architecture and hyperparameters using the previously discussed architecture search methods (random search - RS and grid search - GS) for each experimental setup. Then on the selected setup, we apply the evolutionary algorithm to further optimize the model by changing its acti-

Table 7.3: Final $\overline{Q}_{t_{GE}}$ values for ASCAD fixed and random keys datasets on the Hamming weight leakage model. Here we compare the best obtained value of grid search on the CNN model (CNN - GS_{best}, with its evolved activation function CNN - GP and the best obtained MLP model on random search MLP - RS_{best}.

	Fixed Key			**Random Keys**		
	best result	2nd best	3rd best	best result	2nd best	3rd best
CNN - GS_{best}	299	-	-	606	-	-
CNN - GP	287	331	339	$\overline{Q}_{t_{GE}} > 5000$	$\overline{Q}_{t_{GE}} > 5000$	$\overline{Q}_{t_{GE}} > 5000$
MLP - RS_{best}	561	669	749	1133	1592	1615

vation function. Finally, we compare the results of mentioned techniques on both datasets and leakage models. During the evaluation of GS and RS on ASCAD fixed key dataset with the ID leakage model, we truncated the evaluation of $\overline{Q}_{t_{GE}}$ to 1000 as we observed over 25% of results lied in this subspace, thus leading to an efficient region of interest. Additionally, we focused on 500 traces during the evaluation of GP to further improve efficiency while still obtaining better results. For the random keys dataset, we needed to increase the truncation bar to 1000 as it became more difficult to obtain 25% results in this subspace. Finally, for all of the HW leakage model tasks, we further increased the bar to 5000 as they proved to be quite a bit harder.

In Tables 7.3 and 7.4, we depict the best-obtained results for the Hamming weight and the ID leakage models, respectively. We additionally compare with the state-of-the-art results [185] when possible. The notation $\overline{Q}_{t_{GE}} > x$ denotes that we could not reach GE of 0 in x attack traces. We also depict the three best results obtained with various search techniques.

First, for the HW leakage model (Table 7.3) and fixed key, the best results are obtained with GP, while grid search with CNN performs slightly worse. Random search for MLP results are much worse than the first two, but we still manage to break the target. For random keys, we can see that the best results are reached for CNN with grid search, followed by MLP obtained through random search. Interestingly, CNN evolved with GP cannot converge even with 5000 attack traces. The architectures end with guessing entropy values: 108, 110, and 111 for the top 3 individuals, respectively. We postulate this happens as the random keys dataset is more difficult and becomes easier to overfit. Still, we believe adding more generations to the GP procedure would improve its behavior and attack results.

In Table 7.4, we depict results for the ID leakage model. Interestingly, for the fixed key, we see that both CNN evolved with GP and MLP with random search work better than related work [157]. We reach good results with MLP with random search only for random keys. Again, this shows that random keys setup is more difficult, and we require a more sophisticated search process to reach good results. The CNN - GS_{best} ends with a guessing entropy of 128, while the

Table 7.4: Final $\overline{Q}_{t_{GE}}$ values for ASCAD fixed and random keys datasets on the ID leakage model. Here we compare the best obtained value of grid search on the CNN model (CNN - GS_{best}), version with its evolved activation function (CNN - GP) and the best obtained MLP model on random search MLP - RS_{best}. The star denotes result obtained from reconstructing the resulting architecture in [157].

	Fixed Key			**Random Keys**		
	best result	2nd best	3rd best	best result	2nd best	3rd best
CNN - GS_{best}	191*	-	-	$\overline{Q}_{t_{GE}} > 1000$	-	-
CNN - GP	115	123	130	$\overline{Q}_{t_{GE}} > 1000$	$\overline{Q}_{t_{GE}} > 1000$	$\overline{Q}_{t_{GE}} > 1000$
MLP - RS_{best}	156	162	191	145	163	194

Table 7.5: Results of the EA effectiveness experiment for ASCAD fixed and random keys on the Hamming weight leakage model. The median architecture is compared before MLP - RS_{median} and after evolution MLP - GP.

	Fixed Key		**Random Keys**	
	MLP - RS_{median}	MLP - GP	MLP - RS_{median}	MLP - GP
$\overline{Q}_{t_{GE}}$	2377	1168	3350	$\overline{Q}_{t_{GE}} > 5000$

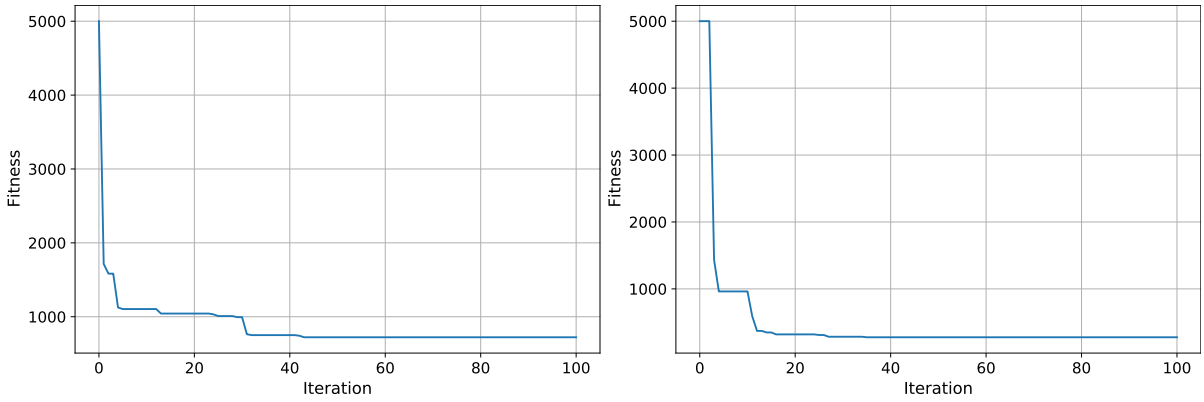
top 3 evolved results end respectively with 125, 128, and 128.

Next, in Table 7.5, we give median results for the HW leakage model when comparing MLP architectures obtained with random search and after evolving activations functions with GP. Note that GP improves the performance significantly for the fixed key, while GP does not converge for the random keys dataset. At the same time, the configuration found with random search manages to break the target in somewhat more than 5000 attack traces, ending with a guessing entropy value of 124. As the results with GP denote that the median does not manage to break the target, this again reiterates that we require more than 100 generations to evolve good activation functions.

Finally, in Table 7.6, we compare the median results for MLP with random search and after using genetic programming when considering the ID leakage model. Observe how for both fixed and random keys settings, GP reaches significantly better results. This indicates that while the random search can find a good performing neural network architecture just by guessing, the average results obtained over a number of solutions are not very good. On the other hand, evolving customized activation functions manages to improve the neural network performance significantly.

Table 7.6: Results of the EA effectiveness experiment for ASCAD fixed and random keys datasets on the ID leakage model. The median architecture is compared before MLP - RS_{median} and after evolution MLP - GP.

	Fixed Key		**Random Keys**	
	MLP - RS_{median}	MLP - GP	MLP - RS_{median}	MLP - GP
$\overline{Q}_{t_{GE}}$	531	279	437	188



(a) Fitness value of MLP - GP.

(b) Fitness value of CNN - GP.

Figure 7.1: The evolution of fitness value on the ASCAD random keys dataset and the HW leakage model.

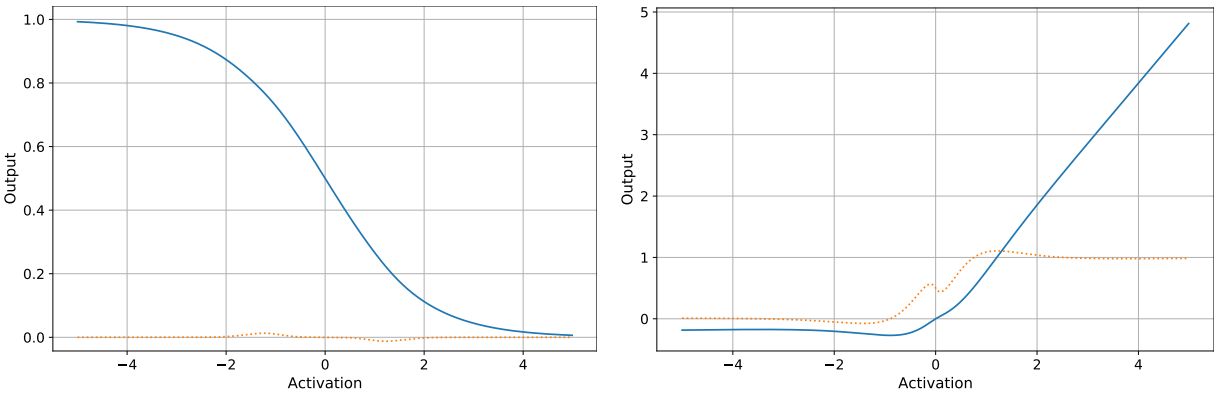
7.5.1 ASCAD Fixed Key

For the HW leakage model, we obtained a CNN architecture with the tail of four hidden dense layers of width 100, activated by *ELU*, and initialized with seed 36. The training setup uses the *RMSProp* optimizer with a learning rate of 0.005 over 20 epochs with batch size 64. For the architecture search of MLP, we obtained a model with eight hidden dense layers of width 478, activated by *ReLU*, and initialized with seed 42. The training setup uses the Adam optimizer with a learning rate of 0.0017 over 50 epochs with batch size 200.

For the HW leakage model and the ASCAD fixed key dataset, we depict in Figure 7.1 the convergence plots for the MLP and CNN architectures. We see that CNN converges faster, where the final fitness is two times smaller than in the MLP case. Next, in Figure 7.2, we depict the best-obtained activation function and its derivation. Both obtained functions have similarities with the commonly used ones. Interestingly, the GP evolved activation functions with similar properties to $tanh(-x)$ (see Figure 7.2a) function and ELU (see Figure 7.2b) activation function.

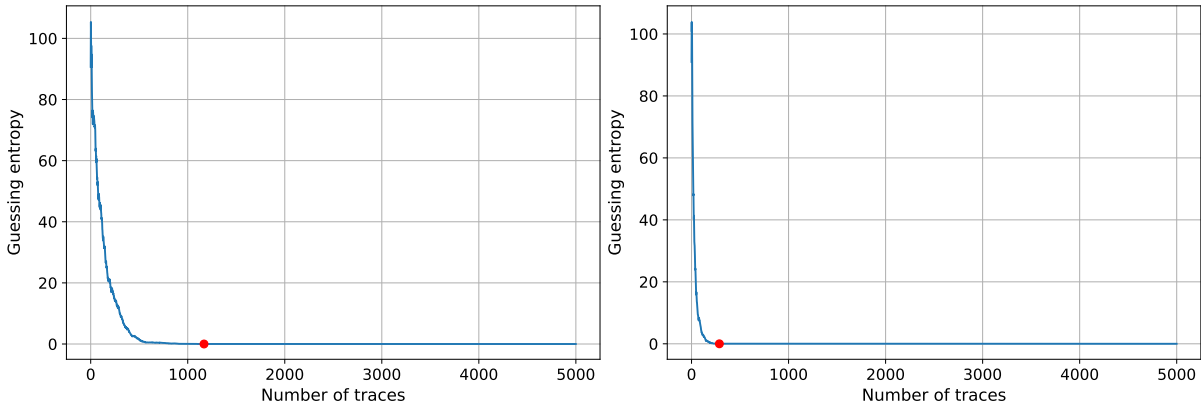
Finally, in Figure 7.3, we depict the number of attack traces required to reach a guessing entropy of 0, and we denoted that value with a red dot. Despite the fact that MLP and CNN architectures perform well, we break the target significantly faster when using CNN than MLP.

For the ID leakage model, we reimplemented the CNN reported in [157] and obtained sim-



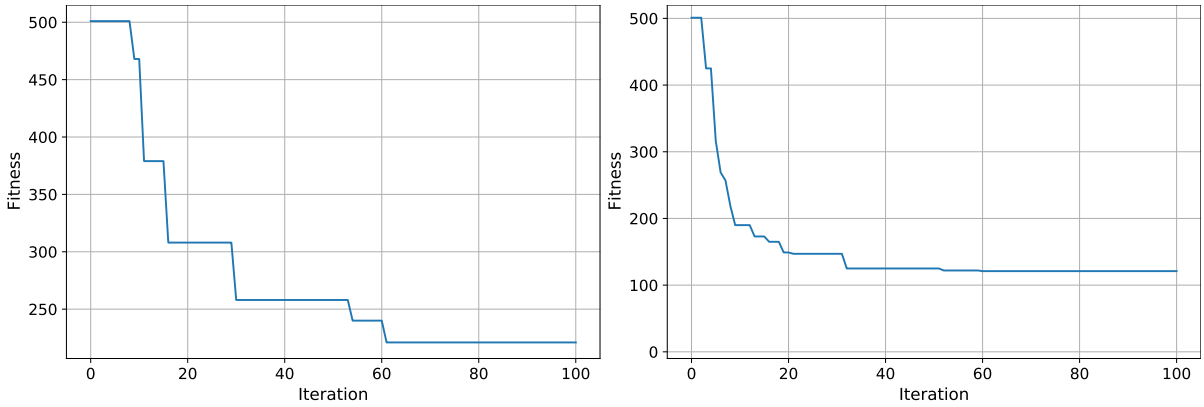
(a) Best activation function for MLP - GP. (b) Best activation function for CNN - GP.

Figure 7.2: Plots represent the 1D slice of the best activation function (solid) and its derivative (dotted) obtained through evolution on the ASCAD fixed key dataset. The slice is defined by setting the first dimension to the x-axis and the second dimension is set to 0.



(a) The guessing entropy of MLP - GP (Table 7.6). (b) The guessing entropy of CNN - GP (Table 7.4).

Figure 7.3: The guessing entropy of best evolved activation functions on the ASCAD fixed key dataset.



(a) Fitness value of MLP - GP.

(b) Fitness value of CNN - GP.

Figure 7.4: The evolution of fitness value for the ASCAD fixed key dataset and the ID leakage model.

ilar results. The training setup uses the Adam optimizer with a learning rate of 0.005 over 50 epochs with batch size 50. The architecture is initialized with a seed equal to 36 and uses the *SELU* activation function. For the architecture search of MLP, we obtained a model with five hidden dense layers of width 661, activated by the *Sine* function and initialized with seed 2. The training setup uses the Adam optimizer with a learning rate of 0.0086 over 50 epochs with batch size 200.

The best activation functions we obtained are denoted below, with the first letter in the subscript corresponding to the setup type and the second one to the architecture type. Note that the functions look rather complex, and it would be hard to expect that a human designer would find them. Still, the experimental results show they work very well.

$$a_{ID,C}(\vec{x}) = sin(erf(\vec{x} - softmin(\vec{x})^2)) - \zeta(\vec{x}) \tag{7.3}$$

$$\zeta(\vec{x}) = \vec{x}^2 \cdot softsign(\sigma(softmin(softsign(\sigma_H(\sigma(normalized(softmax(\vec{x}^2))))))))) \tag{7.4}$$

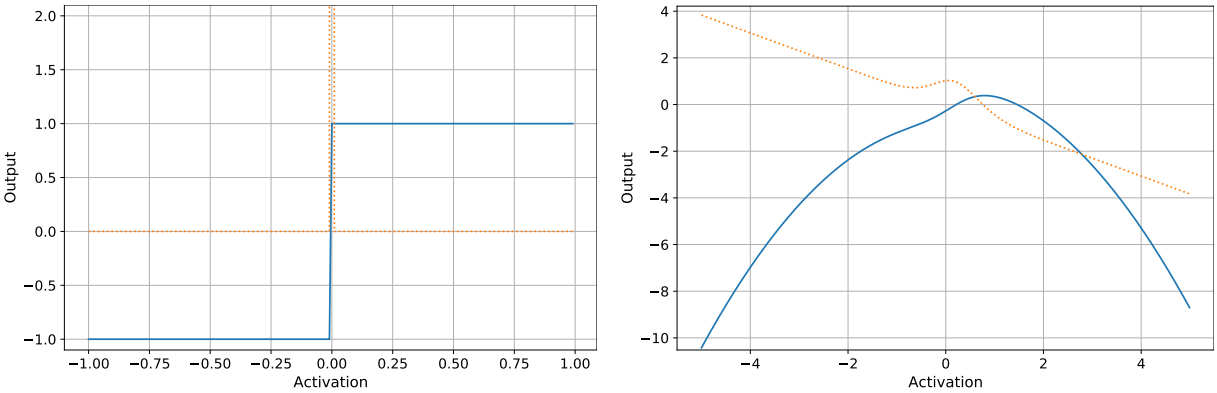
$$a_{ID,M}(\vec{x}) = normalized(softsign(normalized(\vec{x} \cdot tanh(softplus(softsign(erf(\vec{x})))))))) \tag{7.5}$$

$$a_{HW,C}(\vec{x}) = -(tanh(-\vec{x}) \cdot |softsign(2\vec{x}) - (\sigma(tanh(\vec{x})) + ELU(\vec{x}))|) \tag{7.6}$$

$$a_{HW,M}(\vec{x}) = softmax(softmin(sin(normalized(tanh(\frac{-1}{\sigma(\vec{x}^2)^2}))))) - \vec{x}) \tag{7.7}$$

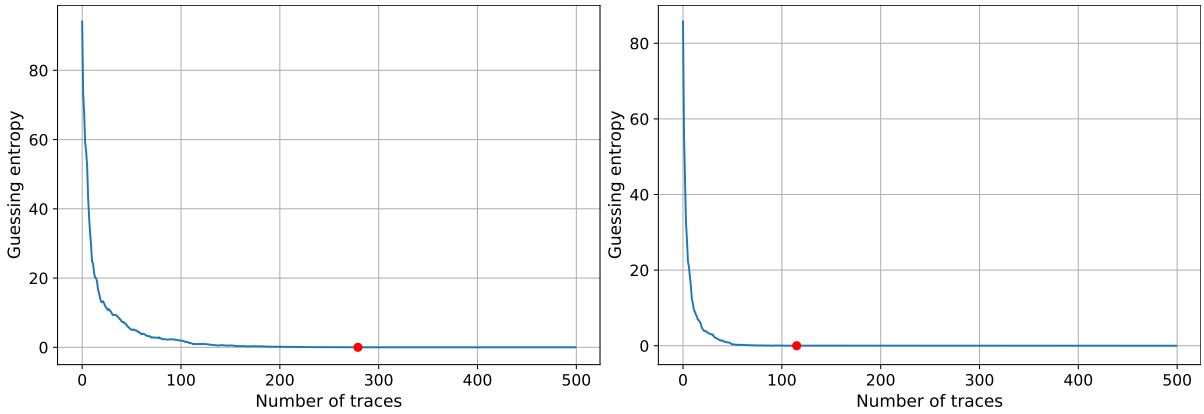
In Figure 7.4, we depict the GP evolution convergence plots for the MLP and CNN architecture for the ASCAD fixed key dataset. Notice how both architectures improve with iterations (clearly showing there is learning happening). Especially strong convergence can be seen for CNN, where the final fitness is more than twice smaller than in the MLP case.

Next, in Figure 7.5, we depict the best-obtained activation function and its derivation. Recall, the derivation is important as we require the differentiable function if we use the back-propagation algorithm, as is common in the training of neural networks. Notice that we obtain



(a) Best activation function for MLP - GP. (b) Best activation function for CNN - GP.

Figure 7.5: Plots represent the 1D slice of the best activation function (solid) and its derivative (dotted) obtained through evolution on the ASCAD fixed key dataset for ID prediction. The slice is defined by setting the first dimension to the x-axis and the second dimension is set to 0.



(a) The guessing entropy of MLP - GP (Table 7.6). (b) The guessing entropy of CNN - GP (Table 7.4).

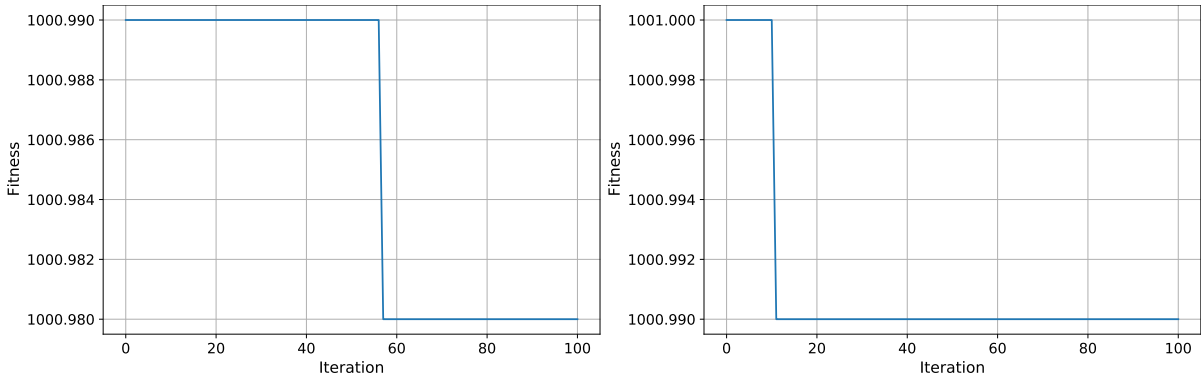
Figure 7.6: The guessing entropy of the best evolved activation functions on the ASCAD fixed key dataset and the ID leakage model.

a non-monotonic function for CNN, which is strikingly different from commonly (and state-of-the-art) used activation functions.

Finally, in Figure 7.6, we depict the number of attack traces required to reach a guessing entropy of 0. That value is denoted with a red dot. Notice that we break the target significantly faster when using CNN than MLP, but both techniques perform well. This ensures that we can find custom activation functions for SCA that perform well regardless of the neural network selection.

7.5.2 ASCAD Random Keys

In the case of the HW leakage model for the random keys dataset and CNN's architecture search, we obtained an architecture tail with four hidden dense layers of width 100, activated by the SELU activation function, and initialized with seed 36. The training setup uses the SGD



(a) Fitness value of MLP - GP.

(b) Fitness value of CNN - GP.

Figure 7.7: The evolution of fitness value on the ASCAD random keys dataset and the ID leakage model.

optimizer with a learning rate equal to 0.005 over 75 epochs with batch size 50. Using the architecture search of MLP, we obtained a model with two hidden dense layers of width 716, activated by SELU and initialized with seed 2. The training setup uses an SGD optimizer with a learning rate of 0.0042 over 50 epochs with batch size 200.

For the ID leakage model and the CNN's architecture search, we obtained an architecture tail with two hidden dense layers of width 100, activated by *tanh* activation function and initialized with seed 36. Using the architecture search of MLP, we obtained a model with five hidden dense layers of width 622, activated by *ReLU* and initialized with seed 2. The training setup uses the Adam optimizer with a learning rate of 0.00086 over 50 epochs with batch size 200.

The activation functions we obtained are shown below, with the first letter in the subscript corresponding to the setup type and the second one to the architecture type. Notice that these activation functions are slightly less complex (having fewer terms) than in the ASCAD fixed key scenario.

$$a_{ID,C}(\vec{x}) = (tanh(|sin(cos(\vec{x}))|))^{-1} \tag{7.8}$$

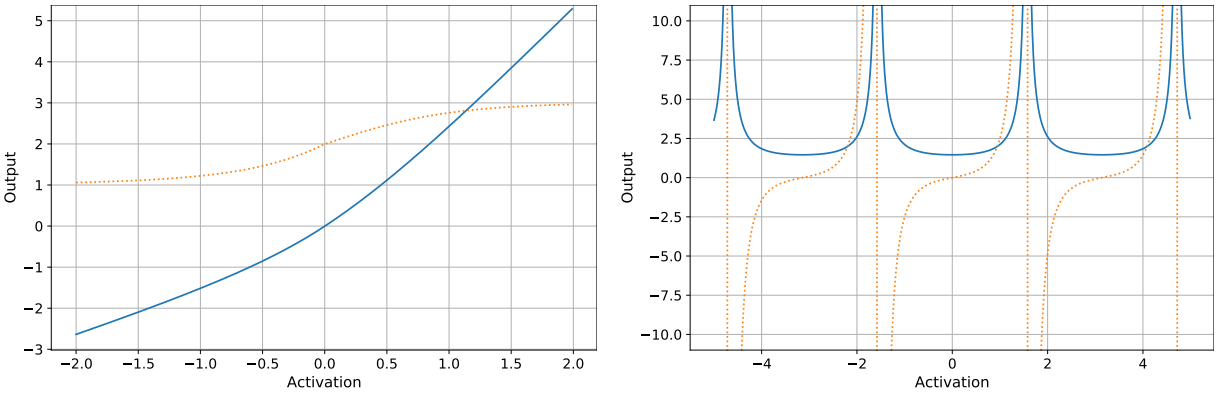
$$a_{ID,M}(\vec{x}) = softplus(\vec{x} + ELU(\vec{x})) + \vec{x} - \sigma(ELU(cos(\sigma_H(cos(softplus(\sigma(\vec{x}))))))) \tag{7.9}$$

$$a_{HW,C}(\vec{x}) = exp(cos(softmax(sin(\vec{x})^{-1}))) \tag{7.10}$$

$$a_{HW,M}(\vec{x}) = ReLU(\sigma_H(cos(softmin(normalized(|softmin(softmax(\vec{x}))|))^2))) \tag{7.11}$$

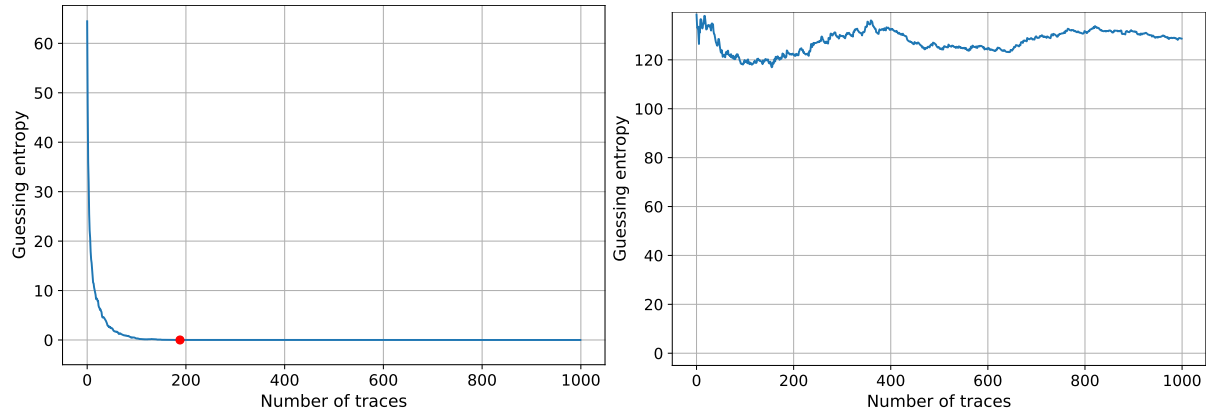
Next, in Figure 7.7, we depict the convergence plots. Again, CNN converges faster, and even with a relatively short number of iterations, there is no more improvement in the fitness value. Considering rather small improvements in the fitness value, we could conclude that the evolution process gets stuck in local optima. One potential solution could be to consider larger mutation rates to stimulate search space exploration.

Figure 7.8 presents plots for the activation function and its derivation. Notice that while



(a) Best activation function for MLP - GP. (b) Best activation function for CNN - GP.

Figure 7.8: Plots represent the 1D slice of the best activation function (solid) and its derivative (dotted) obtained through evolution on the ASCAD random keys dataset. The slice is defined by setting the first dimension to the x-axis and the second dimension is set to 0.



(a) The guessing entropy of MLP - GP (Table 7.6). (b) The guessing entropy of CNN - GP (Table 7.4).

Figure 7.9: The guessing entropy of best evolved activation functions on the ASCAD random keys dataset.

for MLP, the obtained functions have some similarity with commonly used ones, for CNN, the shape is rather unusual (and not intuitive that it would work). Finally, in Figure 7.9, we depict the guessing entropy results. Interestingly, MLP works very well (on the level as for the ASCAD with fixed key dataset), while CNN does not converge. Again, this reiterates that it is easier to tweak MLP architectures, regardless of whether it is done via hyperparameter tuning [180] or evolution of activation functions as investigated here. What is more, the results indicate that MLP architectures are sufficient to break the targets, especially if there is no trace misalignment (recall that in this work, we consider only synchronized traces). For CNNs, working with larger mutation rates and longer evolution could resolve the problems indicated here.

7.6 Conclusions

This chapter explores the potential of neuroevolution in enhancing deep learning-based side-channel analysis. Specifically, we investigate the scenario where genetic programming is utilized to evolve activation functions tailored explicitly for side-channel analysis. To demonstrate the effectiveness of this approach, we conduct experiments on two side-channel analysis datasets and two leakage models. Our results show that activation function evolution can produce improved attack behavior, with the highest efficiency observed in the more specific dataset. However, further research is necessary to evaluate the advantages and limitations of this approach comprehensively.

Our findings also reveal the need for more informative and cost-effective fitness functions to facilitate the faster convergence of better-performing individuals. Such improvements could enhance the practicality and applicability of neuroevolution in side-channel analysis.

Chapter 8

Conclusion

This dissertation was done using the bottom-up methodology, where the research proceeded in the following steps:

1.construction of Boolean functions with adaptive cryptographic properties,

2.construction of vectorial Boolean functions with adaptive cryptographic properties,

3.automatic construction of cryptographic algorithms by using attacker and defence dynamics in the security domain,

4.semi-supervised learning in side-channel attacks,

5.neuroevolutionary procedures for optimization of neural network architecture in the side-channel attacks.

This thesis primarily explores the intersection of artificial intelligence (AI) and cryptography, focusing on applying AI techniques to cryptographic problems. It is important to note that the contributions of this work are predominantly from the AI perspective, with some aspects needing to be more rigorously treated from a cryptographic standpoint. The use of evolutionary computing and machine learning in solving cryptographic problems has been extensively studied, with evolutionary computing algorithms being compelling in optimization problems, such as the design and analysis of symmetric key cryptographic algorithms. However, it should be noted that evolutionary computing algorithms are not universally applicable, and the No Free Lunch theorem highlights that there is no single best evolutionary algorithm[203].

In this dissertation, we investigate niches in constructing Boolean and vectorial Boolean functions where unanswered questions exist. Although these questions may not necessarily have practical importance, they provide new theoretical insights and demonstrate the capabilities of evolutionary computing.

We also explore the possibility of using evolutionary computing algorithms to construct cryptographic algorithms inspired by neural cryptography automatically. Although such experiments may raise doubts among cryptographers, exploring new territory is an intriguing avenue for research, and our results may serve as a guide for future investigations.

In the field of profiled side-channel attacks, previous research has suggested that the application of semi-supervised learning is weak. However, our work indicates that when the number of profiling traces is limited, and if more attack traces can be measured, semi-supervised learning can significantly improve machine learning models. In addition to the type of model learning, we also investigate the model itself in profiled side-channel attacks. While the evolution of activation functions in neural networks is not new, this dissertation presents the results of applying the neuroevolution of activation functions tailored explicitly for side-channel analysis. Our initial findings suggest that there is ample room for future research in this area.

In conclusion, this chapter summarizes the most critical findings from the previous chapters and highlights the original scientific contributions of this dissertation. Finally, we identify open questions and potential avenues for future research.

8.1 Achieved Contribution and Main Conclusions

The main goal of this dissertation is to demonstrate the application of machine learning and evolutionary computing in the design and analysis of cryptographic algorithms with a symmetric key. The conclusions of the scientific contribution of this dissertation are briefly stated below.

8.1.1 Construction of Boolean and Vectorial Boolean Functions

Chapter 4 addressed the construction of maximal nonlinearity Boolean functions through evolutionary algorithms. To resist linear cryptanalysis attacks, Boolean functions need to have high nonlinearity. Bent functions are Boolean functions with maximal possible nonlinearity for a given number of variables. It is also crucial for functions to be balanced for usage in cryptographic algorithms. Our results suggest that one can use evolutionary algorithms to evolve Boolean functions of many different sizes, whereas the best performing algorithm we consider genetic programming. Moreover, we introduce the problem of evolving quaternary bent Boolean functions. Again, tree encoding offers superior results in those experiments. The results for quaternary tree encoding show that we can obtain bent functions for all dimensions we experiment with. Our results are comparable to or better than those obtained with other techniques when evolving bent binary Boolean functions.

In Chapter 4, we investigate S-boxes where the output is smaller than the input. Such S-boxes have practical applications in authentication codes or secret sharing schemes but are also interesting as combinatorial optimization problems and could be used as benchmarks. Here again, we are interested in constructing bent S-boxes, where the genetic algorithm proves the most successful. Moreover, we investigate whether we can use heuristics to generate differentially 6-uniform $(n, n-2)$ functions. Our theoretical work on this topic points us to the

conclusion that this problem is challenging. Our results confirm that the problem is intricate and evolutionary algorithms are of limited success.

8.1.2 Automatic Construction of Cryptographic Algorithms

In Chapter 5, the present study explores the automatic construction of ciphers using Cartesian Genetic Programming (CGP) and bi-level optimization. The study aims to produce ciphers that demonstrate relative resilience against attacks while utilizing fewer active nodes, thereby enhancing their interpretability. Our findings indicate that including additional cipher properties leads to improved results, with the resulting ciphers demonstrating a level of security that prevents the attacker, Eve, from achieving significantly better performance than random guessing.

Our results represent a proof of concept, showcasing the potential for evolutionary algorithms (EAs) to function as automatic cipher builders. However, further refinements are necessary to generate results that would be practically valuable.

8.1.3 Machine Learning Algorithms in Side-channel Attacks

In Chapter 6, we investigate the scenario where the attacker is constrained in the profiling phase but has access to additional information from the attacking measurements to construct the profiled model. We examine two semi-supervised learning methods under various conditions, using machine learning techniques and template attacks, as well as their pooled variant, as side-channel attack methods.

Our findings indicate that semi-supervised learning can be beneficial in numerous scenarios. In particular, significant enhancements are observed for the template attack and its pooled version in the low-noise situation. We note that incorporating extra samples from the attacking phase enhances the estimation of the covariance matrices, resulting in an improvement of over 70% in terms of the attack traces required to guess the key.

8.1.4 Neuroevolution in Side-channel Attacks

Chapter 7 investigates how neuroevolution can improve deep learning-based side-channel analysis. More precisely, we consider the setting where genetic programming evolves activation functions specifically adapted for the side-channel analysis. We conduct experiments for two SCA datasets and two leakage models to show that it is possible to evolve activation functions that improve the attack behavior. We observe that activation function evolution has higher efficiency for a more straightforward dataset, indicating that more work is needed to understand this approach's advantages and drawbacks. Additionally, we observe the need for more informative and cost-effective fitness functions that would lead to better individuals faster. Research in this

direction would improve the effectiveness of neuroevolution regardless of the hyperparameter being evolved.

8.2 Future research

Although this dissertation has provided answers to some questions, there are still numerous unanswered queries.

In Chapter 3, we propose a potential avenue of future research in the domain of secondary constructions. The research direction involves exploring the efficacy of constructing balanced Boolean functions with maximal nonlinearity utilizing secondary constructions. Although secondary constructions in GP have already demonstrated the ability to construct bent Boolean functions for a much larger number of inputs, exploring their effectiveness in constructing balanced Boolean functions with maximal nonlinearity is an interesting area for further study.

Based on the results obtained with quaternary Boolean functions in our research, several potential research directions exist. The first option is to consider quaternary functions with $n > 8$ variables. However, it remains to be seen whether such a research direction would provide advantages over the evolution of binary Boolean functions with $2n$ variables. The second option is to consider the Hamming distance when calculating nonlinearity. Although the mapping between $\mathbb{Z}_4^n$ and $\mathbb{F}_2^{2n}$ is less elegant in this case, finding bent binary Boolean functions is still feasible.

Chapter 4 opens several research avenues. In the first direction, we propose to explore CGP, as it has demonstrated very good results in our experiments. With CGP, we could avoid the need for m independent trees and instead have a graph with m outputs. The second research direction involves exploring larger S-box sizes, especially the bitstring representation observed to perform the best for size $(12, 6)$. Finally, since we observe good performance when using the fitness function with a derivative of the Boolean function, we plan to investigate its behavior on larger Boolean functions.

In Chapter 5, we identified several potential research directions. The first is the use of rounds in the proposed algorithm. It would be interesting to observe attackers' success in such an environment and how much the introduction of rounds contributes to the strength of the cryptographic algorithm. Another research direction involves introducing more complex cryptographic primitives into the set of CGP functional nodes. The entire process of automatic construction of a cryptographic algorithm could be viewed as a multilevel optimization problem where cryptographic primitives and the whole algorithm are constructed in parallel. Lastly, introducing cryptanalytic tools in evaluating the constructed crypto algorithm would likely result in better solutions.

For future work, Chapter 6 opens up scenarios where the number of labeled traces is minimal

(100 labeled traces or less), whereas the number of unlabeled examples is much more significant, e.g., 30 000. We have demonstrated that the most significant benefit of SSL is in these extreme cases. A second research direction involves considering those measurements with the highest probabilities and using the distribution of probabilities from SSL learning. Additionally, in the semi-supervised phase, we used ML classifiers to obtain new labeled measurements, but there is no reason not to try using the TA_p attack. Consequently, we plan to investigate the scenario where the TA_p attack is used as the classifier in self-training. Finally, in a real-world scenario, two different devices should be considered, which may result in (slightly) different distributions (see e.g., [204, 205]).

Finally, since Chapter 7 is the first time considering neuroevolution for SCA, there are many possible research directions for future work. One approach would be to consider evolving other elements of the learning procedure, like the loss function. Another option could be to use neuroevolution to evolve the whole neural network architecture, not restricting activation function only. Moreover, another viable option is to use evolution instead of the backpropagation algorithm so that the activation function search does not need to be restricted to differentiable elements. Considering the application of different activation functions, one natural option would be to consider the evolution of different activation functions for various layers. It would also be interesting to investigate how transferable the obtained activation functions are when considering already designed neural network architectures. Our preliminary testing indicates this transferability to be somewhat limited, but more experiments are required.

Bibliography

[1]Klimov, A., Mityagin, A., Shamir, A., “Analysis of neural cryptography”, in Advances in Cryptology — ASIACRYPT 2002, Zheng, Y., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, str. 288–298.

[2]Eiben, A. E., Smith, J. E., Introduction to Evolutionary Computing. Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

[3]Goldberg, D. E., Holland, J. H., “Genetic algorithms and machine learning”, Machine learning, Vol. 3, No. 2, 1988, str. 95–99.

[4]Koza, J. R., Genetic programming: on the programming of computers by means of natural selection. MIT press, 1992, Vol. 1.

[5]Beyer, H.-G., Schwefel, H.-P., “Evolution strategies–a comprehensive introduction”, Natural computing, Vol. 1, No. 1, 2002, str. 3–52.

[6]Fogel, D. B., Evolutionary computation: toward a new philosophy of machine intelligence. John Wiley & Sons, 2006, Vol. 1.

[7]Darwin, C., On the Origin of Species by Means of Natural Selection. London: Murray, 1859, or the Preservation of Favored Races in the Struggle for Life.

[8]Holland, J. H., Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press, Cambridge, USA, 1992.

[9]Kneževi ć, K., “Generating prime numbers using genetic algorithms”, in 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), 2021, str. 1224-1229.

[10]Knezevic, K., Picek, S., Mariot, L., Jakobovic, D., Leporati, A., “The design of (almost) disjunct matrices by evolutionary algorithms”, in Theory and Practice of Natural Computing, Fagan, D., Martín-Vide, C., O’Neill, M., Vega-Rodríguez, M. A., (ur.). Cham: Springer International Publishing, 2018, str. 152–163.

[11]Koza, J. R., “Evolving a computer program to generate random numbers using the genetic programming paradigm”, 1991.

[12]Miller, J. F., “An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach”, in GECCO, Banzhaf, W., Daida, J. M., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M. J., Smith, R. E., (ur.). Morgan Kaufmann, 1999, str. 1135–1142.

[13]Miller, J. F., (ur.), Cartesian Genetic Programming, ser. Natural Computing Series. Springer Berlin Heidelberg, 2011.

[14]Miller, J. F., Thomson, P., “Cartesian Genetic Programming”, in EuroGP, 2000, str. 121–132.

[15]Knezevic, K., Picek, S., Miller, J. F., “Amplitude-oriented mixed-type cgp classification”, in Proceedings of the Genetic and Evolutionary Computation Conference Companion, ser. GECCO ’17. New York, NY, USA: Association for Computing Machinery, 2017, str. 1415–1418, dostupno na: https://doi.org/10.1145/3067695.3082500

[16]Goldman, B. W., Punch, W. F., “Reducing Wasted Evaluations in Cartesian Genetic Programming”, in Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings, Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. Ş., Hu, B., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, str. 61–72.

[17]Bishop, C. M., Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[18]Friedman, J. H., Bentley, J. L., Finkel, R. A., “An Algorithm for Finding Best Matches in Logarithmic Expected Time”, ACM Trans. Math. Softw., Vol. 3, No. 3, Sep. 1977, str. 209–226.

[19]Vapnik, V. N., The Nature of Statistical Learning Theory. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[20]Platt, J., “Fast Training of Support Vector Machines using Sequential Minimal Optimization”, in Advances in Kernel Methods - Support Vector Learning, Schoelkopf, B., Burges, C., Smola, A., (ur.). MIT Press, 1998.

[21]Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., Murthy, K. R. K., “Improvements to Platt’s SMO Algorithm for SVM Classifier Design”, Neural Comput., Vol. 13, No. 3, Mar. 2001, str. 637–649.

[22]Mitchell, T. M., Machine Learning. New York: McGraw-Hill, 1997.

[23]Collobert, R., Bengio, S., "Links between perceptrons, mlps and svms", in Proceedings of the Twenty-First International Conference on Machine Learning, ser. ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, str. 23.

[24]Fukushima, K., "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", Biological Cybernetics, Vol. 36, No. 4, Apr 1980, str. 193–202.

[25]Lecun, Y., Bengio, Y., Convolutional networks for images, speech, and time-series. MIT Press, 1995.

[26]Goodfellow, I., Bengio, Y., Courville, A., Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

[27]Duch, W., Jankowski, N., "Survey of neural transfer functions", Neural Computing Surveys, Vol. 2, 1999, str. 163–213.

[28]Paar, C., Pelzl, J., Understanding Cryptography - A Textbook for Students and Practitioners. Springer, 2010.

[29]Kneževi ć, K., "Combinatorial optimization in cryptography", in 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017, str. 1324–1330.

[30]Katz, J., Lindell, Y., Introduction to modern cryptography. CRC press, 2014.

[31]Rivest, R. L., Shamir, A., Adleman, L., "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, Vol. 21, No. 2, 1978, str. 120–126.

[32]Barker, E., et al., "Recommendation for pair-wise key-establishment using integer factorization cryptography", dostupno na: https://doi.org/10.6028/NIST.SP.800-56Br2 Mar. 2019.

[33]Diffie, W., Hellman, M., "New directions in cryptography", IEEE transactions on Information Theory, Vol. 22, No. 6, 1976, str. 644–654.

[34]ElGamal, T., "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE transactions on information theory, Vol. 31, No. 4, 1985, str. 469–472.

[35]Koblitz, N., Menezes, A., Vanstone, S., "The state of elliptic curve cryptography", in Towards a quarter-century of public key cryptography. Springer, 2000, str. 103–123.

[36]Merkle, R. C., Hellman, M. E., “On the security of multiple encryption”, Communications of the ACM, Vol. 24, July 1981, str. 465–467, dostupno na: http://doi.acm.org/10.1145/358699.358718

[37]McEliece, R. J., Theory of Information and Coding, 2nd ed. New York, NY, USA: Cambridge University Press, 2001.

[38]Whyte, W., Hoffstein, J., NTRU. Boston, MA: Springer US, 2011, str. 858–861, dostupno na: https://doi.org/10.1007/978-1-4419-5906-5_464

[39]Knudsen, L. R., Robshaw, M., The Block Cipher Companion, ser. Information Security and Cryptography. Springer, 2011, dostupno na: https://doi.org/10.1007/978-3-642-17342-4

[40]Daemen, J., Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis. K.U.Leuven, 1995, http://jda.noekeon.org/.

[41]Carlet, C., “Boolean functions for cryptography and error correcting codes”, Boolean models and methods in mathematics, computer science, and engineering, Vol. 2, 2010, str. 257–397.

[42]Massey, J., “Shift-register synthesis and BCH decoding”, Information Theory, IEEE Transactions on, Vol. 15, No. 1, Jan 1969, str. 122–127.

[43]Carlet, C., “Boolean functions for cryptography and error-correcting codes”, in Boolean Models and Methods in Mathematics, Computer Science, and Engineering, Crama, Y., , Hammer, P. L., (ur.). New York: Cambridge University Press, 2011, str. 257–397.

[44]Carlet, C., “Vectorial Boolean Functions for Cryptography”, in Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st ed., Crama, Y., Hammer, P. L., (ur.). New York, USA: Cambridge University Press, 2010, str. 398–469.

[45]Kerckhoffs, A., “La cryptographie militaire”, Journal des Sciences Militaires, 1883, str. 161–191.

[46]Shannon, C., “Communication theory of secrecy systems”, Bell System Technical Journal, Vol. 28, No. 4, 1949, str. 656–715.

[47]Webster, A. F., Tavares, S. E., “On the Design of S-Boxes”, in Advances in Cryptology — CRYPTO ’85 Proceedings, Williams, H. C., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, str. 523–534.

[48]“Workshop on “Provable Security against Physical Attacks””, Amsterdam, Netherlands. http://www.lorentzcenter.nl/lc/web/2010/383/program.php3?wsid=383. February 10-19 2010.

[49]Mangard, S., Popp, T., Gammel, B. M., “Side-Channel Leakage of Masked CMOS Gates”, in CT-RSA, ser. LNCS, Vol. 3376. Springer, 2005, str. 351–365, San Francisco, CA, USA.

[50]Mangard, S., Oswald, E., Popp, T., Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, December 2006, ISBN 0-387-30857-1, http://www.dpabook.org/.

[51]Gierlichs, B., De Mulder, E., Preneel, B., Verbauwhede, I., “Empirical comparison of side channel analysis distinguishers on DES in hardware”, in 2009 European Conference on Circuit Theory and Design, 2009, str. 391–394.

[52]Kocher, P. C., Jaffe, J., Jun, B., Rohatgi, P., “Introduction to differential power analysis”, J. Cryptographic Engineering, Vol. 1, No. 1, 2011, str. 5–27.

[53]Rothaus, O., “On “bent” functions”, Journal of Combinatorial Theory, Series A, Vol. 20, No. 3, 1976, str. 300–305.

[54]Bernasconi, A., Codenotti, B., Vanderkam, J. M., “A characterization of bent functions in terms of strongly regular graphs”, IEEE Transactions on Computers, Vol. 50, No. 9, Sep 2001, str. 984–985.

[55]Kavut, S., Maitra, S., Yucel, M. D., “Search for boolean functions with excellent profiles in the rotation symmetric class”, IEEE Transactions on Information Theory, Vol. 53, No. 5, May 2007, str. 1743–1751.

[56]Kerdock, A., “A class of low-rate nonlinear binary codes”, Information and Control, Vol. 20, No. 2, 1972, str. 182–187, dostupno na: http://www.sciencedirect.com/science/article/pii/S0019995872903762

[57]Zheng, Y., Pieprzyk, J., Seberry, J., “HAVAL — a one-way hashing algorithm with variable length of output (extended abstract)”, in Advances in Cryptology — AUSCRYPT ’92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings, Seberry, J., Zheng, Y., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, str. 81–104.

[58]Hell, M., Johansson, T., Maximov, A., Meier, W., “A stream cipher proposal: Grain-128”, in 2006 IEEE International Symposium on Information Theory, July 2006, str. 1614–1618.

[59]Adams, C. M., "Constructing symmetric ciphers using the cast design procedure", Designs, Codes and Cryptography, Vol. 12, No. 3, Nov 1997, str. 283–316, dostupno na: https://doi.org/10.1023/A:1008229029587

[60]Méaux, P., Journault, A., Standaert, F.-X., Carlet, C., "Towards stream ciphers for efficient fhe with low-noise ciphertexts", in Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I, Fischlin, M., Coron, J.-S., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, str. 311–343.

[61]Picek, S., Marchiori, E., Batina, L., Jakobovic, D., "Combining Evolutionary Computation and Algebraic Constructions to Find Cryptography-Relevant Boolean Functions", in Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings, 2014, str. 822–831.

[62]Schmidt, K. U., "Quaternary constant-amplitude codes for multicode cdma", IEEE Transactions on Information Theory, Vol. 55, No. 4, April 2009, str. 1824–1832.

[63]Jadda, Z., Parraud, P., Qarboua, S., "Quaternary cryptographic bent functions and their binary projection", Cryptography and Communications, Vol. 5, No. 1, 2013, str. 49–65.

[64]Forrié, R., "The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition", in Advances in Cryptology - CRYPTO' 88, ser. Lecture Notes in Computer Science, Goldwasser, S., (ur.). Springer New York, 1990, Vol. 403, str. 450–468, dostupno na: http://dx.doi.org/10.1007/0-387-34799-2_31

[65]Preneel, B., Van Leekwijck, W., Van Linden, L., Govaerts, R., Vandewalle, J., "Propagation characteristics of Boolean functions", in Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology, ser. EUROCRYPT '90. New York, NY, USA: Springer-Verlag New York, Inc., 1991, str. 161–173, dostupno na: http://dl.acm.org/citation.cfm?id=112331.112345

[66]Dillon, J., "A Survey of Bent Functions*", Reprinted from the NSA Technical Journal. Special Issue, Tech. Rep., 1972, unclassified.

[67]Jadda, Z., Parraud, P., "Z4-nonlinearity of a constructed quaternary cryptographic functions class", in Sequences and Their Applications – SETA 2010: 6th International Conference, Paris, France, September 13-17, 2010. Proceedings, Carlet, C., Pott, A., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, str. 270–283.

[68]Solé, P., Tokareva, N., "Connections between quaternary and binary bent functions", Cryptology ePrint Archive, Report 2009/544, http://eprint.iacr.org/2009/544. 2009.

[69]Jadda, Z., Parraud, P., Qarboua, S., “Quaternary cryptographic bent functions and their binary projection”, Cryptography and Communications, Vol. 5, No. 1, 2013, str. 49–65, dostupno na: https://doi.org/10.1007/s12095-012-0077-3

[70]Millan, W., Clark, A., Dawson, E., “An effective genetic algorithm for finding highly nonlinear boolean functions”, in Information and Communications Security, Han, Y., Okamoto, T., Qing, S., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, str. 149–158.

[71]Millan, W., Clark, A., Dawson, E., “Heuristic Design of Cryptographically Strong Balanced Boolean Functions”, in EUROCRYPT ’98, 1998, str. 489–499.

[72]McLaughlin, J., Clark, J. A., “Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity”, Cryptology ePrint Archive, Report 2013/011, http://eprint.iacr.org/. 2013.

[73]Picek, S., Jakobovic, D., Golub, M., “Evolving Cryptographically Sound Boolean Functions”, in Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, ser. GECCO ’13 Companion. New York, NY, USA: ACM, 2013, str. 191–192.

[74]Mariot, L., Leporati, A., “Heuristic search by particle swarm optimization of boolean functions for cryptographic applications”, in Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings, 2015, str. 1425–1426.

[75]Hrbacek, R., Dvorak, V., “Bent Function Synthesis by Means of Cartesian Genetic Programming”, in Parallel Problem Solving from Nature - PPSN XIII, ser. Lecture Notes in Computer Science, Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J., (ur.). Springer International Publishing, 2014, Vol. 8672, str. 414–423, dostupno na: http://dx.doi.org/10.1007/978-3-319-10762-2_41

[76]Picek, S., Jakobovic, D., Miller, J. F., Marchiori, E., Batina, L., “Evolutionary Methods for the Construction of Cryptographic Boolean Functions”, in Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, 2015, str. 192–204.

[77]Picek, S., Carlet, C., Jakobovic, D., Miller, J. F., Batina, L., “Correlation Immunity of Boolean Functions: An Evolutionary Algorithms Perspective”, in Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, 2015, str. 1095–1102.

[78]Picek, S., Jakobovic, D., Miller, J. F., Batina, L., Cupic, M., "Cryptographic boolean functions: One output, many design criteria", Applied Soft Computing, Vol. 40, 2016, str. 635–653.

[79]Picek, S., Carlet, C., Guilley, S., Miller, J. F., Jakobovic, D., "Evolutionary algorithms for boolean functions in diverse domains of cryptography", Evolutionary computation, 2016.

[80]Picek, S., Knezevic, K., Mariot, L., Jakobovic, D., Leporati, A., "Evolving bent quaternary functions", in 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, str. 1–8.

[81]Picek, S., Sisejkovic, D., Jakobovic, D., "Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties", Eng. Appl. of AI, Vol. 62, 2017, str. 320–330, dostupno na: https://doi.org/10.1016/j.engappai.2016.11.002

[82]Picek, S., Jakobovic, D., "Evolving Algebraic Constructions for Designing Bent Boolean Functions", in Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016, 2016, str. 781–788.

[83]Picek, S., Knezevic, K., Jakobovic, D., "On the evolution of bent (n,m) functions", in 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, str. 2137–2144.

[84]Clark, J. A., Jacob, J., Maitra, S., St ănică, P., "Almost Boolean functions: the design of Boolean functions by spectral inversion", in Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, Vol. 3, Dec 2003, str. 2173–2180 Vol.3.

[85]Mariot, L., Leporati, A., "A Genetic Algorithm for Evolving Plateaued Cryptographic Boolean Functions", in Theory and Practice of Natural Computing - Fourth International Conference, TPNC 2015, Mieres, Spain, December 15-16, 2015. Proceedings, 2015, str. 33–45.

[86]Picek, S., Santana, R., Jakobovic, D., "Maximal nonlinearity in balanced boolean functions with even number of inputs, revisited", in 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, str. 3222–3229.

[87]Tang, X., Tang, D., Zeng, X., Hu, L., "Balanced boolean functions with (almost) optimal algebraic immunity and very high nonlinearity", dostupno na: http://eprint.iacr.org/2010/443 2010.

[88]Clark, J. A., Jacob, J. L., Stepney, S., Maitra, S., Millan, W., "Evolving boolean functions satisfying multiple criteria", in Progress in Cryptology - INDOCRYPT 2002, Third

International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002, ser. Lecture Notes in Computer Science, Menezes, A., Sarkar, P., (ur.), Vol. 2551. Springer, 2002, str. 246–259, dostupno na: https://doi.org/10.1007/3-540-36231-2_20

[89]Eiben, A. E., Smith, J. E., Introduction to Evolutionary Computing, 2nd ed. Springer-Verlag, Berlin Heidelberg New York, USA, 2015.

[90]Koza, J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press, 1992.

[91]Poli, R., Langdon, W. B., McPhee, N. F., A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008, (With contributions by J. R. Koza).

[92]Picek, S., Jakobovic, D., Miller, J. F., Batina, L., Cupic, M., "Cryptographic Boolean functions: One output, many design criteria", Appl. Soft Comput., Vol. 40, 2016, str. 635–653.

[93]Biham, E., Shamir, A., "Differential Cryptanalysis of DES-like Cryptosystems", in Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology, ser. CRYPTO '90. London, UK: Springer-Verlag, 1991, str. 2–21, dostupno na: http://dl.acm.org/citation.cfm?id=646755.705229

[94]Matsui, M., Yamagishi, A., "A new method for known plaintext attack of FEAL cipher", in Proceedings of the 11th annual international conference on Theory and application of cryptographic techniques, ser. EUROCRYPT'92. Berlin, Heidelberg: Springer-Verlag, 1993, str. 81–91, dostupno na: http://dl.acm.org/citation.cfm?id=1754948.1754958

[95]Meier, W., Staffelbach, O., "Fast Correlation Attacks on Stream Ciphers", in Advances in Cryptology - EUROCRYPT '88, ser. Lecture Notes in Computer Science, Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C., (ur.). Springer Berlin Heidelberg, 1988, Vol. 330, str. 301–314.

[96]Courtois, N., Meier, W., "Algebraic Attacks on Stream Ciphers with Linear Feedback", in Advances in Cryptology - EUROCRYPT 2003, ser. Lecture Notes in Computer Science, Biham, E., (ur.). Springer Berlin Heidelberg, 2003, Vol. 2656, str. 345–359.

[97]Daemen, J., Rijmen, V., "Probability distributions of correlation and differentials in block ciphers", J. Mathematical Cryptology, Vol. 1, No. 3, 2007, str. 221–242.

[98]"FIPS 46-3, Data Encryption Standard (DES)", National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA, 1999.

[99]Knudsen, L. R., Robshaw, M., The Block Cipher Companion., ser. Information Security and Cryptography. Springer, 2011.

[100]Daemen, J., Rijmen, V., The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, 2002.

[101]Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., Seurin, Y., Vikkelsoe, C., "PRESENT: An Ultra-Lightweight Block Cipher", in Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems, ser. CHES '07. Berlin, Heidelberg: Springer-Verlag, 2007, str. 450–466.

[102]Mariot, L., Picek, S., Leporati, A., Jakobovic, D., "Cellular automata based s-boxes", Cryptology ePrint Archive, Report 2017/1055, https://eprint.iacr.org/2017/1055. 2017.

[103]Carlet, C., Alsalami, Y., "A new construction of differentially 4-uniform $(n,n-1)$-functions", Advances in Mathematics of Communications, Vol. 9, 2015, str. 541, dostupno na: http://aimsciences.org//article/id/c152a42b-4fdc-49f9-b487-20d028044b61

[104]Carlet, C., "Open Questions on Nonlinearity and on APN Functions", in Arithmetic of Finite Fields, Koç, Ç. K., Mesnager, S., Savaş, E., (ur.). Cham: Springer International Publishing, 2015, str. 83–107.

[105]Carlet, C., Chen, X., Qu, L., "Constructing Infinite Families of Low Differential Uniformity (n,m)-Functions with $m > n/2$", preprint. 2018.

[106]Chanson, S., Ding, C., Salomaa, A., "Cartesian Authentication Codes from Functions with Optimal Nonlinearity", Theor. Comput. Sci., Vol. 290, No. 3, Jan. 2003, str. 1737–1752, dostupno na: http://dx.doi.org/10.1016/S0304-3975(02)00077-4

[107]Carlet, C., Ding, C., Yuan, J., "Linear codes from perfect nonlinear mappings and their secret sharing schemes", IEEE Transactions on Information Theory, Vol. 51, No. 6, June 2005, str. 2089–2102.

[108]Carlet, C., Ding, C., "Highly Nonlinear Mappings", J. Complex., Vol. 20, No. 2-3, Apr. 2004, str. 205–244, dostupno na: http://dx.doi.org/10.1016/j.jco.2003.08.008

[109]Picek, S., Knezevic, K., Jakobovic, D., Carlet, C., "A search for differentially-6 uniform (n, n-2) functions", in 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, str. 1-7.

[110]Nyberg, K., "Perfect Nonlinear S-Boxes", in Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton,

UK, April 8-11, 1991, Proceedings, ser. Lecture Notes in Computer Science, Vol. 547. Springer, 1991, str. 378–386.

[111]Clark, J. A., Jacob, J. L., Stepney, S., "The design of S-boxes by simulated annealing", New Generation Computing, Vol. 23, No. 3, Sep. 2005, str. 219–231, dostupno na: http://dx.doi.org/10.1007/BF03037656

[112]Millan, W., Burnett, L., Carter, G., Clark, A., Dawson, E., "Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes", in Information and Communication Security, ser. LNCS. Springer Berlin Heidelberg, 1999, Vol. 1726, str. 263–274.

[113]Tesa ř, P., "A New Method for Generating High Non-linearity S-Boxes", Radioengineering, Vol. 19, No. 1, Apr. 2010, str. 23–26.

[114]Picek, S., Ege, B., Batina, L., Jakobovic, D., Chmielewski, L., Golub, M., "On Using Genetic Algorithms for Intrinsic Side-channel Resistance: The Case of AES S-box", in Proceedings of the First Workshop on Cryptography and Security in Computing Systems, ser. CS2 '14. New York, NY, USA: ACM, 2014, str. 13–18, dostupno na: http://doi.acm.org/10.1145/2556315.2556319

[115]Picek, S., Mazumdar, B., Mukhopadhyay, D., Batina, L., "Modified Transparency Order Property: Solution or Just Another Attempt", in Security, Privacy, and Applied Cryptography Engineering - 5th International Conference, SPACE 2015, Jaipur, India, October 3-7, 2015, Proceedings, 2015, str. 210–227.

[116]Picek, S., Cupic, M., Rotim, L., "A New Cost Function for Evolution of S-Boxes", Evolutionary Computation, Vol. 24, No. 4, 2016, str. 695–718.

[117]Meyer, L. D., Bilgin, B., "Classification of balanced quadratic functions", Cryptology ePrint Archive, Report 2018/113, https://eprint.iacr.org/2018/113. 2018.

[118]Burnett, L., Carter, G., Dawson, E., Millan, W., "Efficient Methods for Generating MARS-Like S-Boxes", in Proceedings of the 7th International Workshop on Fast Software Encryption, ser. FSE '00. London, UK, UK: Springer-Verlag, 2001, str. 300–314, dostupno na: http://dl.acm.org/citation.cfm?id=647935.740914

[119]Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas, S. M., O'Connor, L., Peyravian, M., Safford, D., Zunic, N., "The MARS Encryption Algorithm", 1999.

[120]Picek, S., Sisejkovic, D., Jakobovic, D., "Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties", Engineering Applications of Artificial Intelligence, 2016.

[121]Abadi, M., Andersen, D. G., "Learning to protect communications with adversarial neural cryptography", CoRR, Vol. abs/1610.06918, 2016, dostupno na: http://arxiv.org/abs/1610.06918

[122]Picek, S., Knezevic, K., Jakobovic, D., Derek, A., "C3po: Cipher construction with cartesian genetic programming", in Proceedings of the Genetic and Evolutionary Computation Conference Companion, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, str. 1625–1633, dostupno na: https://doi.org/10.1145/3319619.3326869

[123]Clark, J. A., Jacob, J. L., Stepney, S., Maitra, S., Millan, W., "Evolving boolean functions satisfying multiple criteria", in INDOCRYPT 2002, ser. LNCS, Menezes, A., Sarkar, P., (ur.), Vol. 2551. Springer, 2002, str. 246–259.

[124]Mariot, L., Leporati, A., "Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications", in GECCO Companion '15. ACM, 2015, str. 1425–1426.

[125]Picek, S., Mariot, L., Yang, B., Jakobovic, D., Mentens, N., "Design of s-boxes defined with cellular automata rules", in Proceedings of the Computing Frontiers Conference, ser. CF'17. New York, NY, USA: ACM, 2017, str. 409–414, dostupno na: http://doi.acm.org/10.1145/3075564.3079069

[126]Picek, S., Miller, J. F., Jakobovic, D., Batina, L., "Cartesian Genetic Programming Approach for Generating Substitution Boxes of Different Sizes", in GECCO Companion '15. New York, NY, USA: ACM, 2015, str. 1457–1458.

[127]Kazymyrov, O., Kazymyrova, V., Oliynykov, R., "A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent", Cryptology ePrint Archive, Report 2013/578, 2013.

[128]Lamenca-Martinez, C., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., Ribagorda, A., "Lamar: A new pseudorandom number generator evolved by means of genetic programming", in Parallel Problem Solving from Nature - PPSN IX. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, str. 850–859.

[129]Picek, S., Sisejkovic, D., Rozic, V., Yang, B., Jakobovic, D., Mentens, N., "Evolving cryptographic pseudorandom number generators", in Parallel Problem Solving from Nature – PPSN XIV. Cham: Springer International Publishing, 2016, str. 613–622.

[130]Hernandez-Castro, J. C., Estevez-Tapiador, J. M., Ribagorda-Garnacho, A., Ramos-Alvarez, B., "Wheedham: An automatically designed block cipher by means of genetic

programming", in 2006 IEEE International Conference on Evolutionary Computation, 2006, str. 192–199.

[131]Ruttor, A., "Neural Synchronization and Cryptography", Doktorski rad, PhD Thesis, 2007, 2007.

[132]Sinha, A., Malo, P., Deb, K., "A review on bilevel optimization: from classical to evolutionary approaches and applications", IEEE Transactions on Evolutionary Computation, Vol. 22, No. 2, 2018, str. 276–295.

[133]Heuser, A., Rioul, O., Guilley, S., "Good is Not Good Enough — Deriving Optimal Distinguishers from Communication Theory", in CHES, ser. Lecture Notes in Computer Science, Batina, L., Robshaw, M., (ur.), Vol. 8731. Springer, 2014, dostupno na: http://dx.doi.org/10.1007/978-3-662-44709-3

[134]Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F., "Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis)", in COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers, 2015, str. 20–33.

[135]Heuser, A., Zohner, M., "Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines", in COSADE, ser. LNCS, Schindler, W., Huss, S. A., (ur.), Vol. 7275. Springer, 2012, str. 249–264.

[136]Cagli, E., Dumas, C., Prouff, E., "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing", in CHES 2017, Taipei, Taiwan, 2017, Proceedings, ser. LNCS, Fischer, W., Homma, N., (ur.), Vol. 10529. Springer, 2017, str. 45–68.

[137]Maghrebi, H., Portigliatti, T., Prouff, E., "Breaking Cryptographic Implementations Using Deep Learning Techniques", in SPACE 2016, Hyderabad, India, 2016, Proceedings, ser. Lecture Notes in Computer Science, Carlet, C., Hasan, M. A., Saraswat, V., (ur.), Vol. 10076. Springer, 2016, str. 3–26.

[138]Heyszl, J., Ibing, A., Mangard, S., Santis, F. D., Sigl, G., "Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations", in CARDIS, ser. Lecture Notes in Computer Science. Springer, November 2013, berlin, Germany.

[139]Lerman, L., Medeiros, S. F., Veshchikov, N., Meuter, C., Bontempi, G., Markowitch, O., "Semi-supervised template attack", in COSADE 2013, Paris, France, 2013, Revised Selected Papers, Prouff, E., (ur.). Springer Berlin Heidelberg, 2013, str. 184–199.

[140]Picek, S., Heuser, A., Jovi ć, A., Legay, A., Knezevic, K., "Profiled sca with a new twist: Semi-supervised learning", IACR Cryptol. ePrint Arch., Vol. 2017, 2017, str. 1085.

[141]Schwenker, F., Trentin, E., “Pattern classification and clustering: A review of partially supervised learning approaches.”, Pattern Recognition Letters, Vol. 37, 2014, str. 4–14, dostupno na: http://dblp.uni-trier.de/db/journals/prl/prl37.html#SchwenkerT14a

[142]Chapelle, O., Schlkopf, B., Zien, A., Semi-Supervised Learning, 1st ed. The MIT Press, 2010.

[143]Bengio, Y., Delalleau, O., Le Roux, N., “Efficient Non-Parametric Function Induction in Semi-Supervised Learning”, Département d’informatique et recherche opérationnelle, Université de Montréal, Tech. Rep. 1247, 2004, dostupno na: http://www.iro.umontreal.ca/~lisa/pointeurs/tr1247.pdf

[144]Mitchell, T. M., Machine Learning, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[145]Begum, S., Chakraborty, D., Sarkar, R., “Data Classification Using Feature Selection and kNN Machine Learning Approach”, in 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Dec 2015, str. 811–814.

[146]Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, Vol. 12, 2011, str. 2825–2830.

[147]Choudary, O., Kuhn, M. G., “Efficient template attacks”, in Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, 2013. Revised Selected Papers, ser. LNCS, Francillon, A., Rohatgi, P., (ur.), Vol. 8419. Springer, 2013, str. 253–270.

[148]TELECOM ParisTech SEN research group, “DPA Contest (2 nd edition)”, http://www.DPAcontest.org/v2/. 2009–2010.

[149]TELECOM ParisTech SEN research group, “DPA Contest (4 th edition)”, http://www.DPAcontest.org/v4/. 2013–2014.

[150]Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F., “The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations”, Cryptology ePrint Archive, Report 2018/476, https://eprint.iacr.org/2018/476. 2018.

[151]Kocher, P. C., “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, in Proceedings of CRYPTO’96, ser. LNCS, Vol. 1109. Springer-Verlag, 1996, str. 104–113.

[152]Kocher, P. C., Jaffe, J., Jun, B., “Differential power analysis”, in Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, ser. CRYPTO ’99. London, UK, UK: Springer-Verlag, 1999, str. 388–397, dostupno na: http://dl.acm.org/citation.cfm?id=646764.703989

[153]Quisquater, J.-J., Samyde, D., “Electromagnetic analysis (ema): Measures and countermeasures for smart cards”, in Smart Card Programming and Security, Attali, I., Jensen, T., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, str. 200–210.

[154]Kocher, P., Jaffe, J., Jun, B., “Differential power analysis”, in Advances in Cryptology — CRYPTO’ 99, Wiener, M., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, str. 388–397.

[155]Chari, S., Rao, J. R., Rohatgi, P., “Template attacks”, in Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, ser. Lecture Notes in Computer Science, Kaliski, B. S., Jr., Koç, Ç. K., Paar, C., (ur.), Vol. 2523. Springer, 2002, str. 13–28.

[156]Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A., “Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis”, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019, str. 148–179.

[157]Zaid, G., Bossuet, L., Habrard, A., Venelli, A., “Methodology for efficient cnn architectures in profiling attacks”, IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2020, No. 1, Nov. 2019, str. 1–36.

[158]Elsken, T., Metzen, J. H., Hutter, F., “Neural architecture search: A survey”, 2019.

[159]Feurer, M., Springenberg, J. T., Hutter, F., “Initializing bayesian hyperparameter optimization via meta-learning”, in Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, ser. AAAI’15. AAAI Press, 2015, str. 1128–1135.

[160]Finn, C., Abbeel, P., Levine, S., “Model-agnostic meta-learning for fast adaptation of deep networks”, in Proceedings of the 34th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, Precup, D., Teh, Y. W., (ur.), Vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, str. 1126–1135.

[161]Finn, C., Levine, S., “Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm”, 2018.

[162]Gonzalez, S., Miikkulainen, R., “Improved training speed, accuracy, and data utilization through loss function optimization”, 2020.

[163]Wistuba, M., Rawat, A., Pedapati, T., “A survey on neural architecture search”, 2019.

[164]Chen, X., Xie, L., Wu, J., Tian, Q., “Progressive differentiable architecture search: Bridging the depth gap between search and evaluation”, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2019.

[165]Zoph, B., Le, Q. V., “Neural architecture search with reinforcement learning”, 2017.

[166]Dauphin, Y. N., Fan, A., Auli, M., Grangier, D., “Language modeling with gated convolutional networks”, CoRR, Vol. abs/1612.08083, 2016.

[167]Maas, A. L., “Rectifier nonlinearities improve neural network acoustic models”, 2013.

[168]Nair, V., Hinton, G. E., “Rectified linear units improve restricted boltzmann machines”, in Proceedings of the 27th International Conference on International Conference on Machine Learning, ser. ICML’10. Madison, WI, USA: Omnipress, 2010, str. 807–814.

[169]Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S., “Activation functions: Comparison of trends in practice and research for deep learning”, 2018.

[170]Kneževi ć, K., Fulir, J., Jakobović, D., Picek, S., Đurasević, M., “Neurosca: Evolving activation functions for side-channel analysis”, IEEE Access, Vol. 11, 2023, str. 284-299.

[171]Standaert, F.-X., Malkin, T. G., Yung, M., “A unified framework for the analysis of side-channel key recovery attacks”, in Advances in Cryptology - EUROCRYPT 2009, Joux, A., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, str. 443–461.

[172]Lerman, L., Medeiros, S. F., Bontempi, G., Markowitch, O., “A Machine Learning Approach Against a Masked AES”, in CARDIS, ser. Lecture Notes in Computer Science. Springer, November 2013, berlin, Germany.

[173]Picek, S., Heuser, A., Jovic, A., Ludwig, S. A., Guilley, S., Jakobovic, D., Mentens, N., “Side-channel analysis and machine learning: A practical perspective”, in 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017, 2017, str. 4095–4102.

[174]Picek, S., Heuser, A., Guilley, S., “Template attack versus Bayes classifier”, J. Cryptogr. Eng., Vol. 7, No. 4, 2017, str. 343–351.

[175]Maghrebi, H., Portigliatti, T., Prouff, E., “Breaking cryptographic implementations using deep learning techniques”, in International Conference on Security, Privacy, and Applied Cryptography Engineering. Springer, 2016, str. 3–26.

[176]Cagli, E., Dumas, C., Prouff, E., "Convolutional neural networks with data augmentation against jitter-based countermeasures", in Cryptographic Hardware and Embedded Systems – CHES 2017, Fischer, W., Homma, N., (ur.). Cham: Springer International Publishing, 2017, str. 45–68.

[177]Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F., "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2019, No. 1, Nov. 2018, str. 209–237, dostupno na: https://tches.iacr.org/index.php/TCHES/article/view/7339

[178]Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C., "Deep learning for side-channel analysis and introduction to ASCAD database", J. Cryptographic Engineering, Vol. 10, No. 2, 2020, str. 163–188.

[179]Perin, G., Chmielewski, L., Picek, S., "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2020, No. 4, Aug. 2020, str. 337–364.

[180]Wu, L., Perin, G., Picek, S., "I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis", Cryptology ePrint Archive, Report 2020/1293, 2020.

[181]Rijsdijk, J., Wu, L., Perin, G., Picek, S., "Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2021, No. 3, Jul. 2021, str. 677–707, dostupno na: https://tches.iacr.org/index.php/TCHES/article/view/8989

[182]Wouters, L., Arribas, V., Gierlichs, B., Preneel, B., "Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2020, No. 3, Jun. 2020, str. 147–168.

[183]Pfeifer, C., Haddad, P., "Spread: a new layer for profiled deep-learning side-channel attacks", Cryptology ePrint Archive, Report 2018/880, 2018.

[184]Zhang, J., Zheng, M., Nan, J., Hu, H., Yu, N., "A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2020, No. 3, Jun. 2020, str. 73–96.

[185]Zaid, G., Bossuet, L., Dassance, F., Habrard, A., Venelli, A., "Ranking loss: Maximizing the success rate in deep learning side-channel analysis", IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2021, No. 1, Dec. 2020, str. 25–55, dostupno na: https://tches.iacr.org/index.php/TCHES/article/view/8726

[186]Liu, Y., Yao, X., "Evolutionary design of artificial neural networks with different nodes", in Proceedings of IEEE International Conference on Evolutionary Computation, 1996, str. 670–675.

[187]Marchisio, A., Hanif, M. A., Rehman, S., Martina, M., Shafique, M., "A methodology for automatic selection of activation functions to design hybrid deep neural networks", 2018.

[188]Stanley, K. O., Miikkulainen, R., "Evolving neural networks through augmenting topologies", Evolutionary Computation, Vol. 10, No. 2, 2002, str. 99–127.

[189]Hagg, A., Mensing, M., Asteroth, A., "Evolving parsimonious networks by mixing activation functions", str. 425 – 432, 2017.

[190]Ramachandran, P., Zoph, B., Le, Q. V., "Searching for activation functions", 2017.

[191]Bingham, G., Macke, W., Miikkulainen, R., "Evolutionary optimization of deep learning activation functions", in Proceedings of the 2020 Genetic and Evolutionary Computation Conference, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, str. 289–296.

[192]Basirat, M., Roth, P. M., "The quest for the golden activation function", CoRR, Vol. abs/1808.00783, 2018.

[193]Nair, V., Hinton, G. E., "Rectified linear units improve restricted boltzmann machines", in Proceedings of the 27th International Conference on International Conference on Machine Learning, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, str. 807–814.

[194]Maas, A. L., Hannun, A. Y., Ng, A. Y., "Rectifier nonlinearities improve neural network acoustic models", in in ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[195]Konda, K., Memisevic, R., Krueger, D., "Zero-bias autoencoders and the benefits of co-adapting features", 2015.

[196]He, K., Zhang, X., Ren, S., Sun, J., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", in Proceedings of the 2015 IEEE International

Conference on Computer Vision (ICCV), ser. ICCV '15. USA: IEEE Computer Society, 2015, str. 1026–1034.

[197]Clevert, D., Unterthiner, T., Hochreiter, S., "Fast and accurate deep network learning by exponential linear units (elus)", in 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, Bengio, Y., LeCun, Y., (ur.), 2016.

[198]Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S., "Self-normalizing neural networks", in Proceedings of the 31st International Conference on Neural Information Processing Systems, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, str. 972–981.

[199]Klabjan, D., Harmon, M., "Activation ensembles for deep neural networks", in 2019 IEEE International Conference on Big Data (Big Data), 2019, str. 206–214.

[200]Manessi, F., Rozza, A., "Learning combinations of activation functions", in 2018 24th International Conference on Pattern Recognition (ICPR), 2018, str. 61–66.

[201]Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., Gagné, C., "DEAP: Evolutionary algorithms made easy", Journal of Machine Learning Research, Vol. 13, jul 2012, str. 2171–2175.

[202]Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., "Pytorch: An imperative style, high-performance deep learning library", in Advances in Neural Information Processing Systems 32, Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., (ur.). Curran Associates, Inc., 2019, str. 8024–8035.

[203]Wolpert, D. H., Macready, W. G., "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, April 1997, str. 67–82.

[204]Renauld, M., Standaert, F., Veyrat-Charvillon, N., Kamel, D., Flandre, D., "A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices", in Advances in Cryptology - EUROCRYPT 2011, Tallinn, Estonia, 2011. Proceedings, ser. Lecture Notes in Computer Science, Paterson, K. G., (ur.), Vol. 6632. Springer, 2011, str. 109–128.

[205]Choudary, O., Kuhn, M. G., "Template Attacks on Different Devices", in Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE

2014, Paris, France, April 13-15, 2014. Revised Selected Papers, Prouff, E., (ur.). Cham: Springer International Publishing, 2014, str. 179–198.

Nomenclature

Monte Carlo Algorithms

CGPCartesian Genetic Programming

ESEvolutionary Strategy

GAGenetic Algorithm

GPGenetic Programming

Cryptography

(n,m)-functionFunction F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$

δ_F Differential delta uniformity of S-box F

$\varphi(n)$ Euler function

N_F Nonlinearity of S-box F

N_f Nonlinearity of Boolean function f

W_F Walsh-Hadamard transform of S-box F

W_f Walsh-Hadamard transform of Boolean function f

AESAdvanced Encryption Standard

APNAlmost Perfect Nonlinear

ARXAddition, rotation and exclusive OR

CACollision attacks

CMOSComplementary metal-oxide-semiconductor

CPACorrelation power analysis

DDTDifference distribution table

DOMDifference of means

DPADifferential power analysis

ECDLPElliptic curve discrete logarithm problems

FAFault attack

GEGuessing entropy

HDHamming distance

HWHamming weight

KPAKnown Plaintext Attack

LFSRLinear feedback shift register

LRALinear regression analysis

MC-CDMAMulticode code-division multiple access

OTPOne-time pad

PAAPower analysis attack

PDNPulldown network

PUNPullup network

RSARivest–Shamir–Adleman, public key cryptosystem

SCASide-channel attack

SNRSignal to noise ratio

SPASimple power analysis

SPNSubstitution Permutation Network

TATemplate attack

TA_p Pooled version of template attack

Evolutionary Computation

λ Number of offsprings in evolutionary strategy

μ Number of parents in evolutionary strategy

k Tournament size

l Levels-back parameter in CGP

M Number of generations

N Population size

n_c Number of columns in CGP

n_n Number of node input connections in CGP

n_o Number of graph output connections in CGP

n_r Number of rows in CGP

P_c Crossover probability

P_m Mutation probability

t_d Tree depth in genetic programming

Machine Learning

θ Model parameters

θ^* Estimated model parameters

$\vec{x}$ Feature vector

C A class of the example

k-NN k-nearest neighbor algorithm

N Number of examples

n Dimension of the example

X Set of all possible examples

ACCAccuracy

ANNArtificial neural networks

CTrade-off between the size of the margin and the total penalty in SVM

CNNConvolutional neural network

FNFalse negative classification

FPFalse positive classification

i.i.d.Independent and identical distribution of examples

MLPMultilayer perceptron

NBNaive Bayes

ReLUrectified linear unit

SMOSequential minimal optimization

SSLSemi-supervised learning

SVMSupport vector machines

TNTrue negative classification

TPTrue positive classification

Mathematical Symbols and Operators

$\mathrm{O}(\cdot)$ Upper bound, $f(n) = \mathrm{O}\big(g(n)\big)$ if $\exists$ some constant c: $f(n) \leq c \cdot g(n)$

$\in$ Member of

$\mathbb{F}$ Field (arbitrary)

$\mathbb{F}_2$ Galois field with 2 elements

$\mathbb{F}_2^n$ n-dimensional vector space over $\mathbb{F}_2$

$\mathbb{N}$ Ring of natural numbers

$\mathbb{Z}$ Ring of integers

$\mathbb{Z}_q$ Ring of integers modulo q

$\oplus$ Addition modulo 2 (exclusive OR)

$\vec{a} \cdot \vec{b}$ Inner product of vectors $\vec{a}$ and $\vec{b}$

$a \to b$ Mapping a to b

$P(x|y)$ Conditional probability

$P(x,y)$ Joint probability

$tanh(x)$ hyperbolic tangent function

:Such that

TTTruth table

Index

List of Figures

List of Tables

List of Algorithms

Biography

Karlo Knežević was born in Zagreb on July 4, 1989. He graduated from the Dragutin Domjanić Elementary School and the XV. science and mathematics high school. At the Faculty of Electrical Engineering and Computing of the University of Zagreb, he completed undergraduate and graduate studies in computing, majoring in computer science. The graduate study ends with honors among the 10% of the best students in the generation.

For several years, he worked as a software engineer and head of the development and research department at Sofa IT. As an assistant and researcher on the EvoCrypt project, he worked for several years at the Department of Electronics, Microelectronics, Computer and Intelligent Systems of the Faculty of Electrical Engineering and Computing and as an external associate at the College of Algebra. He later worked at the University of Algebra as a lecturer and leader of several specialist educations. He is also the owner of a business for IT services and consulting. He currently works as the head of the artificial intelligence team at Sofa IT. So far, he has been a mentor for a dozen undergraduate and graduate theses and a reviewer at several international conferences.

He has published ten articles in journals and conferences with international reviews. He participated in several international training courses at prestigious conferences and universities. He received several scholarships during his higher education. He is a member of the international technical organizations IEEE and ACM.

He is married and the father of one child.

List of published works

Papers in Journals

1.Kneževi ć, K., Fulir, J., Jakobović, D., Picek, S., Đurasević, M., NeuroSCA: Evolving Activation Functions for Side-channel Analysis, IEEE Access, Vol. 11, December 2022, 284-299.

2.Đurasevi ć, M., Jakobović, D., Knežević, K., Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing, Vol. 48, November 2016, 419-430.

Papers at International Scientific Conferences

1.Kneževi ć, K., Generating Prime Numbers Using Genetic Algorithms, 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), September 2021, 1224-1229.

2.Kneževi ć, K., Picek, S., Jakobović, D., Hernandez-Castro, J., What Is Your MOVE: Modeling Adversarial Network Environments, EvoApplications 2020: Applications of Evolutionary Computation, April 2020, 260–275.

3.Picek, S., Kneževi ć, K., Jakobović, D., Đerek, A., C^3PO: cipher construction with cartesian genetic programming, GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion, July 2019, 1625–1633.

4.Kneževi ć, K., Picek, S., Mariot, L., Jakobović, D., Leporati, A., The Design of (Almost) Disjunct Matrices by Evolutionary Algorithms, TPNC 2018: Theory and Practice of Natural Computing, December 2018, 152-163.

5.Picek, S., Heuser, A., Jovi ć, A., Knežević, K., Richmond, T., Improving Side-Channel Analysis Through Semi-supervised Learning, CARDIS 2018: Smart Card Research and Advanced Applications, November 2018, 35-50.

6.Picek, S., Kneževi ć, K., Jakobović, D., Carlet, C., A Search for Differentially-6 Uniform (n, n-2) Functions, 2018 IEEE Congress on Evolutionary Computation (CEC), July 2018, 1-7.

7.Picek, S., Kneževi ć, K., Mariot, L., Jakobović, D., Leporati, A., Evolving Bent Quaternary Functions, 2018 IEEE Congress on Evolutionary Computation (CEC), July 2018, 1-8.

8.Kneževi ć, K., Picek, S., Miller, J. F., Amplitude-oriented mixed-type CGP classification, GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference Companion, July 2017, 1415–1418.

9.Picek, S., Kneževi ć, K., Jakobović, D., On the evolution of bent (n, m) functions, 2017 IEEE Congress on Evolutionary Computation (CEC), June 2017, 2137-2144.

10.Kneževi ć, K., Combinatorial Optimization in Cryptography, 2017 40th International Convention on Information, Communication and Electronic Technology (MIPRO), May 2017, 1324-1330.

Životopis

Karlo Knežević rođen je u Zagrebu, 4. srpnja 1989. Završio je Osnovnu školu Dragutina Domjanića te XV. prirodoslovno-matematičku gimnaziju. Na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu završio je preddiplomski i diplomski studij računarstva, smjer računarska znanost. Diplomski studij završava s pohvalom kao dio 10% najboljih studenata u generaciji.

Nekoliko godina radio je kao programski inženjer te voditelj odjela za razvoj i istraživanje u tvrtki Sofa IT. Kao asistent te istraživač na projektu EvoCrypt radio je nekoliko godina na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva, te vanjski suradnik na Visokom učilištu Algebra. Na Visokom učilištu Algebra radio je kasnije kao predavač i kao voditelj nekoliko specijalističkih edukacija. Ujedno je vlasnik obrta za informatičke usluge i savjetovanje. Trenutno radi kao voditelj tima za umjetnu inteligenciju u Sofa IT. Dosad je bio mentor na desetak preddiplomskih i diplomskih radova, kao i recenzent na više međunarodnih konferencija.

Objavio je desetak članaka u časopisima te konferencijama s međunarodnom recenzijom. Sudjelovao je na nekoliko međunarodnih usavršavanja na prestižnim konferencijama i sveučilištima. Dobitnik je više stipendija tijekom visokog obrazovanja. Član je međunarodnih tehničkih organizacija IEEE i ACM.

Oženjen je i otac jednog djeteta.