VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

KRYPTOANALÝZA POSTRANNÍCH KANÁLŮ POMOCÍ METOD HLUBOKÉHO UČENÍ

SIDE-CHANNEL CRYPTANALYSIS USING DEEP LEARNING METHODS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jakub Matuška

AUTHOR

VEDOUCÍ PRÁCE — Ing. Pavel Sikora

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Jakub Matuška ***ID:*** 211804

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Kryptoanalýza postranních kanálů pomocí metod hlubokého učení

POKYNY PRO VYPRACOVÁNÍ:

Student nastuduje a teoreticky popíše současné modely hlubokého učení pro profilující útoky postranními kanály na kryptografické algoritmy (př. AES). Vybranou nejvhodnější metodu implementuje ve vlastní aplikaci, která bude umožňovat natrénování vybraného modelu a následné provedení útoku pomocí natrénovaného modelu. K trénování a validaci modelu využije veřejně dostupné datasety (př. ASCAD nebo DPA COntest). Dále student navrhne vlastní model, který porovná vhodnou metrikou s vybraným modelem.

DOPORUČENÁ LITERATURA:

[1] Study of deep learning techniques for side-channel analysis and introduction to ascad database. Emmanuel, Prouff, et al. CoRR, 1-45, 2018.

[2] Keras. Chollet, Francois and others. 2015. [cit. 2022-09-09] Dostupné z: https://keras.io

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: Ing. Pavel Sikora

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Kryptografické systémy se z teoretického hlediska v dnešní době stávají neprolomitelnými. S limitovaným výpočetním výkonem je tak efektivnější útočit na jejich implementaci než na teoretický model jako takový. Přesně k tomu slouží útoky postranními kanály. S příchodem umělé inteligence se tyto útoky stávají velmi efektivními i proti různým druhům maskování a nahrazují dříve používané statistické metody. V dnešní době jsou útoky postranními kanály de facto vždy doprovázeny jednou či více metodami hlubokého učení. Tato práce představuje a prakticky ukazuje použití těchto metod v praxi. Přináší dodatečné nástroje pro trénování neuronových modelů a pro realizaci CPA a SITM útoků. V práci je představena analýza pomocí korelačního koeficientu a její význam pro vytváření nových architektur, společně s nástrojem pro její realizaci. Dále je představen návrh vlastní architektury pro útok na veřejně dostupný ASCAD dataset. Model s navrženou architekturou je následně porovnán vhodnou metrikou s vybranými referenčními modely. Na závěr, je představeno vylepšení útoku SITM metodami hlubokého učení, které je následně implementováno do konzolové aplikace.

KLÍČOVÁ SLOVA

analýza, ASCAD, CPA, DL-SCA, hluboké učení, kryptoanalýza, neuronové sítě, postranní kanály, SCA, SITM, umělá inteligence, útok

ABSTRACT

Cryptographic systems are getting unbreakable on paper. Therefore attacks on the implementations using side-channels are getting in front of others. Especially when neural networks (NN) got involved in this field. With deep learning, these attacks can recover secret keys even on implementations with countermeasures. Deep learning assisted side-channel analysis (DL-SCA) dominated this field over the statistical methods. That is why it is important to understand its concepts. This thesis will showcase these methods and introduce some new tools regarding correlation power analysis (CPA) and the training of NNs. An attack on ASCAD dataset will take place and the proposed NN to conduct this attack will be evaluated against other models using proper metrics. Lastly, improvements to SITM (See-In-The-Middle) attack using deep learning are proposed and implemented in the console application.

KEYWORDS

analysis, artificial intelligence, ASCAD, attack, CPA, cryptanalysis, deep learning, DL-SCA, neural networks, SCA, side-channel, SITM

MATUŠKA, Jakub. *Kryptoanalýza postranních kanálů pomocí metod hlubokého učení*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 69 s. Diplomová práce. Vedoucí práce: Ing. Pavel Sikora

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jakub Matuška
VUT ID autora:	211804
Typ práce:	Diplomová práce
Akademický rok:	2022/23
Téma závěrečné práce:	Kryptoanalýza postranních kanálů pomocí metod hlubokého učení

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Pavlu Sikorovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat rodině a blízkým přátelům, neboť kdyby bývávali nebyli, nebyl bych byl býval to dotáhl až sem. Měl bych jim koupit čokoládu...

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Moderní kryptografie přichází s algoritmy, stojícími na matematických důkazech složitosti jejich prolomení. Používané algoritmy se tak s aktuálním výpočetním výkonem dnešních počítačů staly teoreticky neprolomitelné a bezpečné. Každý teoretický algoritmus je ale potřeba implementovat do reálného světa na reálné procesory. Ty však mají něco s čím teoretický model nepočítá a sice fyzikální projevy do okolí. Tyto fyzikální projevy prozrazují dodatečné informace, které lze využít, k prolomení daného algoritmu. Tyto informační úniky se nazývají postranní kanály. Nad těmito kanály lze provádět analýzu a zkoumat, kolik informace uniká a zda je množství uniklé informace dostačující k prolomení daného algoritmu.

Právě v této oblasti, útoků postranními kanály, nastává v posledních letech velký posun zapříčiněný příchodem algoritmů umělé inteligence, konkrétně pak hlubokých neuronových sítí. Hledání korelace mezi uniklou informací a měřenou fyzikální veličinou, tak může být plně přenecháno neuronové síti, která v procesu trénování závislost nalezne sama. Není tak potřeba ručně vytvářet statistické modely pro odhad této závislosti, jak tomu bylo před používáním neuronových sítí.

Tento trend používání neuronových sítí pro útoky postranními kanály dosahuje velmi dobrých výsledků oproti dřívějším metodám a současné vědecké práce potvrzují tento trend. Používání neuronových sítí v tomto oboru se stalo všeobecně uznávaným standardem.

V rámci diplomové práce bude nejprve popsán potřebný teoretický základ samotných postranních kanálu a neuronových sítí, resp. hlubokého učení. Následující kapitola témata spojí do jedné a představí útoky postranními kanály za pomocí metod hlubokého učení. V další kapitole bude popsán útok SITM, obecné kroky a jejich aplikace na AES. Následně bude předložena teorie analýzy náměrů korelačním koeficientem a její význam pro budování modelů hlubokého učení. Bude představena vytvořená aplikace pro vykonávání takové analýzy a bude provedena ukázková analýza. V další části se práce bude zabývat útokem na veřejný ASCAD dataset metodami hlubokého učení. Bude navržen vlastní model, který bude vhodnou metrikou porovnán s vybranými referenčními modely. V závěru práce bude představen návrh k vylepšení útoku SITM pomocí metod hlubokého učení. A budou popsány vytvořené aplikace umožňující trénování navržených architektur a realizaci útoku SITM vylepšený metodami hlubokého učení.

1 Postranní kanály

Postranní kanál je klasický komunikační kanál ve smyslu přenosu informací. Tento komunikační kanál se nazývá postranním, neboť jím neprochází informace o systému jako takovém, ale pouze metadata o stavu či procesech tohoto systému. Těmito metadaty může být například váha bankovního trezoru. Z těchto metadat, lze pak odvodit užitečnou informaci o daném systému. Například kolik zlata se v trezoru nachází. Toto odvození je však námi vytvořené a není zaručené, že je pravdivé. Za předpokladu, že v trezoru je pouze zlato, můžeme z váhy trezoru (metadat), informaci o množství zlata v trezoru, přímo odvodit. Avšak v trezoru pouze zlato být nemusí. Může se v něm nacházet například kamení, které tam majitel dal právě pro ztížení odvození užitečné informace o hodnotě zlata v trezoru. Dalším příkladem může být zvukový postranní kanál, kdy z nahraného zvuku klávesnice je útočník schopen odvodit původně napsaný text [1]. V informačním světě takovými postranními kanály mohou být:

- Velikost šifrovaného paketu,
- rychlost a četnost komunikace,
- velikost procesu v paměti,
- doba zpracování instrukce,
- chybové hlášky.

To zda je útočník z postranního kanálu schopen odvodit užitečnou informaci, případně s jakou pravděpodobností je schopen zjistit, že dané odvození je pravdivé, je oblast kterou se zabývá kryptoanalýza postranních kanálů (Side-channel Analysis – SCA).

1.1 Postranní kanály v kryptografii

V kryptografii se únik informace postranním kanálem využívá k prolomení dané implementace kryptografického algoritmu. Při implementaci kryptografického algoritmu, ať už softwarové či hardwarové, vznikají postranní kanály. Tyto kanály jsou tvořeny fyzikálními veličinami. Měřením těchto veličin lze získat metadata o daném kryptosystému. Pokud existuje závislost mezi těmito metadaty a vnitřním stavem či procesem kryptosystému, pak lze sestavit model, jež převede metadata na informaci o systému. Postranními kanály typicky uniká pouze malé množství informace. Proto se cílí na odvození závislosti metadat na tajném klíči příslušného kryptosystému, na místo celého otevřeného textu.[1] Se znalostí tajného klíče je možno dešifrovat šifrový text a získat tak text otevřený. Není potřeba znát celý klíč, neboť zbytek klíče může

[1] Při šifrování většího množství dat je velikost tajného klíče značně menší než velikost otevřeného textu.

být dohledán metodou hrubé síly, tedy postupným zkoušení všech možností. Předpokládá se, že klíče mají dostatečnou velikost, tak aby útokem hrubou silou nebyl realizovatelný v polynomiálním čase s aktuálním výpočetním výkonem. Při znalosti části klíče se počet nutných pokusů pro jeho uhádnutí snižuje. To znamená, že i malé množství uniklé informace postranním kanálem může znamenat prolomení daného kryptosystému. [2]

1.1.1 Druhy postranních kanálů

Jednotlivé druhy postranních kanálu se liší podle typu měřené fyzikální veličiny. V kryptografii jsou nejčastějšími druhy postranních kanálu: časový, proudový, elektromagnetický a optický. [3]

Časový postranní kanál

U časového postranního kanálu se měří doba vykonávání operací závislých na tajné informaci, například doba zpracování autentizační metody. Tohoto postranního kanálu lze využít všude, kde je doba zpracování závislá na tajné informaci. To může nastat například při jednoduché implementaci algoritmu ověření hesla. Kde se jednotlivé symboly zadaného hesla kontrolují postupně od prvního po poslední, jeden po jednom se symboly z hesla uloženého v databázi. Neshoda symbolu na konkrétní pozici znamená okamžité ukončení algoritmu a zamítnutí zadaného hesla. Taková implementace je náchylná na útok pomocí časového postranního kanálu. Útočníkovi stačí měřit dobu vykonávání takového algoritmu a měnit pouze první číslici, dokud se doba vykonávání výrazně nezvětší. Pak je jasné, že ověřovací algoritmus přešel na kontrolu další číslice. Útočník tak ví, že aktuálně použitá první číslice byla zadána správně. Číslici zafixuje a přejde na další. Takto útočník hádá jednu číslici za druhou, namísto hádání celé sekvence číslic najednou. Takto si sníží složitost uhodnutí hesla z 9^n na $9n$, pří délce hesla n. [4]

Proudový postranní kanál

U proudového postranního kanálu se měří proudová spotřeba kryptografického zařízení, například mikroprocesor vykonávající kryptografický algoritmus. Procesor je tvořen z tranzistorů. Jejich propojení je složité a komplexní. Jsou napájeny konstantním napájecím napětím, ze kterého odebírají proud potřebný pro svou práci. Tento proud je závislý na zpracovávaných datech – na hodnotách jednotlivých bajtů. Pokud existuje tato závislost odebíraného proudu na zpracovávaných datech, pak se

lze, měřením tohoto proudu a vymodelováním zpětné transformace závislosti, dostat zpět ke zpracovávaným datům. Obor, který zkoumá proudový odběr a jeho zranitelnost se nazývá proudová analýza (Power Analysis – PA). [5, 6]

Elektromagnetický postranní kanál

Změny proudu vyvolávají změny v elektromagnetickém (EM) poli. Jak již bylo řečeno, procesory jsou tvořeny z tranzistorů, resp. klopných obvodů. Změny v těchto obvodech vytváří změny v proudu a ty změny v EM poli. Změny v tomto poli lze měřit a následně hledat jejich závislosti na zpracovávaných datech, ze kterých lze odvodit původně zpracovávaná data. Tento postranní kanál byl objeven roku 1943 v rámci tajných služeb. Tehdy používané šifrovací zařízení 131-B2, při stisku klávesy emitovalo EM záření, ze kterého bylo možno odvodit stisknuté klávesy, tedy psaný text. National Security Agency (NSA) roku 2007 odtajnila dokument *TEMPEST: A Signal Problem* popisující tuto situaci. [7]

Optický postranní kanál

Při změně vnitřních stavů v procesoru je emitováno malé množství fotonů. Intenzitu a lokalitu těchto fotonů umí měřit speciální metoda nazývaná pikosekundová zobrazovací obvodová analýza (Picosecond Imaging Circuit Analysis – PICA) [8]. Zařízení umožňující měření touto metodou jsou velmi drahá a vzácná. Jedno takové zařízení vlastní Francouzská vesmírná agentura (Centre National d'Études Spatiales – CNES), kde při správné synchronizaci lze identifikovat přepínání jednotlivých tranzistorů v paměti na PIC16F84A mikrokontroléru [9]. Pokud se takto útočník zaměří na okamžik, kdy mikrokontrolér pracuje s klíčem, lze z naměřených fotonů hodnotu zpracovávaného klíče dopočítat. Okamžikem, kdy mikrokontrolér pracuje s klíčem může být například fáze `AddRoundKey` v algoritmu AES [10], kdy algoritmus provádí operaci XOR s bajty otevřeného textu a rundovního klíče.

1.2 Útoky postranními kanály

Útoky postranními kanály zneužívají únik informací postranními kanály fyzických zařízení. Nejčastěji cílí na odvození tajného hesla či klíče z jednoho nebo více druhů postranních kanálů. Pod zkratkou SCA se v zahraniční literatuře používá i pojem Side-channel attack, tedy útoky postranních kanálů. Takové označení může působit zmatky, avšak oba pojmy se zabývají, jak z názvu vyplývá, stejnou problematikou pouze z jiného pohledu.

Největším problémem při útoku postranními kanály je správná synchronizace měření a cílového zařízení. Například u OpenSSL AES128 pouze 0,00028% z celkové doby šifrování je skutečně závislých na tajném klíči [11].

Útoky postranními kanály se dělí do dvou hlavních skupin a to na ne-profilující a profilující útoky.

1.2.1 Ne-profilující útoky

U tohoto typu útoků útočník nejprve naměří určitý počet náměrů postranního kanálu dané implementace, na kterou cílí. Nad těmito náměry pak provádí statistickou analýzu pro odhalení závislosti náměrů a tajné vnitřní informace. Tyto útoky bývají jednodušší na provedení, neboť útočník potřebuje pouze náměry. Avšak ne-profilující útoky dávají obecně horší výsledky než útoky profilující, kde je předpokládána plná kontrola nad fyzickým zařízením pod útokem. Do této kategorie patří jednoduchá proudová analýza (Simple Power Analysis – SPA) [5], diferenciální proudová analýza (Differential Power Analysis – DPA) [12], analýza korelačním koeficientem (Correlation Power Analysis – CPA) [13].

1.2.2 Profilující útoky

Profilující útoky předpokládají přístup k umělé kopii cílového zařízení. Takové referenční zařízení pak útočníkovi dovolí ladit jednotlivé kroky a parametry implementace na kterou útočí. Díky tomu je schopen se na útok lépe připravit. Tyto útoky mívají oproti ne-profilujícím lepší výsledky, avšak na útočníka jsou kladeny větší nároky.

Profilující útok se skládá ze dvou částí a to vytváření profilu a samotného útoku. V první fázi je využito referenčního zařízení pro podrobnou analýzu postranního kanálu, nalezení závislosti měřených hodnot a tajné informace, vytvoření modelu a jeho naprofilování na danou implementaci. V druhé fázi je připravený model z předchozí fáze použit nad náměry z reálného cílového zařízení. Výsledkem je pak odhad tajné informace (klíče). Příkladem jsou útoky pomocí šablon (Template Attack) založené na Gaussově modelu [14] a útoky založené na stochastických modelech [15]. Útoky využívající neuronové sítě, které jsou ústředním tématem této diplomové práce, také spadají do kategorie profilujících útoků. Převážně pro jejich potřebu dopředného natrénování a až následné možnosti útoku.

1.3 Protiopatření

Protiopatření (countermeasure) jsou metody a techniky pro zamezení úniku informace postranním kanálem, resp. ztížení nalezení závislosti mezi měřenou veličinou a citlivou vnitřní informací. Mají za cíl vytvořit nezávislost mezi spotřebou a aktuálně zpracovávanou proměnou. Obecně se tyto techniky dělí do dvou skupin: maskování (masking) [16, 17] a skrývání (hiding) [18].

1.3.1 Skrývání

Skrývání se snaží zamaskovat místa úniku informace, tak aby bylo pro útočníka těžké unikající informaci přečíst. To zajišťují buď v časové nebo amplitudové rovině.

Pro úspěšný útok je potřeba mít náměry správně zarovnané (synchronizované). Pokud tomu tak není, útočník potřebuje značně více náměrů pro úspěšný útok. Skrývání v časové doméně využívá přesně tohoto efektu a cíleně zanáší časové posuny vykonávaných operací, pro zanesení desynchronizace do náměru, a tedy ztížení případného útoku. Příkladem mohou být techniky jako náhodné vkládání nadbytečných operací, vyčkávání náhodného časového intervalu mezi jednotlivými operacemi nebo cílená záměna (shuffling) jednotlivých na sobě nezávislých operací, typicky pořadí substituce jednotlivých bajtů pomocí S-boxu.

Druhá skupina skrývajících technik upravuje amplitudu náměru, aby skryla závislosti amplitudy na vnitřní zpracovávané veličině. Toho lze docílit buď zajištěním konstantní hodnoty měřené veličiny nebo naopak hodnoty plně znáhodnit, aby mohly nabývat jakýchkoliv hodnot v jakémkoliv čase. [18]

1.3.2 Maskování

Maskování oproti skrývání redukuje závislost měřené veličiny na citlivé proměnné maskováním samotné citlivé proměnné. Maskování lze implementovat čistě softwarově. [18]

Základní princip maskování jako protiopatření proti SCA spočívá v přepočítání citlivé proměnné v pomocí dočasné masky m před samotným zpracováním citlivé proměnné. Na výstupu algoritmu je však potřeba masku m opět odstranit, tak aby maskování mělo vliv pouze na vnitřní výpočet, ale ne na výsledek celého kryptografického algoritmu. Ke zpětnému přepočtu slouží inverzní maska m'. Obecně můžeme proces maskování zapsat následovně:

$$v_m = v * m \tag{1.1}$$

proces demaskování pak,

$$v = v_m * m' \tag{1.2}$$

kde operace $*$ značí specifickou operaci podle použitého kryptografického algoritmu. Nejčastěji však $*$ značí exkluzivní OR (XOR), modulární součet či multiplikativní násobení. Příkladem konkrétních maskovacích technik jsou: boolovské maskování [19], multiplikativní maskování [20], maskování rotací S-boxu (Rotating S-Box Masking – RSM) [21, 22] a afinní maskování [23, 24].

2 Hluboké učení

Hluboké učení (Deep Learning – DL) je podmnožinou strojového učení (Machine Learning – ML) s učitelem (supervised learning). Učitel nejprve připraví datovou množinu – dataset – na které se model hlubokého učení bude trénovat. Jednotlivými záznamy v datasetu jsou vstupy a výstupy funkce, kterou má model aproximovat. Neuronové sítě (Neural Network – NN) se skládají z několika vrstev, dohromady tvořícími architekturu modelu. Hlubokými neuronovými sítěmi (Deep Neural Network – DNN) se rozumí neuronové sítě s velkým počtem těchto vnitřních vrstev.

2.1 Neuronové sítě obecně

Neuronové sítě obecně převádí určitý počet vstupních parametrů na určitý počet výstupů, prostřednictvím *vnitřních vrstev* složených z neuronů a vážených hran mezi nimi. Vstupní parametry jsou předávány první *vstupní vrstvě*. Výstup NN je pak realizován *výstupní vrstvou*, kde jednotlivé uzly představují možné výstupy sítě. Vstupem a výstupem NN jsou tedy vektory, matice a tenzory. Uspořádání (propojenost) vnitřních vrstev a váhy jednotlivých spojů určují závislost mezi jednotlivými vstupními parametry a výstupem. Tato závislost je tím, co je trénováno v procesu učení, tak aby NN dosahovala požadovaných výsledků. Námi požadované výsledky mohou být popsány funkcí $f(\vec{x})$, kde $\vec{x}$ je vektor vstupních parametrů. Funkce $f(\vec{x})$ generuje jednotlivé záznamy v datasetu a je reprezentací reálného problému. NN se pak snaží aproximovat tuto funkci $f(\vec{x})$ funkcí $\hat{f}(\vec{x})$. Čím lépe $\hat{f}(\vec{x})$ aproximuje $f(\vec{x})$, tím přesnějších výsledků NN dosahuje.

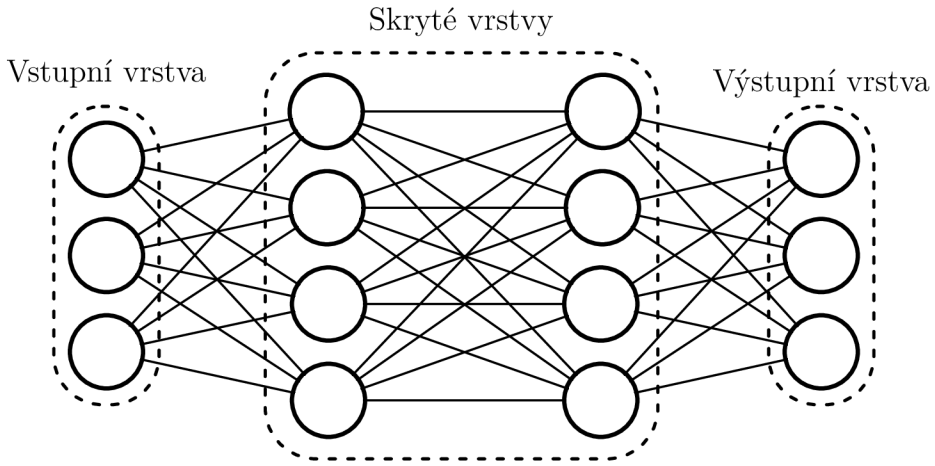
Každá NN potřebuje být nejprve natrénována na daný problém, který má řešit. Na začátku jsou jednotlivé váhy hran NN inicializovány náhodně[1] [25]. Závislost mezi vstupem a výstupem je náhodná – $\hat{f}(\vec{x})$ je náhodná. NN nevykonává žádnou specifickou funkci (nic neumí). Postupným učením na známých datech, se vnitřní parametry NN nastaví tak, aby lépe aproximovala funkci $f(\vec{x})$. [26, 27]

2.1.1 Neuron

Neuronová síť je tvořena z vrstev neuronů (perceptronů). Inspirace byla převzata z biologie od reálných mozkových neuronů. Perceptron má n vstupů (synapsí), jeden výstup a aktivační funkci. Vnitřní potenciál neuronu ξ se vypočítá dle vzorce 2.1:

$$\xi = \sum_{i=1}^{n} x_i w_i + b \tag{2.1}$$

[1] Váhy jsou ve většině případech inicializovány náhodně, ale mohou být inicializovány i jinými způsoby například pomocí Xavier a He metody.

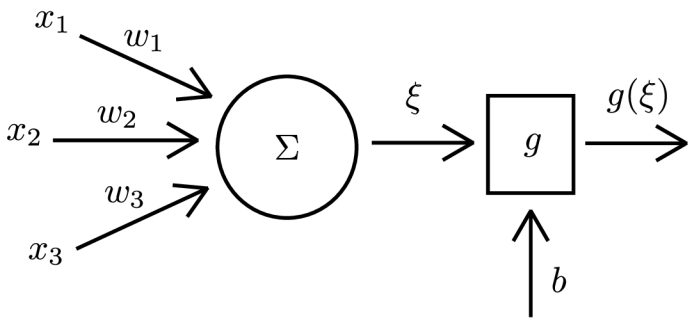


Obr. 2.1: Neuronová síť.

kde:

- x_i označuje výstupní hodnotu i-tého předchozího neuronu,
- w_i váhu spojení s i-tým předchozím neuronem a
- b je práh aktuálního neuronu (bias).

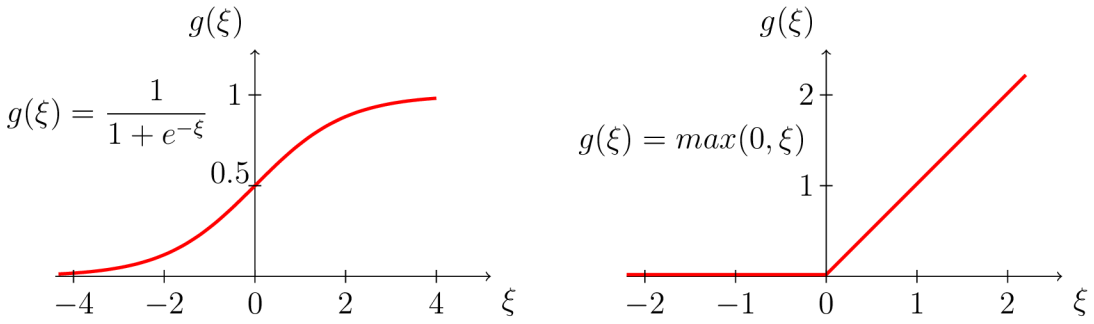
Vnitřní potenciál ξ se použije jako vstup aktivační funkce $g(\xi)$ a výsledek je výstupní hodnotou neuronu. [27]



Obr. 2.2: Neuron.

2.1.2 Aktivační funkce

Aktivační funkce definuje, kdy je konkrétní neuron aktivní, resp. jakou hodnotu výstupu má. Aktivační funkce zavádí do modelu nelinearitu. Jednotlivé vrstvy mohou být vyjádřeny funkcí $f_1(x)$, kde x jsou vstupní parametry. Složení více vrstev za sebe lze vyjádřit jako vložené funkce $f(x) = f_3(f_2(f_1(x)))$. Funkce $f(x)$ pak realizuje převedení vstupu na výstup prostřednictvím vnitřních vrstev. Pokud by jednotlivé funkce (vrstvy) byly lineární, poté by i výsledná $f(x)$ musela být lineární funkcí.



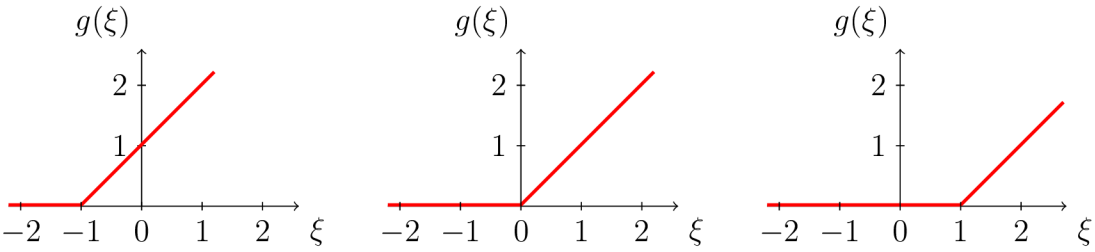
(a) Sigmoid (b) ReLU

Obr. 2.3: Průběhy aktivačních funkcí.

Pak by se všechny vnitřní funkce mohli být nahrazeny jednou lineární funkcí. Model by tak nebyl schopný aproximovat složitější funkce než lineární. Proto jsou aktivační funkce kritické a mají značný vliv na průběh trénování. Jako příklad lze uvést funkci Sigmoind a ReLU. Jejich průběh je možno vidět na obrázku 2.3. [27, 28, 29]

2.1.3 Práh neuronu

Práh neuronu (bias) vyjadřuje, kdy je neuron aktivován, resp. zrychluje nebo zpožďuje (posouvá doleva resp. doprava) aktivační funkci. Tento efekt je možno vidět na obrázku 2.4. To provádí přičtením konstantní hodnoty, tzv. prahu. Nenulový práh zajišťuje, že i neuron se všemi předchozími vazbami rovnými nule, může mít jiný než nulový výstup. Tak je zajištěno nešíření tzv. mrtvých neuronů. Takové neurony nemají žádný smysluplný výstup, resp jejich výstup je nezávislý na vstupu do neuronu. [27, 28, 29]



(a) Práh 1 (b) Práh 0 (c) Práh -1

Obr. 2.4: Efekt prahu neuronu na aktivační funkci.

2.1.4 Parametry neuronové sítě

Nastavitelné vnitřní parametry neuronové sítě jsou to, co umožňuje NN se učit z dat a řešit daný problém. Nastavení těchto parametrů udává chování sítě, resp. definuje vztah mezi vstupem a výstupem. Těmito parametry jsou především jednotlivé váhy hran (spojení) a prahové hodnoty (bias). Proces trénování má za účel automaticky tyto parametry nastavit tak, aby byla minimalizována ztrátová funkce. Způsob, jakým jsou vnitřní parametry sítě automaticky upravovány zajišťuje optimalizátor ve fázi zpětné propagace, viz kapitola 2.1.12. [27, 28, 29]

2.1.5 Hyper-parametry

Hyper-parametry jsou externím nastavením sítě a procesu trénování. Hyper-parametry mají zásadní vliv na trénování modelu, tedy na výsledky podávané modelem. Optimální hodnoty hyper-parametrů není možno odvodit z trénovacích dat a je nutné je volit experimentálně. Příkladem těchto hyper-parametrů je počet epoch, velikost dávek (batch size), hodnota rychlosti učení (learning rate), typ aktivační funkce a typ optimalizátoru. [27, 28, 29]

2.1.6 Generalizace

Stále přetrvávajícím problémem je, aby NN aproximovala funkci $f(\vec{x})$ nejen na trénovacím datasetu, ale i na validačních a testovacích datech jež nebyla použita při trénování. Tato schopnost se nazývá generalizace. Špatná generalizace může být zapříčiněna například malým množstvím trénovacích dat, ale i strukturou modelu nebo způsobem trénováním. S pojmem generalizace se úzce pojí podtrénování (underfitting) a přetrénování (overfitting).

Podstrénování je situace, kdy model nedosahuje dobrých výsledků ani na trénovacích datech. Model nedokáže nalézt závislosti obsažené v datech. To může nastat například v situaci, kdy trénování modelu neprobíhalo dostatečně dlouhou dobu.

Přetrénování, oproti tomu je situace opačná, kde si model začíná pamatovat vzorce v trénovacím datasetu. Dosahuje tak skvělých výsledků, avšak pouze na trénovacích datech. Na validačních datech, na kterých nebyla NN trénovaná, ale stále reprezentují stejný reálný problém, pak model není schopen dosahovat stejných výsledků. Model je v takovém případě příliš specifický na daný trénovací dataset a nemá potřebnou generalizaci. [27, 28, 29]

2.1.7 Dávky

Jednotlivé prvky datasetu (náměry) se mohou seskupovat do dávek (batches). K trénování modelu jsou použity tyto dávky namísto jednotlivých dat. Dávky umožňují

efektivnější paralelní výpočty pomocí grafické karty (Graphics Processing Unit – GPU). Velikost dávek má dopad na proces učení. Například trénování pomocí menších dávek je charakteristické menší přesností oproti trénování pomocí větších dávek. [27, 28, 29, 30]

2.1.8 Epochy

Jednou epochou se nazývá stav, kdy byl model natrénován na všech prvcích datasetu právě jednou. Jinak řečeno, v první epoše se model setkal se všemi prvky poprvé, v druhé epoše podruhé a tak dále. Počet epoch je významným hyper-parametrem a jeho volba ovlivňuje přetrénování a podtrénování sítě. Pokud při trénování použijeme nedostatečný počet epoch, bude model podtrénovaný. Naopak pokud použijeme epoch příliš mnoho, model se přetrénuje. Vhodným počtem epoch pro ukončení trénování může být bod, kde se ztrátová funkce na validační množině začne zvětšovat. Způsob takového ukončení trénování se nazývá brzké ukončení (Early Stopping). [27, 28, 29]

2.1.9 Ztrátová funkce

Pro určení, jak dobře NN aproximuje původní funkcí $f(\vec{x})$, slouží ztrátová funkce (Loss function). Vyjadřuje chybu odhadu modelu. Proces učení je řešením problému optimalizace této ztrátové funkce. Při trénování se snažíme dosáhnout co nejmenší hodnoty ztrátové funkce – co nejmenší chyby. [27, 28, 29]

2.1.10 Dopředná propagace

Provedením dopředné propagace se převedou vstupní parametry na výstup. Výpočet prochází od první vstupní vrstvy postupně, až k poslední výstupní vrstvě. Odtud název dopředná propagace. Tato fáze je charakteristická, jak pro trénování, tak i při samotném použití NN v praxi.

Pro rychlejší výpočet se používá akcelerace pomocí GPU. Ty jsou velmi efektivní při výpočtu opakujících se operací. To však platí v případě, že se výpočty dají vyjádřit maticově. GPU jsou konstruovány pro velmi efektivní paralelní výpočet maticových operací. Rovnici 2.1, popisující výpočet jednoho neuronu, převedeme na maticový počet celé vrstvy následovně:

- Všechny hodnoty předchozích neuronů (předchozí vrstvy) x převedeme na sloupcový vektor $x^{(1)} = \{x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}\}^T$, kde označení $x_1^{(1)}$ označuje hodnotu výstupu z prvního neuronu x_1 z první vrstvy (vstupní vrstva). Vektor $x^{(2)}$ by pak obsahoval všechny výstupy neuronů z druhé vrstvy.

- Všechny váhy poskládáme do matice W, kde jednotlivé sloupce budou představovat odpovídající spojení s předchozím neuronem a jednotlivé řádky pak konkrétní hodnoty vah pro daný neuron. I-tý řádek matice W je vektorem vah spojení s předchozím neuronem. Výsledek sumy ve vzorci 2.1 může být vyjádřen jako násobení dvou vektorů a sice i-tého řádku matice W a sloupcového vektoru $x^{(1)}$.
- Na konec je ještě potřeba přičíst hodnoty prahů jednotlivých neuronů. Všechny hodnoty prahů vložíme do sloupcového vektoru b a ten přičteme k násobku vektorů $W_i * x^{(1)}$.

Takto vznikne následující maticový počet:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2.2}$$

zkrácenou notací pak,

$$Wx^{(1)} + b. \tag{2.3}$$

Po použití aktivační funkce $g(.)$ na výsledek, tedy použití $g(.)$ na každý prvek výsledného vektoru, získáme vektor výsledných hodnot neuronů další vrstvy $x^{(2)}$:

$$x^{(2)} = g(Wx^{(1)} + b). \tag{2.4}$$

Ten bude použit jako vstup pro další vrstvu v dalším kroku. Takto se efektivně vypočítá celá dopředná propagace neuronové sítě pomocí GPU. [31]

2.1.11 Zpětná propagace

U zpětné propagace se upravují vnitřní parametry sítě (váhy) od poslední vrstvy k první vrstvě. Tato propagace je použita pouze pro úpravu parametrů sítě – trénování sítě. Podle ztrátové funkce se určí odchylka aktuální predikce (predikčního vektoru) od skutečné hodnoty. Tato odchylka je pak řídící jednotkou pro optimalizátor, který udává, jakým způsobem mají být jednotlivé parametry upraveny, pro co nejrychlejší zmenšení predikční chyby. [27, 28, 29]

2.1.12 Optimalizátor

Optimalizátor popisuje způsob, jak jsou vnitřní parametry (váhy) modelu upravovány za účelem co nejrychlejšího nastavení parametrů tak, aby měl model co

nejmenší chybu predikce určenou chybovou funkcí. Na začátku nevíme, jaké je optimální nastavení parametrů. Pomocí optimalizátoru a chybové funkce jsme schopni dosáhnout tohoto optima.

Klasickým příkladem optimalizátoru je **Gradientní sestup** (Gradient Descent, GD), který z náhodného bodu křivky chybové funkce sestupuje postupnými kroky do nižších a nižších hodnot až dosáhne minima. Směr je určen zápornou parciální derivací chybové funkce v daném bodě podle všech proměnných. Krok je určen parametrem zvaným **learning rate**, tedy rychlost učení. Krok je realizován upravením parametrů modelu tak, aby se posunul daným směrem o danou vzdálenost. Těmito postupnými kroky GD nalezne minimum chybové funkce – optimalizuje přesnost predikce daného modelu. Častým problémem bývá uvíznutí modelu v lokálním minimu. Postup gradientního sestupu jde vždy z daného bodu směrem největšího zmenšení chybové funkce. Tak ovšem optimalizátor nebere v potaz celou funkci a její globální minima, ale dosáhne pouze do nejbližšího lokálního minima. Tento problém se často řeší správnou volbou rychlosti učení (learning rate).

Dalším optimalizátorem je Stochastický gradientní sestup (Stochastic Gradient Descent, SGD). Ten používá dávky na místo celého datasetu a je tak méně výpočetně náročný oproti klasickému GD. Dále SGD využívá momentum, pro překonání chvilkových nárůstů a nalezení tak lepšího minima. Mezi další optimalizátory patří Adagrad [32], RMSprop [33] a Adam [34]. [27, 28, 29, 31]

2.2 Vícevrstvá perceptronová síť

Vícevrstvá perceptronová síť (Multi-Layer Perceptrons, MLP) je základní architektura neuronových sítí. Je tvořena vstupní vrstvou, několika plně propojenými vrstvami a výstupní vrstvou. Plně propojené vrstvy znamenají, že každý neuron z jedné vrstvy je propojen se všemi neurony vrstvy následující.

To kolik neuronů je v jednotlivých vrstvách a kolik takových vrstev je v modelu neuronové sítě obsaženo, závisí na komplexnosti řešeného problému. Čím více vrstev a čím více neuronů v těchto vrstvách, tím více vnitřních parametrů model má a tím je schopen řešit komplexnější problémy. Komplexnost sítě by měla odpovídat komplexitě řešeného problému. NN musí být dostatečně komplexní, aby dokázala daný problém řešit. Nesmí být však příliš komplexní, aby trénovaní netrvalo příliš dlouho a aby nedošlo k přetrénování sítě. Zároveň však platí, že pouhé přidávání komplexnosti nevyřeší všechny problémy. Nelze tak vyřešit neřešitelné problémy pouhým přidáváním několika vrstev. [27, 28, 29, 31]

2.3 Konvoluční neuronová síť

Konvoluční neuronové sítě (Convolutional Neural Network, CNN) [28] kombinují MLP s konvolučním filtrem a podvzorkovacími (pooling) vrstvami. Jsou vhodné pro hledání prostorových vzorů v 1D časových posloupnostech (time-series) [35], v 2D pixelových bitmapách [36] nebo i v 3D záznamech z dohledových videosystémů [37]. Prostorový vzor je extrahován pomocí filtru (kernelu). Filtr se aplikuje na původní obraz, tak že se jednotlivé prvky zarovnané na sebe násobí a výsledky jednotlivých součinů jsou sečteny. Tak vznikne jeden prvek nového obrazu s extrahovaným vzorem. Filtr se posune o určitý krok (stride) a opět se aplikuje pro extrakci dalšího prvku. Pro 1D obraz se použije 1D filtr, pro 2D obraz pak 2D filtr. Existují i 3D filtry pro 3D konvoluci sloužící k extrakci vzorů z prostoru i času najednou (spatiotemporal feature) [38]. Velikost filtru je většinou značně menší než obraz samotný (u 2D konvoluce jsou nejčastější rozměry filtru 3x3, 5x5 a 7x7).

Většinou se aplikuje na stejný obraz větší počet filtrů najednou (ze stejného obrazu se extrahují různé vzory). Počet použitých filtrů násobí počet vstupních parametrů do další vrstvy. Aby tomuto efektu bylo zabráněno a počet parametrů byl redukován, používají se podvzorkovací (pooling) vrstvy. Zde je opět použit filtr, nyní však pouze pro určení skupiny hodnot obrazu, ze kterých bude vybrána maximální, resp. průměrná hodnota. Při použití filtru 2x2 se ze skupiny čtyřech hodnot vybere právě jedna. Filtr se posune a výběr se provede znovu. Tak je počet parametrů zmenšen na čtvrtinu.

CNN vrství extrahované vzory za sebe. Tím se nejprve z jednoduchých vzorů (hran) postupně stávají komplexnější. Tak se abstrakce zvyšuje až po vrstvy, které pracují se vzory například pro detekci očí, nosu, dopravních značek, aut a podobně. Výstup tohoto řetězce konvolučních vrstev, proložených podvzorkovacími vrstvami, je přiveden na sérii plně propojených vrstev. Tyto vrstvy pak s detekovanými vzory pracují dál a vykonávají například samotnou klasifikaci nebo logické spojení detekovaných abstraktních vzorů. [27, 28, 29, 31]

3 Hluboké učení v SCA

Jak již bylo řečeno, NN potažmo DNN disponují schopností aproximovat komplexní funkce. V posledních letech se této vlastnosti začalo silně využívat v oblasti SCA. Taková kombinace, se pak označuje za útoky postranními kanály s pomocí metod hlubokého učení (Deep Learning Side-channel Attack – DL-SCA). Na místo ručního vytváření statistických modelů, pro určení závislosti mezi náměry a tajnou informací, se použije DL pro aproximaci funkce zobrazení jednotlivých náměrů do množiny možných klíčů $\mathcal{K}$. Většina DL-SCA modelů neutočí přímo na hodnotu klíče, ale na výstup S-boxu v první rundě. Klíč se až poté zpětně odvodí. Tato práce se zabývá útoky na implementace algoritmu AES.

Pro úspěšné natrénování (vytvoření profilu), DL potřebuje dostatečně velký počet náměrů postranního kanálu dané implementace. Náměr musí obsahovat část závislou na tajné informaci (klíči). U algoritmu AES to nejčastěji bývá operace `AddRoundKey` v první nebo poslední rundě. Jako cílové hodnoty (labels) jsou pak použity výstupy z S-boxu, tedy hodnoty po operaci `SubBytes`. Výstupem DNN je pak predikce těchto hodnot. Pro přílišnou velikost klíče se však obvykle používá princip rozděl a panuj. Útočí se pouze na jeden bajt klíče najednou, resp. DNN je natrénována pro klasifikaci hodnot možného klíče pouze v jednom bajtu. DNN v takovém případě klasifikuje náměr do 256 tříd na místo 2^{128} tříd, jak by tomu bylo u AES-128. DNN je tak možné natrénovat značně rychleji a přesněji. Pro získání celého klíče se musí natrénovat tolik DNN, kolika-ti bajtový je klíč. [39]

Při použití DL-SCA odpadá potřeba ručního nastavování parametrů predikčního modelu oproti klasickým SCA metodám. DL-SCA poskytuje dobré výsledky i v případech, kdy implementace používá protiopatření [40], dokonce i při použití více druhů protiopatření najednou. DL-SCA útoky nachází použité klíče značně rychleji, resp. za použití menšího počtu náměrů ve fázi útoku než klasické metody SCA. To je zároveň i důvod, proč je v současné době použití DL v SCA takovým trendem.

3.1 Výstup modelu

DL model je reprezentován jako aproximační funkce $\hat{f}(\vec{x})$, která pro vstup $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ (vektor naměřených hodnot, náměr), kde n je počet vzorků v náměru, určí pravděpodobnostní vektor $\vec{y}$ možných hodnot jednoho bajtu klíče. Vektor $\vec{y} = \{p_0, p_1, \ldots, p_{255}\}$ je výstupem modelu a jednotlivé hodnoty p_0 až p_{255} určují pravděpodobnost, že byl použit klíč s hodnotou bajtu podle odpovídajícího indexu prvku. Například hodnota prvku p_32 tak udává s jakou pravděpodobností byl použit klíč s hodnotou 32. [48]

Pro další použití se často vytvoří predikční vektor $\vec{g} = \{g_1, g_2, \ldots, g_{256}\}$, kde g_1 určuje hodnotu klíče s největší pravděpodobností a g_{256} s nejmenší. Vektor $\vec{g}$ tedy určuje pořadí možných hodnot klíče podle pravděpodobnosti jejich použití.

Tato schopnost seřazení možných hodnot klíče podle pravděpodobnosti je velmi dobrou vlastností DL-SCA modelů. Pokud DNN klíč klasifikuje špatně, tedy g_i nebude odpovídat správné hodnotě klíče, útočník ví, jaké hodnoty klíče má vyzkoušet jako další. Bude postupně zkoušet další prvky vektoru $\vec{g}$. To značně ulehčuje uhodnutí klíče hrubou silou.

3.2 Metriky

Jako společné metriky hodnocení modelů se v DL-SCA používají metriky jako Success Rate [42], Guessing Entropy [43] a Key Recovery Difficulty [44]. Tato potřeba speciálních metrik vyvstává z potřeby jednotného hodnocení modelů mezi sebou, i za předpokladu útoků na různé implementace s různými protiopatřeními.

3.2.1 Úspěšnost odhadu

Úspěšnost odhadu (Success Rate) používá zmíněný pravděpodobnostní vektor $\vec{g}$. Nejprve se určí řád obnovení klíče o, podle kterého se úspěšnost odhadu bude počítat. Pokud není specifikován, uvažuje se obvykle první řád. Řád o udává rozsah, ve kterém se uvažuje predikce za správnou. Pro třetí řád tak platí, že pokud je odhad hodnoty použitého klíče v predikčním vektoru $\vec{g}$ na jedné z prvních tří pozic, je odhad považován za úspěšný. Výsledná úspěšnost odhadu je spočítána jako poměr úspěšných odhadů ke všem provedeným odhadům.

3.2.2 Entropie odhadu

Pomocí metriky úspěšnosti odhadu nedokážeme zaznamenat přetížení způsobené dodatečným výpočtem při neúspěšném odhadu. Tedy, pokud se skutečný klíč nenachází na prvním místě, je potřeba vyzkoušet další prvky predikčního vektoru $\vec{g}$, až do nalezení skutečného klíče. To přidává dodatečnou režii nezapočítanou do metriky úspěšnosti odhadu. Tento problém řeší metrika entropie odhadu (guessing entropy). Pro tu je směrodatný průměrný počet klíčů nutných k vyzkoušení. Jinak řečeno, je to průměrná pozice skutečného klíče v predikčním vektoru $\vec{g}$. Pokud je entropie odhadu rovna nule[1], pak model správně klasifikoval daný náměr a hodnota prvního bajtu klíče se nachází na první pozici g_1 predikčního vektoru $\vec{g}$.

[1] V některých programovacích jazycích jako je například Matlab, jsou prvky vektoru indexovány od jedné. V takových případech je nejlepší možná hodnota entropie odhadu jedna.

3.3 Veřejné databáze

V procesu profilujících útoku postranním kanálem se nachází spoustu kritických částí jako je získání referenčního zařízení, získání co nejkvalitnějšího náměru, získání dostatečného počtu náměru, zaměření náměru na úsek ovlivněný klíčem a předzpracování náměru a extrahování důležitých bodů. Pro eliminaci, alespoň některých těchto problémů, vznikly ve vědecké komunitě veřejně dostupné databáze. Tak se výzkumníci mohou soustředit čistě na tvorbu modelu. Další výhodou je společná úroveň pro porovnávání modelů mezi sebou. Dá se tak objektivně rozhodnout o kvalitě modelu. Kompletní přehled veřejných databází, jak pro symetrické, tak asymetrické algoritmy předkládá [45]. Popis tří hlavních databází pro algoritmus AES popisuje [46].

3.3.1 ASCAD

Nejznámější a nejpoužívanější je databáze **ASCAD** (Advanced Side-Channel Analysis Dataset) [41]. Databáze se stala standardem v oblasti DL-SCA. Používá HDF5 formát. Snaží se napodobit roli Modified National Institute of Standards and Technology (MNIST) datasetu v oboru DL-SCA. MNIST dataset obsahuje velké množství ručně psaných číslic a je hojně využívaný pro porovnávání modelů v oblasti klasifikace obrazu [47]. Náměry jsou prováděny na ATMega8515 čipu se softwarovou implementací AES-128. Každý náměr má k sobě přiřazená i metadata (otevřený text, klíč, šifrový text, maska a desynchronizace). Z původních náměrů jsou odvozeny další dva datasety (*50desync* a *100desync*). Tyto náměry pak představují desynchronizaci a útoky tak lépe reprezentují realitu a představují větší výzvu. Má dvě varianty a sice **ASCADf** s fixním klíčem a **ASCADv1** s náhodnými klíči. Varianta ASCADf obsahuje 50 000 náměrů pro trénování a 10 000 náměrů pro útok. Každý náměr měl původně 100 000 vzorků. Autoři databáze však vybrali pouze 700 vzorků zachycujících důležité body v náměru pro útok na třetí bajt klíče. Varianta ASCADv1 má pak 200 000 náměrů pro trénovaní a 100 000 pro útok. Každý náměr v ASCADv1 má 1 400 vybraných vzorků z původních 250 000, pro útok na třetí bajt klíče. U ASCADf jsou použity náhodné otevřené texty, ale stejný klíč. U ASCADv1 jsou náhodné jak otevřené texty, tak klíče. ASCAD má i novou verzi **ASCADv2** [48]. Oproti první verzi, tato obsahuje značně větší výzvu při útoku.

3.3.2 AES_HD

AES_HD [49] dataset je nezabezpečený (implementace bez protiopatření). Náměry jsou měřeny na Xilinx Virtex-5 FGPA (hardwarová implementace AES-128). Z důvodů HW implementace náměry obsahují větší šum oproti softwarovým implementacím. Obsahuje 100 000 náměrů každý s 1250 hodnotami. Rozdělení na profilující

část a útočnou část je ponechána na uživateli. Existují i rozšíření čítající až 500 000 náměrů [50, 51].

3.3.3 AES_HD_MM

AES_HD_MM [52] je dataset s maskovanou verzí AES_HD. Náměry jsou měřené na desce SASEBO-GII FPGA s HW implementací AES-128. Databáze obsahuje 5 600 000 náměrů, každý s 3125 vzorky. Databáze není tak známá, a tudíž není tolik používaná.

3.3.4 AES_RD

AES_RD [53] je časově náhodně zpožděný (random delay) dataset. Časové zpoždění je jedním z protiopatření. V současné době však DL-SCA modely, obzvláště pak ty konvoluční, naleznou klíče i přes náhodné časové zpoždění. Proto není dataset v posledních letech už tolik používán.

3.3.5 DPA Contest

DPA Constest byla soutěž, o co nejúspěšnější útok, pomocí postranních kanálů na náměrech zveřejněných organizátorem. Tato soutěž byla během několika let opakována a v rámci každého kola byla zveřejněná nová databáze náměrů. I přesto, že soutěž již skončila, datasety jsou stále dostupné. Databáze náměrů **DPAv2** [54] je již málo používaná nezabezpečená databáze. Databáze **DPAv4** [55] sloužila jako standard pro jednodušší metody SCA a DL-SCA. Náměry byly měřené na chytré kartě založené na čipu Atmel ATMega-163, na kterém běžel maskovaný algoritmus AES-256. V databázi se nachází 30 000 náměrů [56]. **DPAv4.2** [57] opravuje nalezené chyby v DPAv4, ale ani tato databáze není příliš používána.

3.3.6 CHES_CTF

CHES_CTF [58] je dataset, který je oproti ASCAD o něco těžší prolomit. Proto by mohl být zajímavou výzvou. Bohužel, nedostatečná podpora od tvůrců a pouhých 10 000 náměrů na profilující fázi, vytváří problémy s ověřováním výsledků, a tak se ani tento dataset příliš často nepoužívá.

3.3.7 Portability

Portability dataset [59] byl vytvořen pro potřeby zkoumání přenositelnosti AES algoritmu pro potřeby profilování [60]. Obsahuje náměry softwarové implementace

bez protiopatření. To z něj dělá dataset nepoužitelný pro porovnávání nejlepších DL-SCA metod, neboť pro ně, by tento dataset nebyl výzvou.

3.4 Současný stav problematiky

U zrodu DL-SCA s použitím modelů MLP stál pan Martinásek s literaturou [61, 62, 63], kde porovnává útoky pomocí MLP modelu s klasickými SCA metodami. Použití modelů CNN, pak bylo představeno v [64] a [65]. Významným milníkem bylo představení databáze ASCAD [41], která se stala standardem a společným měřítkem, kde je možno porovnat jednotlivé modely. Další základní kámen byl položen v [66], kde byla představena metodologie vytváření minimálních modelů s výrazně lepšími výsledky, než dosavadní modely na veřejných datasetech. Krátce po [66] přišel [67], který metodologii upravil a zajistil tak snížení počtu vnitřních parametrů za udržení stejných výsledků klasifikace. V článku [67] se vysvětluje časté nepochopení problematiky, a tak se článek stává velkým přínosem do oboru DL-SCA. Problém přenesení útoku z veřejných datasetů, kde je zařízení, na kterém se model trénuje stejné jako zařízení, na které se útočí, do reálného prostředí, kde taková možnost často chybí, musí trénování probíhat na odlišném zařízení (cross-device) studuje [68] a [60]. Modely trénují na databázích pocházejících z různých zařízení a tím zajišťují větší generalizaci.

Zajímavou moderní metodou v oboru DL-SCA je **AutoSCA** [69], který automatizuje hledání nejlepších hyper-parametrů pomocí Bayesovské optimalizace (Bayesian Optimisation). Tato metoda umožňuje nastavení modelu tak, aby dosahoval dobrých výsledků nezávisle na datasetu a typu modelu. Další zajímavou prací je [70], která upravuje původní metodiku pro tvorbu NN pomocí zpětnovazebního učení (Reinforcement Learning) [71], pro použití v SCA.

Použití reziduální sítě (Residual Networks, ResNets) popisují [72, 73] a [74] s poměrně slušnými výsledky. Nicméně výsledky z [72] jsou těžké reprodukovatelné vzhledem k nejasnému popisu konstrukce modelu a použití proprietálního datasetu. Práce [73] je první, jež používá novou a bezpečnější databázi **ASCADv2**. Pro obtížnost realizace útoku na dataset ASCADv2 (dosud nebyl realizován úspěšný útok bez dopomocné znalosti vnitřních stavů maskování) autoři dávají modelu část známého vnitřního stavu maskování ve fázi trénování. Pro jedinečnost použití ASCADv2 je těžké porovnat výsledky s ostatními modely. V neposlední řadě práce [74], která staví na práci [75], upravuje reziduální bloky pro minimalizaci množství potřebného nastavování hyper-parametrů. Výsledky této práce ukazují, že použití reziduálních sítí je vhodné v případech, kdy máme k dispozici velký počet náměrů pro trénování. Literatura popisující současný stav problematiky je například [44, 45, 46].

4 Útok SITM

V rámci diplomové práce se naskytla možnost účasti na projektu zabývajícím se vylepšením útoku SITM (See-In-The-Middle) [76], pomocí metod hlubokého učení. Tento útok se soustředí na vnitřní rundy symetrických šifer a s dopomocí postranních kanálů se je pokouší prolomit. Tato kapitola se bude zabývat teoretickým konceptem útoku SITM, vysvětlením jeho podstaty, jednotlivých kroků a jejich aplikaci na AES. Návrh a realizace vylepšení tohoto útoku metodami hlubokého učení, pak bude představeno v kapitole 7.

4.1 Úvod do SITM útoku

Koncept útoku SITM byl představen na konferenci CHES (Conference on Cryptographic Hardware and Embedded Systems) roku 2020 [76]. Dva roky na to vyšel článek s návrhem vylepšení SITM útoku pro použití na AES-256 s maskováním [77].

Doposud byly používány protiopatření proti SCA převážně na první a poslední rundě [78]. To byla odpověď na trend útoků zaměřujících se na odvozování tajné informace pomocí postranního kanálu z první či poslední rundy. Minimum útoků útočilo na vnitřní rundy. Přidávání maskování i na vnitřní rundy přidává nutnost dodatečného výkon [78, 79]. Proto je použití maskování pouze na první a poslední rundu dobrým kompromisem.

Tohoto faktu využívá útok SITM, který útočí na vnitřní rundy blokových šifer. Tím se stává efektivním oproti maskování, které skrývá pouze první a poslední rundy. SITM útok využívá kombinace diferenciální kryptoanalýzy a postranních kanálů. To mu umožňuje fungovat i v nepříznivých podmínkách s nízkým poměrem signálu k šumu. Tyto výhody útok staví na poměrně zajímavé příčky existujících útoků a o útok se začíná zajímat více a více lidí.

V současnosti se objevila metoda SCADPA (Side-Channel Assisted Differential Plaintext Attack), která využívá rozdíly v procesu šifrování dvou specificky zvolených otevřených textů a postranní kanály k prolomení blokové šifry založené na permutaci bitů jako jsou PRESENT a GIFT [80, 81]. Při této metodě se počáteční rozdíl určí volbou dvou otevřených textů, které jsou jeden po druhém vloženy do procesu šifrování. Následně útočník analyzuje, jak se dané rozdíly na vstupu propagují do výstupu z prvního S-boxu. K tomuto účelu slouží právě postranní kanál. Znalost propagace rozdílů, pak umožňuje jednoduché zpětné odvození tajného klíče. Tato jednoduchost však pramení z jednoduchosti podstaty bitově permutačních šifer. Proto tato metoda sama o sobě není použitelná pro složitější substitučně permutační šifry se silně difúzními funkcemi jako je funkce `MixColumns` v šifře AES.

Autoři SITM útoku na této SCADPA metodě založili svůj útok, kde metodu rozšířili, aby byla použitelná, nejen pro blokové šifry založené na permutaci bitů, ale obecně na blokové šifry založené na substitučně permutační síti (Substitution-permutation network – SPN). Jejich výsledná metoda, jak již bylo zmíněno, umožňuje oproti původní SCADPA metodě útočit i na funkce hlouběji zanořené v procesu šifrování. [76]

4.2 Princip útoku

Útok SITM využívá diferenciální kryptoanalýzu a s dopomocí postranního kanálu je schopný cílit na vnitřní rundy SPN šifry. Diferenciální kryptoanalýza v tomto případě zkoumá propagaci vstupního rozdílu mezi dvěma otevřenými texty v procesu šifrování. Vstupním rozdílem je myšlen rozdíl mezi vstupními otevřenými texty, resp. vstupními bloky, před začátkem šifrování. Tento vstupní rozdíl se označuje δ. Počáteční rozdíl se v průběhu procesu, díky difúzi, rozšiřuje do celého bloku. Při podrobném prozkoumání charakteru šíření rozdílů, lze určit počáteční rozpoložení bajtů, které jsou a nebo naopak nejsou rozdílné mezi dvěma otevřenými texty. To znamená, že v počátečním bloku (před samotným šifrováním) se otevřené texty liší pouze ve specifických bajtech. Rozložení těchto tzv. aktivních bajtů[1] je pro každý typ šifry individuální a závisí na předchozí analýze.

Počáteční rozložení bajtů je voleno tak, aby aktivní bajty konvergovaly k sobě v následujících rundách. Konvergence však není závislá pouze na rozpoložení aktivních bajtů, ale i na jednotlivých počátečních rozdílech aktivních bajtů. Konvergence se tak u dvou náhodných otevřených textů lišících se ve specifických bajtech objevuje pouze s určitou pravděpodobností. Při reálném útoku se hodnoty počátečních otevřených textů volí náhodně, dokud se konvergence neobjeví.

Ve chvíli, kdy je nalezen vhodný pár otevřených textů, útočník může najít všechny možné hodnoty rozdílů aktivních bajtů a pro každý z nich zpětně dopočítat jaké hodnoty klíče mohly být použity. Tak lze výrazně redukovat prostor klíčů. V tomto redukovaném prostoru pak lze použitý klíče nalézt hrubou silou. Rozdíl mezi bajty po konvergenci se nazývá výstupním rozdílem a značí se Δ. [76]

Jednotlivé kroky obecného SITM útoku lze shrnout následovně:

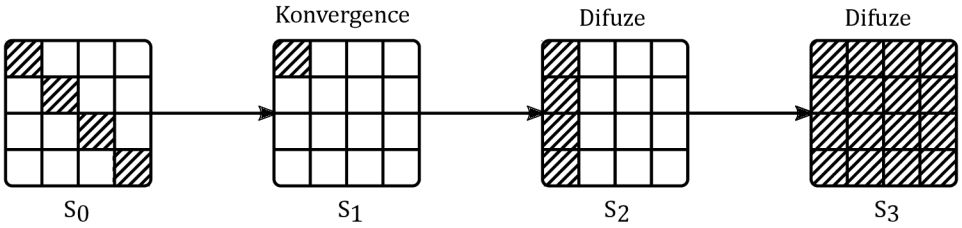
1. Zanalyzovat propagaci aktivních bajtů a určit počáteční rozložení bajtů,
2. najít konvergující pár otevřených textů s počátečním rozložením aktivních bajtů,

[1] Aktivním bajtem se označuje bajt, jehož hodnota je rozdílná oproti bajtu na stejné pozici v kontrolním bloku, resp. mají nenulový rozdíl.

3. provést odhad možných výstupních rozdílů Δ a provést zpětné odvození možných klíčů,
4. opakovat kroky 1-3 dokud není odhalen celý klíč.

4.3 Aplikace na AES

Tato kapitola popisuje aplikaci obecných kroků SITM útoku na blokovou šifru AES. Útok je možný na délky klíčů 128, 192 i 256 [76]. Proces se však v jednotlivých krocích lehce liší. V rámci této práce bude uvažována varianta s délkou klíče 128 bitů. Následující popis jednotlivých kroků a ukázkový příklad byl vypracován na základě návodu [82] poskytnutého autory a na původním SITM článku [76].



Obr. 4.1: Propagace počátečních aktivních bajtů.

4.3.1 Analýza propagace rozdílů

Po analýze propagace rozdílu skrze jednotlivé rundy, autoři přichází s navržením počátečního rozložení aktivních bajtů. Toto počáteční rozložení tvoří hlavní diagonálu, jak lze vidět na obrázku 4.1. Konvergence nastává po provedení první rundy, tedy po operacích `AddRoundKey`, `SubBytes`, `ShiftRows` a `MixColumns`. Konvergence je závislá na jednotlivých vstupních rozdílech bajtů v diagonále a objevuje se s pravděpodobností 2^{-22}. Konvergence může proběhnout do bajtů s_0, s_1, s_2 nebo s_3, při následujícím označení jednotlivých bajtů v bloku:

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}.$$

Do druhé rundy, difúze způsobená funkcí `MixColumns` zajistí, že se jeden aktivní bajt z první rundy rozšíří do celého sloupce. Do jakého sloupce se rozšíří je závislé na tom, jaký je aktivní bajt v rundě první. Pokud jednotlivé bloky označíme S_i, kde i označuje rundu, ve které se daný blok nachází, tedy S_0 označuje vstupní blok

před první rundou, S_1 pak ten samý blok, ale po provedení první rundy šifrování, S_2 po provedení druhé rundy atd. Závislost aktivních bajtů v první a druhé rundě lze popsat následovně:

$$s_0 \text{ aktivní v } S_1 \longleftrightarrow s_0, s_1, s_2, s_3 \text{ aktivní v } S_2,$$
$$s_1 \text{ aktivní v } S_1 \longleftrightarrow s_{12}, s_{13}, s_{14}, s_{15} \text{ aktivní v } S_2,$$
$$s_2 \text{ aktivní v } S_1 \longleftrightarrow s_8, s_9, s_{10}, s_{11} \text{ aktivní v } S_2,$$
$$s_3 \text{ aktivní v } S_1 \longleftrightarrow s_4, s_5, s_6, s_7 \text{ aktivní v } S_2.$$

Ve třetí rundě je již celý blok aktivní vždy, bez ohledu na to, jaké bajty byli aktivní v předchozích rundách. Takto vypadá analýza propagace aktivních bajtů a výsledkem je určení počátečního rozložení aktivních bajtů po diagonále, tak aby v první rundě konvergovaly do jednoho aktivního bajtu. S touto znalostí je možno se posunout k druhému kroku SITM útoku a sice nalezení konvergujícího páru vstupních bloků. [76]

4.3.2 Generování vstupních bloků

Druhým krokem je generování vstupních bloků podle určeného počátečního rozložení aktivních bajtů a hledání takového páru vstupních bloků, které konvergují do jednoho ze čtyř bajtů v S_1. Podle počáteční analýzy víme, že počáteční rozložení aktivních bajtů je po diagonále. Generování počátečních bloků probíhá tak, že jsou všechny bajty v bloku zafixovány, mimo bajtů na diagonále, tzn. kromě bajtů s_0, s_5, s_{10} a s_{15}. Tyto bajty jsou náhodně generovány pro každý nový blok. Takto se postupuje, dokud není nenalezna dvojice bloků splňující podmínku konvergence. Matematicky však nelze rozpoznat, kdy taková situace nastala, neboť tajný klíč nám není znám. V tuto chvíli přichází na řadu postranní kanál. Pozorováním proudového či elektromagnetického postranního kanálu lze zjistit, zda konvergence nastala a pokud ano, tak v jakém bajtu. Znalost v jakém bajtu tato konvergence nastala je klíčová pro další krok, a to zpětného částečného odvození klíče.

Tato část, je také ústředním bodem příspěvku této práce pro útok SITM, neboť právě zde se určuje, v jakém ze čtyř bajtů nastala změna, resp. jaký bajt je aktivní v první rundě. To je problém klasifikace, ve které hluboké učení vyniká. Přínosem této práce je nasazení hlubokého učení do této části klasifikace náměrů do čtyř tříd podle aktivních bajtů po první rundě. Autoři třídění náměrů prováděli ručně, podle oka, což přinášelo značné nároky na čas a lidskou sílu a v určitých případech nebyla klasifikace ani možná. Proto je zapojení hlubokého učení do tohoto procesu značným zlepšení. Tomuto procesu se bude podrobněji věnovat kapitola 7. [76]

4.3.3 Částečné odvození klíče

Po nalezení odpovídajícího páru a zjištění její třídy, tzn. zjištění jaký bajt je aktivní po první rundě, se přejde do třetího kroku, neboli do zpětného částečného odvození klíče. Pro tento proces je kritická znalost, jaký bajt byl aktivní v S_1, resp. jaký sloupec byl aktivní v S_2. Ve chvíli, kdy víme jaký bajt byl aktivní v S_1 je možné zkoušet jeho jednotlivé hodnoty Δ (výstupní rozdíly) a pro každou hodnotu zpětně dopočítat možné hodnoty před první rundou po přidání klíče funkcí `AddRoundKey`. Se znalostí otevřeného textu pak lze pomocí funkce XOR dopočítat potenciálně použité hodnoty bajtů klíče na diagonále, tj. bajtů klíče na pozicích s_0, s_5, s_{10} a s_{15}.

Jelikož možných diferencí jednoho bajtu je 256, není obtížné vyzkoušet všechny hodnoty. Díky zpětné propagaci jednoho aktivního bajtu do čtyř bajtů, je možno zjistit dodatečnou informaci o čtyřech hodnotách bajtů klíče i když jsou ve skutečnosti odhadovány rozdíly pouze jednoho. [76, 82]

Tabulka distribuce rozdílů

Při zpětném odvození se používá tabulka distribuce rozdílů (Difference Distribution Table – DDT) pro urychlení výpočtů. Tuto tabulku lze předpočítat dopředu a při samotném útoku v ní pouze vyhledávat. Tabulka popisuje závislost mezi vstupními rozdíly a výstupními rozdíly. Sloupce tabulky reprezentují vstupní rozdíly δ a řádky pak výstupní rozdíly Δ. Pro jednotlivé hodnoty x pak platí:

$$S(x) \oplus S(x \oplus \delta) = \Delta. \tag{4.1}$$

Slovy by se rovnice 4.1 dala popsat tak, že výstupní rozdíl je roven rozdílu[2] výstupů z S-boxu pro obě vstupní hodnoty. Proměnná x vyjadřuje první vstupní hodnotu otevřeného textu a přičtením vstupního rozdílu k ní se dostáváme na vstupní hodnotu druhého otevřeného textu. Rovnice tak vyjadřuje, že pro výpočet výstupního rozdílu po první rundě je nutné jednotlivé hodnoty nejprve substituovat pomocí S-boxu a následně od sebe odečíst.

Rovnici pro určitý vstupní a výstupní rozdíl splňuje buď více hodnot (obvykle dvě) nebo žádná. Příklad tabulky distribuce rozdílu pro šifru PRESENT je možno vidět na obrázku 4.2. Příklad pro AES není uveden, neboť by tabulka byla příliš velká. [82]

Pro výpočet tabulky se pak vytvoří program, který postupně prochází jednotlivé elementy tabulky a zkouší, pro která x rovnice 4.1 platí. Tyto hodnoty x pak ukládá na danou pozici $[\delta, \Delta]$. Příklad kódu realizující tento výpočet tabulky je možno nalézt na GitHub[3] nebo v příloze práce.

[2] Rozdíl dvou hodnot je zde počítán pomocí operace XOR.

[3] `https://gist.github.com/xmatus33/cfeb1d8bd2655bb5707bc238227d8716`

Δ \ δ	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1			9A		36		078F				5E		1C		24BD
2						8E	34		09	5f		1D	67AB	2C	
3	CDEF	46	12					3B		0A			58	79	
4			47		8D				35AC		0B		2F		169E
5		CDEF		0145						2389		67AB			
6		9B	CDEF	37		06	25		18					4A	
7	67AB		03	8C				5D				2E	49	1F	
8						17	AD		6F	4E	2389	0C		5B	
9	0145			9D	BE			2A			7C	3F		68	
A		02	56	BF	9C					7D	1A	48	3E		
B			8B		27	35AC		169E			4F		0D		
C		8a		26	0145	9f	bc		7e					3d	
D	2389	57			AF			4C		1B	6D			0E	
E		13		AE					24BD	6C		59			078F
F						24BD	169E	078F							35AC

Obr. 4.2: Ukázka tabulky distribuce rozdílů pro šifru PRESENT.

Tabulka možných výstupních rozdílů

Pomocí předpočítané tabulky distribuce rozdílů se pak při útoku vytváří tabulka možných výstupních rozdílů. Jelikož, některé elementy v tabulce chybí, resp. jsou prázdné, není možných výstupních rozdílů 256, ale značně méně (typicky v rozmezí 10 až 20).

Pro nalezení možných výstupních rozdílů Δ jsou postupně zkoušeny všechny možné Δ_i (v případě AES-128 je to 255 možností[4]) a hledat která Δ_i mají neprázdné prvky na specifických místech v tabulce. Pro každý možný výstupní rozdíl Δ_i je potřeba zkontrolovat 4 prvky. Specifická místa v tabulce jsou určena podle vstupních rozdílů na diagonále a jistým offsetem způsobeným funkcí `MixColumns`.

Funkce `MixColumns` je realizována násobením matic, a sice vynásobením bloku dat s konstantní maticí. Při zpětném odvození je potřeba realizovat funkci inverzní k funkci `MixColumns`, tedy funkci `InvMixColumns`. Ta je také realizována násobením matic a však maticí inverzní. Ta má následující podobu:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}.$$

Jednotlivé offsety jsou určeny sloupcem této matice. Sloupec je vybrán podle toho, jaký bajt byl původně aktivní v S_1. Toto je ona kritická část, kde je nutné znát pozici kam diagonála konvergovala po první rundě. Výběr offsetů v závislosti na aktivním bajtu po první rundě je následující:

[4]Výstupní rozdíl nemůže být roven 0, proto není možností 256, ale pouze 255.

$$
\begin{aligned}
s_0 \text{ aktivní v } S_1 &\longrightarrow \text{offsets} = (\texttt{0E}, \texttt{09}, \texttt{0D}, \texttt{0B}),\\
s_1 \text{ aktivní v } S_1 &\longrightarrow \text{offsets} = (\texttt{09}, \texttt{0D}, \texttt{0B}, \texttt{0E}),\\
s_2 \text{ aktivní v } S_1 &\longrightarrow \text{offsets} = (\texttt{0D}, \texttt{0B}, \texttt{0E}, \texttt{09}),\\
s_3 \text{ aktivní v } S_1 &\longrightarrow \text{offsets} = (\texttt{0B}, \texttt{0E}, \texttt{09}, \texttt{0D}).
\end{aligned}
$$

Jednotlivé prvky vektoru offsets určují offset řádku, kde je hledán neprázdný element tabulky. Sloupec je pak určen vstupními rozdíly bajtů na diagonále.

Uveďme si příklad. Mějme následující pár otevřených textů:

`4C3C3F54C7AAD34E607110C753C5E990`, `033C3F54C725D34E607131C753C5E90F`

můžeme si povšimnout, že se otevřené texty liší skutečně pouze v bajtech diagonály, tedy bajtech s_0, s_5, s_{10} a s_{15}. Jejich vstupní rozdíly pak jsou:

$$
\begin{aligned}
\delta_1 &= \texttt{4C} \oplus \texttt{03} = \texttt{4F},\\
\delta_2 &= \texttt{AA} \oplus \texttt{25} = \texttt{8F},\\
\delta_3 &= \texttt{10} \oplus \texttt{31} = \texttt{21},\\
\delta_4 &= \texttt{90} \oplus \texttt{0F} = \texttt{9F}.
\end{aligned}
$$

Následně postranním kanálem zjistíme, že tento pár po první rundě skutečně konverguje, a to do bajtu s_0. Tím víme, že použijeme vektor offsetů roven `(0E, 09, 0D, 0B)`. Nyní máme vše pro nalezení možných výstupních rozdílů Δ. Začneme postupně zkoušet jednotlivé možnosti a kontrolovat podmínku existence elementů na specifických místech v tabulce distribuce rozdílu. Začneme s $\Delta_i = 1$ neboť výstupní rozdíl nemůže být roven nule. Nejprve se posuneme o řádek pomocí prvního offsetu `0E` a operace multiplikace nad Galoisovým tělesem GF(2^8), se specifickým polynomem pro AES. Tím se posuneme na řádek `0E`. Zde se podíváme do sloupce definovaného prvním vstupním rozdílem δ_1, tedy do sloupce `4F`, kde se nachází prvky `26,69`. Element tabulky tak není prázdný a my nezamítáme, že výstupní rozdíl by mohl být roven jedné. Je však potřeba zkontrolovat další tři elementy abychom mohli skutečně prohlásit, že $\Delta_i = 1$ je skutečně možným výstupním rozdílem.

Následně tedy stejný postup opakujeme ještě třikrát s tím rozdílem, že použijeme druhou, třetí a čtvrtou hodnotu z vektoru offsetu a k nim odpovídající δ_2, δ_3 a δ_4.

Po nalezení všech možných výstupních rozdílů a zapamatování si jednotlivých nenulových elementů z tabulky distribuce rozdílů, můžeme vytvořit tabulku možných výstupních rozdílů. Každý řádek této tabulky představuje možný výstupní rozdíl a každý sloupec jeden ze čtyř diagonálních bajtů. Příklad takové tabulky je vidět na obrázku 4.1.

Pomocí tabulky distribuce rozdílů a zohlednění změn rozdílů aktivních bajtů způsobenou funkcí `MixColumns`, jsme zpětně překonaly funkce `MixColumns` a `SubBytes`[5].

[5]Funkce `ShiftRows` nemění hodnoty diference bajtů a proto na ni nemusíme brát ohled při zpětném odvozování.

Tab. 4.1: Tabulka možných výstupních rozdílů pro uvedený příklad.

Δ	$k_{00} \oplus$ 4C	$k_{11} \oplus$ AA	$k_{22} \oplus$ 10	$k_{33} \oplus$ 90
1A	16,59	65,EA	CF,EE	62,FD
29	96,D9	3E,B1	85,A4	78,E7
42	28,67	58,D7	59,78	16,89
5D	AB,E4	40,CF	81,A0	45,DA
66	03,4C	2C,A3	D8,F9	1D,82
71	AF,E0	78,F7	DE,FF	39,A6
74	82,CD	5D,D2	07,26	4E,D1
95	07,48	43,CC	87,A6	65,FA
9C	97,D8	00,3D,8F,B2	44,65	7F,E0
CC	1D,52	37,B8	93,B2	5F,C0
D7	37,78	63,EC	56,77	3E,A1
E7	3A,75	7B,F4	1B,3A	63,FC
EB	BB,F4	34,BB	CD,EC	54,CB

Zbývá poslední funkce a tou je funkce `AddRoundKey`. Tabulka 4.1 tak ukazuje hodnoty diagonály po operaci `AddRoundKey`, tedy hodnoty klíče sečtené s otevřeným textem. [82]

Tabulka možných klíčů

Poté co jsme získaly tabulku možných výstupních rozdílů a v ní obsažené možné hodnoty na diagonále po operaci `AddRoundKey`, zbývá provést zpětný výpočet i této funkce. To naštěstí není problém, neboť funkce `AddRoundKey` pouze pomocí funkce XOR sečte jeden blok klíče s jedním blokem otevřeného textu. Díky inverzovatelnosti funkce XOR tak stačí na tyto hodnoty opět použít funkci XOR a tentokrát přičíst známou hodnotu otevřeného textu. Tím získáme přímo hodnotu klíče.

To znamená, pro získání tabulky s možnými hodnotami klíče stačí když na každou hodnotu z tabulky možných výstupních rozdílů použijeme funkci XOR a přičteme odpovídající bajt diagonály otevřeného textu[6]. Tím získáme tabulku s možnými kandidáty pro 4 bajty klíče na hlavní diagonále. V průměru tato tabulka obsahuje 2^8 možných klíčů. Příklad takové tabulky lze vidět na obrázku 4.2. [82]

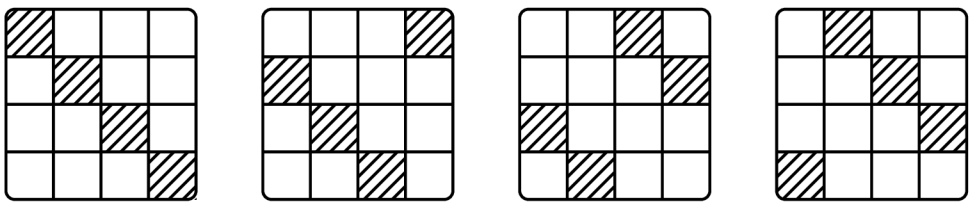
[6]Při AES-128 klíč zabírá přesně jeden blok. Tento blok je v první funkci `AddRoundKey` čistě přičten k bloku otevřeného textu. Proto se nemusíme zabývat rozvojem klíče, ani jeho přetečením do dalších rund.

Tab. 4.2: Tabulka možných klíčů pro uvedený příklad.

Δ	k_{00}	k_{11}	k_{22}	k_{33}
1A	5A,15	CF,40	DF,FE	F2,6D
29	DA,95	94,1B	95,B4	E8,77
42	64,2B	F2,7D	49,68	86,19
5D	E7,A8	EA,65	91,B0	D5,4A
66	4F,00	86,09	C8,E9	8D,12
71	E3,AC	D2,5D	CE,EF	A9,36
74	CE,81	F7,78	17,36	DE,41
95	4B,04	E9,66	97,B6	F5,6A
9C	DB,94	AA,97, 25,18	54,75	EF,70
CC	51,1E	9D,12	83,A2	CF,50
D7	7B,34	C9,46	46,67	AE,31
E7	76,39	D1,5E	0B,2A	F3,6C
EB	F7,B8	9E,11	DD,FC	C4,5B

4.3.4 Opakování pro ostatní diagonály

Pomocí tří předchozích kroků lze redukovat počet možných hodnot klíče pro hlavní diagonálu, tedy pro čtyři bajty klíče. Pro redukci zbylých bajtů klíče, je nutno tyto kroky opakovat pro ostatní tři vedlejší diagonály. Je tak opět nutno zvolit rozložení počátečních aktivních bajtů, vygenerovat nové páry otevřeného textu a zpětně vypočítat možné hodnoty klíče pro všechny odhady výstupních rozdílů. Výsledkem jsou čtyři tabulky možných klíčů pro bajty všechny bajty ve čtyřech diagonálách. Následně lze již klíč prolomit hrubou silou, tedy vyzkoušením všech možností. Rozložení hlavní a vedlejších diagonál lze vidět na obrázku 4.3. [76]



Obr. 4.3: Rozložení hlavní a vedlejších diagonál.

5 Analýza korelačním koeficientem

Analýza korelačním koeficientem je důležitou součástí pro pochopení a připravení náměrů pro modely hlubokého učení. Zároveň, lze s její pomocí přímo útočit na nezabezpečené implementace kryptografických algoritmů. Proto bude tato kapitola zaměřena právě na analýzu pomocí korelačního koeficientu (Correlation Power Analysis – CPA). Bude v ní představena teorie k samotnému korelačnímu koeficientu, jeho využití při útoku a analýze. Dále bude popsán vytvořený nástroj pro provádění této analýzy a útoku. A v neposlední řadě bude nástroj použit k analýze a útoku na veřejně dostupný dataset náměrů DPA Contest v4.2.

5.1 Korelační koeficient

Korelační koeficient, též Pearsonův korelační koeficient, se používá pro nalezení lineární závislosti mezi dvěma náhodnými proměnnými [83]. Pomocí kovariance je definován vztahem:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{\sigma^2(X) \cdot \sigma^2(Y)}}, \tag{5.1}$$

kde:

- $\rho(X,Y)$ je korelace mezi náhodnými proměnnými X a Y,
- $Cov(X,Y)$ je kovariance mezi X a Y,
- $\sigma(X)$ je směrodatná odchylka náhodné proměnné X a
- $\sigma(Y)$ je směrodatná odchylka náhodné proměnné Y.

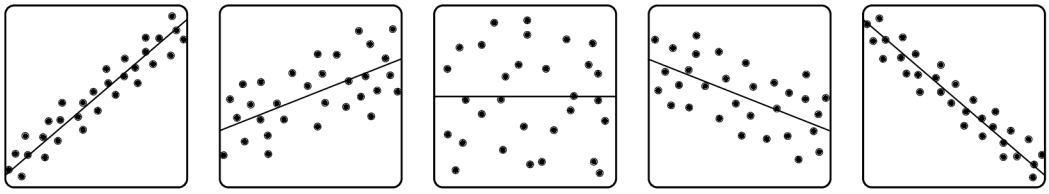
Nechť $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ a $\vec{y} = \{y_1, y_2, \ldots, y_n\}$ jsou vektory s počtem prvků n, kde každý prvek představuje realizaci náhodné proměnné X, resp. Y. Pak lze výpočtem korelačního koeficientu pro $\vec{x}$ a $\vec{y}$ zjistit vzájemnou korelaci těchto vektorů, resp. jejich prvků. Výpočet korelačního koeficientu nad vektory je definován vztahem:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})^2}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{5.2}$$

kde $\bar{x}$ a $\bar{y}$ označuje průměrnou hodnotu prvků vektoru $\vec{x}$, resp. vektoru $\vec{y}$.

Korelační koeficient je bezrozměrnou veličinou z intervalu <-1,1>. Pro $r < 0$ platí, že změna prvků v jednom vektoru je doprovázena opačnou změnou prvků v druhém vektoru (negativní závislost). Pro $r = 0$ platí, že mezi prvky vektoru $\vec{x}$ a vektoru $\vec{y}$ neexistuje žádná statisticky zjistitelná závislost. Naopak pro hodnoty $r > 1$ platí, že změna prvků v jednom vektoru bude doprovázena změnou ve stejném směru (pozitivní závislost). Čím více se r blíží 1 resp. -1, tím silnější korelace mezi

vektory existuje. Grafické znázornění významných hodnot korelačního koeficientu lze vidět na obrázku 5.1.



(a) r = 0.8 (b) r = 0.4 (c) r = 0 (d) r = -0.4 (e) r = -0.8

Obr. 5.1: Vizualizace korelačního koeficientu.

5.2 Využití korelačního koeficientu

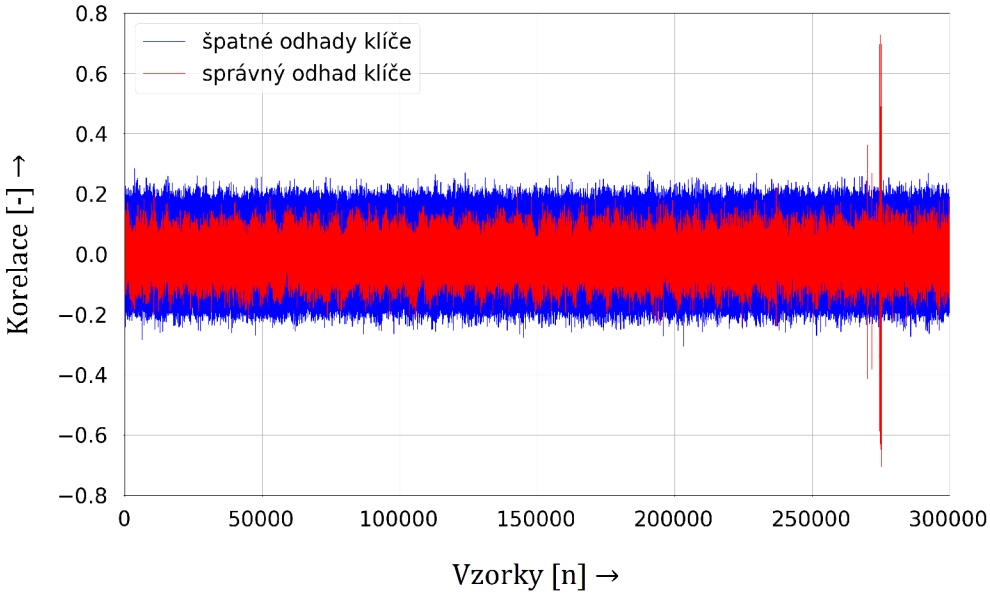
Korelační koeficient, resp. schopnost nalézt závislost mezi dvěma vektory, se využívá pro nalezení závislosti mezi naměřenou veličinou a tajnou informací při útoku/analýze[1] postranním kanálem. Útok spočívá v postupném odhadování hodnoty tajné informace (klíče), výpočtu hypotetické hodnoty měřené veličiny a následné vypočítání korelačního koeficientu mezi hypotetickými hodnotami a skutečně naměřenými hodnotami. Pokud se korelační koeficient bude blížit -1 nebo 1, pak existuje statistická závislost mezi hypotetickými a skutečnými hodnotami měřené veličiny. Z toho lze odvodit, že použitý odhad hodnoty tajné informace pro výpočet hypotetické hodnoty byl správný. Stejná hodnota tajné informace byla pravděpodobně použita i v době, kdy byl náměr pořízen a hodnota tajné informace byla úspěšně odhalena.

Korelační koeficient se počítá pro každou hodnotu náměru. Pro jeden odhad tajné informace tak vznikne jeden vektor $\vec{r}$ o velikosti n, kde n je velikost samotného náměru. Jinak řečeno, výsledný vektor má stejný počet prvků jako náměr samotný a každý prvek výsledného vektoru určuje korelaci v daném bodě náměru. Tato vlastnost umožňuje nejen odhalit tajnou informaci z náměru, ale i nalézt místo resp, čas v náměru, kde se daná hodnota tajné informace vyskytuje.

Pro různé vstupy algoritmu, tedy různé odhadované hodnoty, z nichž je počítána hypotetická hodnota měřené veličiny, se dají lokalizovat různé operace/místa v náměru. Například pro nalezení operace substituce u symetrické šifry AES se použije jako vstupní hodnota výstup S-boxu pro daný otevřený text a konkrétní odhad klíče. Po provedení korelace, bude na místě, kde je substituce S-boxem prováděna výrazně

[1]Útok a analýza jsou v této oblasti do značné míry zaměnitelné, neboť se analýza provádí útokem, resp. při útoku se dozvídáme informace o daném náměru.

vyšší. Graficky znázorněný výsledek korelace lze vidět na obrázku 5.2, kde je pozice provádění substituce prvního bajtu, indikováno značně vyšší mírou korelace. [83, 13]



Obr. 5.2: Průběh korelace pro výstup S-boxu šifry AES.

5.3 Implementace

Možnost analýzy náměru za pomocí korelačního koeficientu je silným nástrojem pro přípravu a zpracování náměrů pro modely hlubokého učení. Proto byl v rámci práce vytvořen program pro provádění této CPA analýzy. Program je implementován v programovacím jazyce Python. Python byl zvolen pro jednoduchost psaní a následnému porozumění kódu a pro jednoduchost provádění případných změn v programu, oproti kompilovaným programům. Současná řešení [84] nebo [85] provádí výpočty korelace ve smyčkách, což je v tak abstraktním a vysokém programovacím jazyce jako je Python značně pomalé řešení. Proto byla vytvořena vlastní implementace pomocí maticového výpočtu a knihovny Numpy [86]. Program byl implementován na prolamování symetrické šifry AES s velikostí klíče 128 bitů. Program je veřejně dostupný prostřednictvím veřejného GitHub repozitáře[2].

[2] `https://github.com/xmatus33/CPA`

5.3.1 Vstup programu a možnosti nastavení

Program je tvořen modulárně, a tedy je dělen do jednotlivých sekcí, které vykonávají specifické činnosti. Specifika činností lze jednoduše změnit na jednom místě pomocí nastavitelných proměnných. Ty mohou specifikovat na jaké bajty klíče se bude program zaměřovat, jaký bude vstup pro korelaci, zda se bude korelace provádět přes celý náměr případně pouze pro jeho podmnožinu pro rychlejší výpočet, pro kolik náměrů bude korelace prováděna, případně po kolika skupinkách se budou náměry přičítat. Tyto a více specifikací mohou být v programu nastaveny. Podrobnější popis jednotlivých specifikací, co dělají, kdy je vhodná jejich úprava a jaké hodnoty je ideální nastavit, bude více vysvětleno v následujících kapitolách.

5.3.2 Maticový výpočet

Výpočet korelace je prováděn dle vzorce 5.2, kde za x je dosazena hypotetická hodnota měřené veličiny (vypočítaný hypotetický model) a za y je dosazena reálně naměřená hodnota (hodnota z náměru). Vzorec sám o sobě slouží pro výpočet pouze jedné hodnoty korelace mezi hypotetickým modelem a reálnou veličinou. Pro výpočet se používá n náměrů, kde $n > 1$. Čím větší je n tím přesnější výsledek bude, viz kapitola 5.4. Pro výpočet celkového průběhu korelace pro jeden odhad klíče je potřeba použít vzorec 5.2 x krát, kde x je počet vzorků (samples) v jednom náměru. Pro celkový odhad jednoho bajtu klíče je potřeba vypočítat tyto průběhy pro všechny možné odhady klíče (256 možností). Útok na jeden bajt tak potřebuje $256 * x$ výpočtů vzorce 5.2. V náměrech, kde se x blíží statisícům až miliónům vzorků[3] je čas potřebný pro takový výpočet značný.

Proto byl vzorec 5.2 převeden na maticový výpočet, tak aby byly vypočteny všechny hodnoty pro všechny odhady klíče najednou. Operace probíhají nad maticemi pomocí knihovny Numpy. Při výpočtu je použito pomocných proměnných pro uchování mezivýpočtů. Tři hlavní pomocné proměnné uchovávají aktuální hodnoty třech sum objevujících se ve vzorci. Tak je umožněno postupné přičítání dodatečných náměrů, tedy postupné navyšování počtu náměrů n použitých pro výpočet korelace. Tato schopnost je výhodnou v okamžiku, kdy je výpočet pro velký počet náměrů n příliš paměťově náročný a program by jinak skončil z důvodů vyčerpání operační paměti výpočetního stroje. Výpočet po menších skupinách tak umožňuje dosahovat přesných výsledků bez omezení velikostí operační paměti.

Výpočet po skupinách umožňuje i sběr doprovázejících dat ohledně průběhu výpočtu, resp. jeho zpřesňování. Tyto data pak mohou být vyobrazena do grafů a mohou tak přinášet dodatečné informace. Jednotlivé grafy a informace z nich získané

[3]Celý proces šifrování jednoho 126 bitového bloku pomocí šifry AES je v DPA Contest v4.2 zaznamenán v náměru s 1 704 402 vzorky

budou představeny dále.

5.3.3 Výstup programu

Korelační matice je vždy počítána pouze pro jeden bajt. V případě specifikace více bajtů se vypočítá korelační matice zvlášť pro každý bajt (pro 16 bajtů bude vypočítáno 16 korelačních matic). Korelační matice má rozměry 256krát počet vzorků v náměru. Každý řádek odpovídá odhadu jedné hodnoty klíče. Výsledná matice obsahuje 256 průběhů (řádků). Každý průběh popisuje korelaci pro daný odhad hodnoty klíče. Pouze jeden z 256 průběhu bude vykazovat velkou korelace v určitém bodě. Výskyt této korelace říká, že odhad klíče v daném průběhu (řádku korelační matice) byl správný. Výsledný odhad hodnoty klíče daného bajtu je určen indexem daného řádku této matice.

Program k odhadu hodnoty klíče vrací jak výstup, tak i dodatečné data o průběhu výpočtu. Tyto data jsou entropie odhadu, dosažené maximální hodnoty pro každý odhad klíče a samotná korelační matice. Entropie odhadu byla již popsána v kapitole 3.2.2. Popisuje kolik úsilí bude ještě útočník muset vynaložit pro prolomení klíče. Z maximálních dosažených hodnot lze odvodit, jak moc je daný výsledek skutečně pravdivý, resp. jakou roli hrála náhoda způsobená šumem. S přibývajícími náměry klesá hodnota šumu a správný odhad přestává být náhodným odhadem, ale cíleným výběrem průběhu s výrazně vyšší hodnotou korelace. Předání samotné korelační matice, pak umožňuje zpětnou kontrolu a vyobrazení průběhů do grafu. Grafy a podrobnější popis jednotlivých hodnot bude představen v následující kapitole.

5.4 Provedení útoku

Tato kapitola se zabývá samotným provedením útoku pomocí korelačního koeficientu na veřejný dataset DPA Constest v4.2 [57]. Pro ukázku bude útok prováděn pouze na jeden bajt klíče, a to první bajt. Útok na zbytek bajtů by probíhal stejným způsobem, pouze by se lišily vstupní hodnoty.

Jako vstup útoku je použit výstup S-boxu, do kterého bude vstupovat odhad klíče a otevřený text. Tento odhad vytvoří hypotetickou hodnotu, pro kterou je počítána korelaci v náměru. Pokud se povede najít vysokou korelaci je pravděpodobné, že zvolený odhad klíče byl správný. Pokud ne, je vyzkoušen jiný odhad. Výstup S-boxu se spočítá dle vztahu:

$$v = S\text{-}box(PT \oplus K_{guess}), \tag{5.3}$$

kde PT je bajt otevřeného textu, K_{guess} je odhad klíče a funkce S-box je dáno substituční tabulkou definovanou institucí NIST [87].

Výstup vzorce 5.3 se použije pro výpočet hypotetického modelu spotřeby pomocí Hammingovy váhy. Mezi tímto hypotetickým modelem a skutečným náměrem už je počítána samotná korelace podle vztahu 5.2.

5.4.1 Popis datasetu

Dataset náměrů DPA Constest v4.2 je dělen na samotné náměry a na index náměrů, který drží metadata k náměrům. Náměry jsou děleny do 24 souborů, kde se v jednom souboru nachází 2500 náměrů. Dataset používá 16 různých klíčů, kde s každým klíčem bylo provedeno 5000 náměrů a každému klíči tak odpovídají dva soubory. Každý náměr se skládá z 4 704 402 vzorků s rozsahem -128 až 127 a zachycuje celý proces šifrování, včetně všech 10 rund šifry AES-128, načítání hodnot z registrů apod. Soubory je možno stáhnout z oficiálních stránek datasetu[4]. Pro ukázku bylo při provádění útoku použito prvního klíče `k00` konkrétně jeho 2500 prvních náměrů (první soubor). [57]

5.4.2 Odmaskování

Implementace AES, na které byly náměry měřeny, používá RSM maskování. Proto pro výpočet korelace nemůže být použito čistého výstupu S-boxu podle vzorce 5.3, ale je potřeba připočítat i toto RSM maskování. RSM maskování bělí hodnotu výstupu S-boxu jednou z 16 před-definovaných hodnot. Těchto 16 hodnot se společně nazývá vektorem masek `M` a jeho hodnoty jsou definovány v následujícím výpisu:

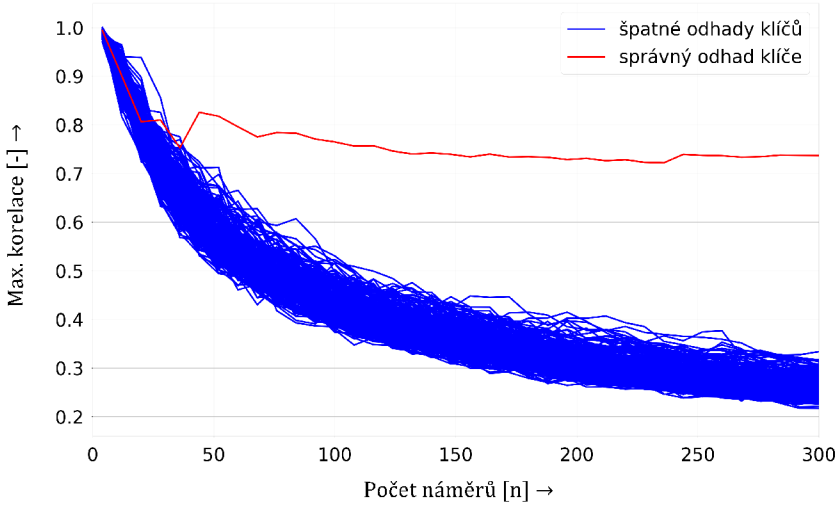
Výpis 5.1: Vektor masek

```
M = (0x03,0x0C,0x35,0x3A,0x50,0x5F,0x66,0x69,
     0x96,0x99,0xA0,0xAF,0xC5,0xCA,0xF3,0xFC)
```

To, která hodnota bude použita určuje tzv. offset. Ten může nabývat hodnot od 0 do 15. Hodnota offsetu plus jedna pak určí index masky z vektoru masek, která bude použita pro samotné bělení. Pokud je například offset roven jedné, pak se jako maska pro bělení výstupu S-boxu použije hodnota 0x35 hexadecimálně, resp. 53 decimálně. Bělení samo o sobě pak probíhá pomocí funkce XOR a to pouhým přičtení vybrané masky `M[offset + 1]` k výstupu S-boxu ze vztahu 5.3.

Pro každý bajt je použit jiný náhodný offset. Použité hodnoty offsetů jsou uloženy v metadatech přiložených k náměrům. Odkud je možno offsety vyčíst a použít je pro započtení RSM maskování do hypotetického modelu. [88]

[4] `https://www.dpacontest.org/v4/42_doc.php`



Obr. 5.3: Vývoj max. hodnot korelace vůči počtu náměrů.

5.4.3 Analýza pomocí korelačního koeficientu

S vypočteným hypotetickým modelem proudového odběru lze přejít k samotné analýze. Pro výpočet korelace je použit program popsaný v kapitole 5.3. Jako výstup jsou získány čtyři hodnoty a to: odhad klíče, entropie odhadu, dosažené maximální hodnoty pro každý odhad klíče a samotnou korelační matici. Detailní popis výstupů je popsaný v kapitole 5.3.3. Pomocí výstupních hodnot se lze dozvědět nejpravděpodobnější hodnotu použitého klíče. Tím vzniká možnost odvodit informace dodatečné.

Pozice

Pomocí analýzy korelačním koeficientem lze zjistit pozici prováděné operace v zasynchronizovaných náměrech. Jinak řečeno lze nalézt pozici, kde se pracuje s hodnotou použitou pro výpočet hypotetického modelu. V našem případě byly použity hodnoty výstupů z S-boxu. Analýzou je tedy možno zjistit, kde se provádí substituce jednotlivých bajtů, resp. kde se provádí operace `SubBytes`.

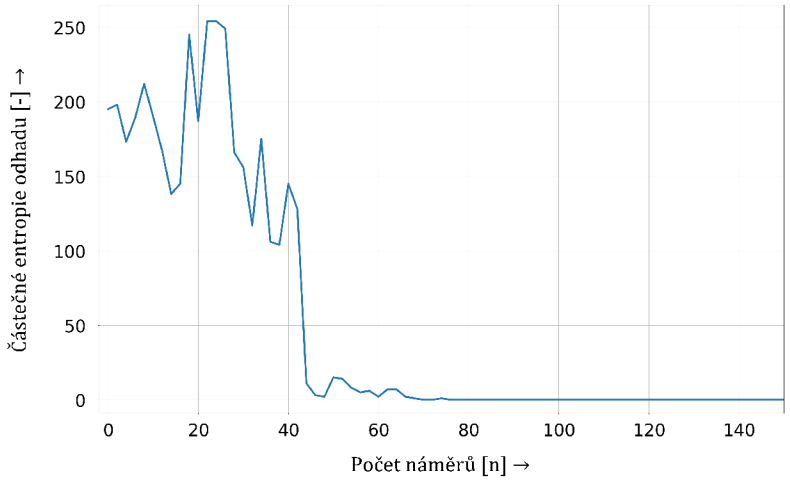
Schopnost lokalizace specifických funkcí je kritická pro útoky pomocí hlubokého učení, neboť lze vybrat specifické vzorky (PoI, Points of Interest) z náměru a pouze ty použít jako vstup dané neuronové sítě. Tím se celý model i jeho trénování, výrazně zjednoduší.

Samotná pozice je dána vzorkem s nejvyšší hodnotou korelace. Tuto pozici lze vyčíst z korelační matice, z řádku správného odhadu klíče. V našem případě je to pozice vzorku 274801. Na obrázku 5.2 můžeme skutečně v tomto místě pozorovat nárůst korelace. Tímto způsobem lze lokalizovat jakoukoli operaci v náměru. Stačí měnit hodnoty vstupující do výpočtu hypotetického modelu.

Počet náměrů

Při útoku je důležitým parametrem počet náměrů potřebných pro útok. Čím méně náměrů útočníkovi stačí k úspěšnému odhadu klíče, tím zranitelnější daná implementace je. Při útoku korelačním koeficientem je při nízkém počtu náměrů korelace velmi vysoká – blížící se jedné – ve všech bodech náměru (velký šum). S rostoucím počtem náměrů se úroveň korelace snižuje až po určitý bod. Kde se začne oddělovat reálná korelace v bodě provádění dané operace od hodnot průměrných (od šumu). Na obrázku 5.3 lze vidět průběh maximálních hodnot korelace pro jednotlivé odhady. Špatné odhady jsou vyznačeny modrou barvou a správný odhad červenou. Je možno si všimnout, že při použití 40 a více náměrů se skutečně korelace správného odhadu výrazně oddělí a lze tak s poměrnou jistotou označit nalezenou hodnotu klíče za skutečně použitou při pořízení náměru.

Počet 40 náměrů je významným bodem i u vývoje částečné entropie odhadu (Partial Guessing Entropy - PGE), kterou lze pozorovat na obrázku 5.4. Oddělení správného odhadu od špatných, se zde projevuje výrazným snížením entropie odhadu. Od tohoto bodu se již správný odhad výrazně odlišuje od ostatních. Je tak možno správně určit hodnotu použitého klíče.



Obr. 5.4: Vývoj PGE vůči počtu náměrů.

6 Útok na ASCAD dataset

Tato kapitola se zabývá nasazením metod hlubokého učení na veřejné datasety. Konkrétně pak na ASCAD dataset. Nejprve budou představeny modely přiložené tvůrci datasetu. Tyto modely budou použité pro porovnání s vlastním navrženým modelem, jež bude popsán v samostatné sekci. Následně bude popsán proces trénování a validace. Na závěr budou všechny natrénované modely porovnány pomocí zvolené metriky.

6.1 Referenční ASCAD modely

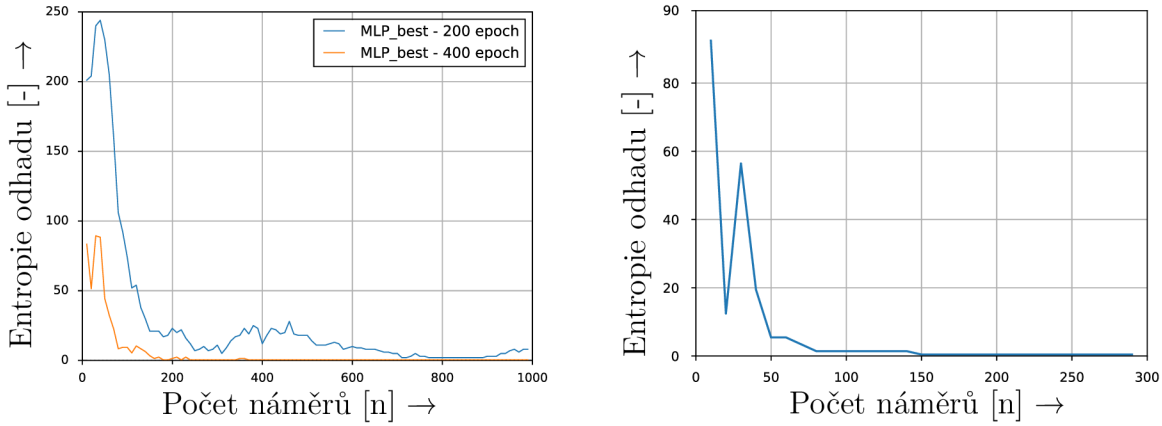
Autoři datasetu přiložili i referenční modely k porovnávání ostatních modelů. Tyto modely jsou MLP_best a CNN_best [41]. Jedná se o nejlepší modely, které autoři představili. Popis daných modelů se nachází v ASCAD GitHub repozitáři [48]. Dataset obsahuje náměry s fixním klíčem a s proměnným klíčem. Náměry s promměným klíčem byly přidány později. Přiložené modely proto pracují s fixním klíčem. To znamená, že pro všechny náměry byl použit stejný klíč.

Perceptronový model

Přiložený MLP_best model je vícevrstvým perceptronovým modelem. Má šest plně propojených vrstev s 200 uzly v každé vrstvě. Tato architektura je nejjednodušší architekturou. V dnešní době se používají komplexnější architektury, jako jsou například konvoluční či rekurentní neuronové sítě. Autoři model trénovali 200 epoch. Tento počet určili jako kompromis mezi časem potřebným pro trénování a přesností modelu. Při počtu 200 epoch model již nedává dobré výsledky, v porovnání s dnešními modely, a tak byl model v rámci této práce dotrénován na 400 epoch. V takovém případě už dosahuje výsledků porovnatelných s CNN. Takto dotrénovaný model bude použit dále pro porovnávání s ostatními.

Konvoluční model

Přiložený CNN_best model je konvolučním modelem. Skládá se ze dvou částí, a to samotné konvoluční části a klasifikační částí. Konvoluční část je složena z více bloků konvoluce a redukce (pooling), které se starají o hledání vzorců v náměru. Klasifikační část se skládá z plně propojených vrstev tak, jak je tomu i u MLP modelu. Tato část slouží ke klasifikaci do potřebných tříd (256 tříd reprezentujících hodnotu klíče). Avšak tím, že jí zde předchází konvoluční část, může být značně jednodušší. Na obrázku 6.1 je možno vidět výsledky obou modelů. Jako metrika byla použita entropie odhadu v závislosti na počtu použitých náměru při trénování.



(a) MLP_best. (b) CNN_best.

Obr. 6.1: Výsledky referenčních modelů z [41].

6.2 Návrh vlastního modelu

V rámci práce byla navržena vlastní architektura NN pro dosažení, co nejlepších výsledků. Model je založen na konvoluční architektuře, neboť dle výsledku z grafu 6.1 dosahuje lepších výsledků než MLP síť. Oproti architektuře CNN_best je však navrhovaná architektura značně méně komplexní. Návrh modelu probíhal dle metodologie [66], kde je snaha o vytvoření co nejjednodušší a nejmenší NN. Tak bude dosahováno krátkých trénovacích časů a síť bude mít menší tendenci k hledání vzorů, které ani v náměru nemusí být. Navrhovaný model má tedy výrazně redukovaný počet parametrů potřebných k natrénování. Byť byl přidán jeden blok konvoluce, byla zmenšena velikost a počet jednotlivých konvolučních filtrů. Díky zmenšení velikosti filtrů, již nedochází ke splývání bodů zájmů (Points of Interest, PoIs), ze kterých se snažíme získat uniklou informaci. Model tak může být menší, ale stále dosahovat stejných či lepších výsledků. Navrhovaný model má šest konvolučních bloků, každý s jednou konvoluční vrstvou a jednou průměrnou redukcí (average pooling). Na konvoluční bloky jsou připojeny dvě plně propojené vrstvy, obě s počtem uzlů 1028. V prvních třech konvolucích je počet filtrů osm v každé vrstvě. V posledních třech konvolucích pak 12. Velikosti filtrů jsou postupně od první konvoluce 1, 3, 3, 6, 6 a 9. Rychlost učení, optimalizátor, aktivační funkce a funkce ztrátová byly ponechány z CNN_best modelu [41]. Velikost dávek je 100. U trénování nebyla pozorována horní hranice počtu epoch, pro kompromis mezi dobou trénování a výsledkem bylo zvoleno 200 epoch. Konkrétní hodnoty hyper-parametrů byly zvoleny experimentálně. Pro trénování bylo použito 45000 náměrů z ASCADv1 datasetu. Pro validaci pak 5000 náměrů a pro útok 10000 náměrů.

6.3 Trénování a validace

Trénování probíhalo na grafické kartě Nvidia GeForce RTX 3080 10 GB. Jako programovací jazyk byl použit Python, a pro práci s NN byla použita knihovna Keras [89]. Jako vývojové prostředí byl zvolen Jupyter LAB [90].

Pro trénování modelů byl vytvořen kód v prostředí Jupyter LAB. Kód umožňuje trénování a validaci výše zmiňovaných modelů. Pro každý model je vytvořena vlastní sekce, kde je architektura modelu definována. V případě potřeby je umožněno rychle a jednoduše provádět jakékoliv změny. Modely mohou být uloženy a zpětně načteny prostřednictvím formátu HDF5. Kód použitý pro trénování a validaci je dostupný na GitHub repozitáři[1] či v příloze práce.

Po vytvoření nenatrénovaneného modelu s požadovanou architekturou pomocí definovaných sekcí je možno model trénovat pomocí funkce `fit()`. Trénování potřebuje pro svůj běh trénovací náměry a jim odpovídající správné hodnoty klíče. Tyto hodnoty jsou předány funkci `fit()` prostřednictvím parametrů.

Pro trénování je kritické nastavení jednotlivých hyper-parametrů, jako je počet epoch či velikost dávek. Tyto parametry jsou definovány pro jednotlivé modely společně s architekturou a jsou předány funkci `fit()` opět pomocí parametru.

Modely útočí na výstup S-boxu v první rundě šifry AES. Hodnoty, na které se cílí tedy nejsou přímo hodnoty klíče. Konkrétní hodnoty klíče se pak dopočítají. Porovnání modelů však bude probíhat nad výstupy z S-boxu.

Jako metrika hodnocení byla zvolena entropie odhadu, neboť je v podstatě standardem při validaci SCA modelů obecně. Oproti jednodušším metrikám jako je čistě přesnost modelu poskytuje více informací o úspěšnosti útoku. Hodnocení modelů probíhá na náměrech, určených pro útok. S těmito náměry se modely doposud nikdy nesetkaly a lze tak na nich ověřit jejich funkčnost. Výsledkem hodnocení daného modelu je pak graf vývoje entropie odhadu v závislosti na počtu náměrů použitých při útoku.

6.4 Porovnání modelů

V této části je porovnán navržený model s referenčními modely popsanými v sekci 6.1 – MLP_best a CNN_best. Tabulka 6.1 porovnává parametry jednotlivých modelů. Je možno vidět značné zjednodušení navržené konvoluční sítě oproti CNN_best ale mnohonásobně komplexnější než MLP_best model. Navzdory tomu čas vyžadovaný pro natrénování navržené sítě je poloviční.

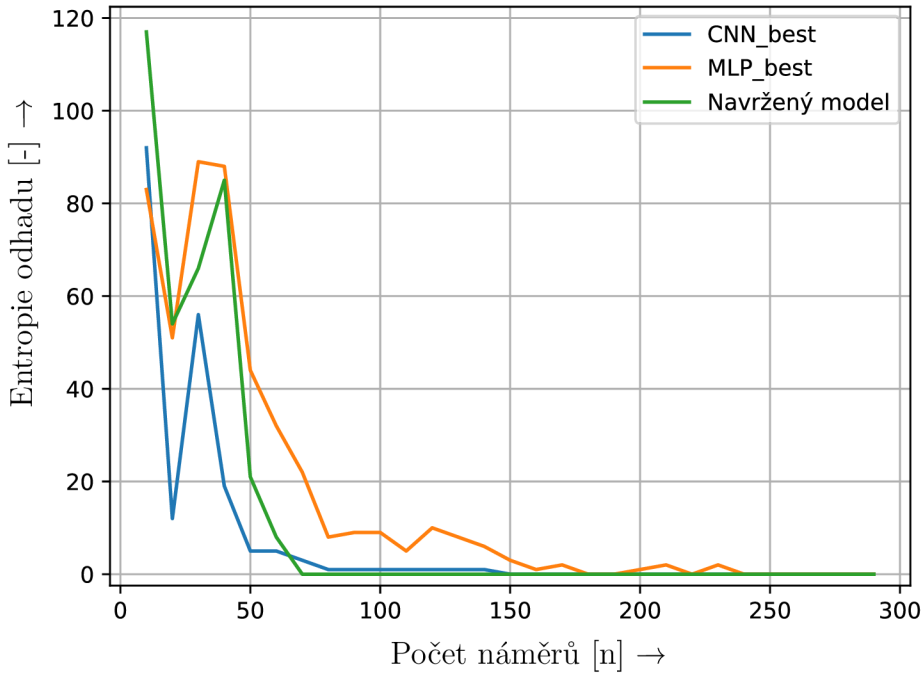
Pro porovnání modelů byla vybrána metrika entropie odhadu. Výsledky modelů zanesené do grafu je možné porovnat na 6.2. Do 50. náměru si navržený model

[1] `https://gist.github.com/xmatus33/77cf44ce80282fe960a54118e44c4144`

Tab. 6.1: Parametry modelů.

Název modelu	Počet parametrů	Čas trénování	Velikost modelu
MLP_best	352 456	414 s	2.9 kB
CNN_best	66 652 544	1045 s	509 MB
Navržený model	5 639 984	265 s	44 MB

nevede o nic lépe než MLP_best. Následně však rapidně konverguje k nule. Od 70. náměru je již entropie odhadu nula a tedy model správně predikuje výstupu z S-boxu. CNN_best dosahuje nuly po 150 náměrech a MLP_best po 180. V tomto ohledu je navržený model nejlepší ze tří porovnaných. Můžeme si také všimnout značené stability modelu po dosažení nuly oproti modelu MLP_best. Při použití navrženého modelu útočníkovi pro úspěšný útok stačí získat 70 náměru z cílového zařízení.



Obr. 6.2: Porovnání navrženého modelu s modely MLP_best a CNN_best.

7 SITM útok a hluboké učení

Průběh a jednotlivé kroky standardního útoku SITM byly popsány v kapitole 4. Tato kapitola do schématu útoku SITM přináší zabudování hlubokého učení pro účely klasifikace náměrů do čtyř tříd. Třídy jsou určeny podle pozice aktivního bajtu v první, resp. druhé a třetí rundě. Následující pravidla popisují rozdělení párů otevřených textů do jednotlivých tříd:

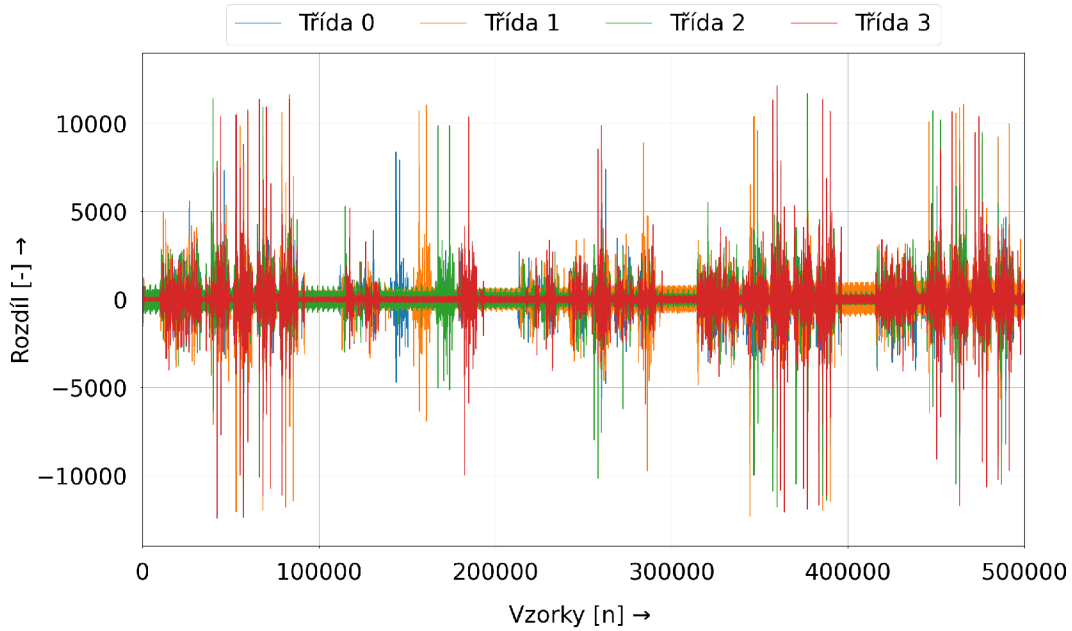
$$
\begin{aligned}
s_0 \text{ aktivní v } S_1 &\longrightarrow \text{náměr z třídy 0},\\
s_1 \text{ aktivní v } S_1 &\longrightarrow \text{náměr z třídy 3},\\
s_2 \text{ aktivní v } S_1 &\longrightarrow \text{náměr z třídy 2},\\
s_3 \text{ aktivní v } S_1 &\longrightarrow \text{náměr z třídy 1}.
\end{aligned}
$$

Správná klasifikace náměrů je kritickou částí pro úspěšné provedení útoku SITM. Krok zpětného odvození klíče potřebuje tuto třídu znát, neboť bez ní nedokáže správně odvodit možné hodnoty klíče. Autoři prováděli klasifikaci náměrů ruční inspekcí rozdílového náměru. Rozdílovým náměrem se rozumí takový náměr, který vznikne odečtením dvou náměrů v každém době vzorkování. Zmíněné dva náměry jsou získány měřením čipu při šifrování s oběma náměry z páru otevřeného textu vygenerovaného v druhém kroku útoku SITM.

Autoři při pořizování jednoho náměru použili průměrování z 1024 měření. To znamená, že se čip při práci s daným otevřeným textem a tajným klíčem naměří 1024krát, jednotlivé hodnoty se zprůměrují, a tak vzniká jeden náměr. S takto velkým průměrováním jsou závislosti v náměru patrné pouhým okem. To je také způsob, jakým autoři náměry klasifikovali. Rozdílový náměr s průměrováním 1024 lze vidět na obrázku 7.1, kde jsou jednotlivé třídy barevně odděleny a skutečně lze pozorovat jejich vzájemný rozestup.

Průměrování 1024 náměrů však v praxi není praktické. Proto by bylo vhodné průměrování snížit, ideálně úplně, tak aby stačilo pouze jedno měření. Při snižování průměrování však dochází k většímu zkreslení a náměry přestávají být tak čitelné. Pouhým okem je analýza nemožná. Zde přichází na řadu hluboké učení a jeho schopnost klasifikace. Hluboké učení tuto klasifikaci zvládne rychleji a přesněji, než je toho schopen člověk. Ušetří se tak čas a zároveň získáme možnost značného snížení průměrování až tam, kde už to vypadá, že žádný vzorec není. Nasazením hlubokého učení, konkrétně konvolučních sítí, jsme schopní se z průměrování 1024 dostat na průměrování čtyřech měření a stále si zachovat výbornou přesnost. Celý proces SITM útoku s nasazeným modelem hlubokého učení pro klasifikaci realizuje vytvořená aplikace dostupná na GitHub repozitáři[1] nebo v elektronické příloze práce.

[1] `https://github.com/xmatus33/SITM_using_deep_learning`



Obr. 7.1: Rozdílový náměr s průměrováním 1024, zachycující 5 rund šifrování AES-128.

Aplikace je napsána v programovacím jazyce Python. Jejím vstupem je pár otevřených textů a odkaz na uložený rozdílový náměr, ze kterého natrénovaná neuronová síť uložená ve stejném adresáři klasifikuje třídu. Ta je následně použita při výpočtu SITM útoku. Aplikace se obsluhuje pomocí příkazovové řádky a jednotlivé vstupní hodnoty se zadávají pomocí parametrů. Příklad užití je přiložen jak na GitHub tak v příloze.

7.1 Návrh architektury

Hlavním úkolem u vytváření modelů hlubokého učení je navržení architektury tak, aby nebyla příliš složitá, trénování netrvalo příliš dlouho a nedocházelo k podtrénování a naopak, aby nebyla příliš jednoduchá a nedocházelo k přetrénování.

V rámci této práce byly navrženy a vyzkoušený dva typy architektur, a to vícevrstvá perceptronová síť (MLP) a síť konvoluční (CNN). U obou architektur byla prozkoumána jejich šířka i hloubka tak, aby dosahovaly co nejlepších výsledků. Konkrétní hodnoty byly doladěny pomocí Keras Tuner[2]. Tuner umožňuje nastavit proměnné a jejich rozsah. Tuner sám nalezne, která kombinace je nejlepší.

Náměry pro trénování byly získány pomocí měřící stanice sestavené pro tento účel v laboratořích univerzity VUT v Brně. Měřící stanice byla založena na desce SAKURA-G a přídavné desce SAKURA-W, pro čtení chytrých karet. Měření bylo

[2] `https://keras.io/keras_tuner/`

Tab. 7.1: Rozdělení datasetu.

Data	Počet náměrů	Procentuální rozdělení
Trénovací	3600	60 %
Validační	720	20 %
Testovací	720	20 %

prováděno pomocí Agilent MSO3103B osciloskopu se vzorkovací frekvencí 1 GHz. Měřila se proudová spotřeba softwarové AES implementace. Pro tento účel byl použit programovací jazyk C. Každý náměr obsahoval pět rund šifrování AES a byl průměrován ze čtyřech měření. Dataset byl naměřen tak, aby zastoupení jednotlivých tříd bylo rovnoměrné. Rozložení naměřených dat pro fáze trénování, validaci a testování popisuje tabulka 7.1. S testovacími a validačními daty se neuronová síť při trénování nikdy nesetkala.

Při hledání CNN architektury byly použité proměnné a jejich rozsahy dle tabulky 7.2. Jednotlivé parametry byly použity pro trénování. Jeden konvoluční blok se skládal z jedné 1D konvoluční vrstvy a jedné redukční vrstvy. Počet filtrů se měnil podle hloubky konvolučního bloku. V prvním konvolučním bloku byl počet filtrů dle výběru tuneru. V každém následujícím bloku byl počet filtrů násoben hloubkou tohoto bloku. To znamená, že pro druhý blok byla volba tuneru vynásobena dvěma, pro třetí blok třemi atd. Totéž platí o počtu neuronů v plně propojených vrstvách po konvoluci. Počet neuronů ve vrstvě je násoben hloubkou dané vrstvy. Byly použity vrstvy `Dropout`, které náhodně dočasně vypínají určité procento neuronů v procesu trénování, aby nedocházelo k přetrénování sítě. Ukázku vložení konvolučních bloků do CNN architektury, lze vidět na výpisu 7.1.

Výpis 7.1: Vložení konvolučních bloků do CNN architektury.

```
1 kern_size = [9, 5, 5, 6, 7, 8]
2 for i in range(6):
3     model.add(
4         Conv1D(9*(i+1), kern_size[i], activation='relu')
5     )
6     model.add(AveragePooling1D(2, strides=2))
7     model.add(Dropout(0.1))
```

Výpis 7.2: Vložení klasifikační části do CNN architektury.

```
1 for i in range(2):
2     model.add(Dense(74*(i+1), activation='relu'))
3     model.add(Dropout(0.1))
```

Tab. 7.2: Použité proměnné a jejich rozsahy při hledání CNN architektury.

Proměnná	Rozsah
Počet konvolučních bloků	[1, 2, 3 ,4, 5, **6**, 7, 8]
Počet filtrů	[3, 5, 7, 8, **9**, 10, 12]
Počet plně propojených vrstev	[1, **2**, 3, 4]
Počet neuronů v MLP vrstvách	[32, 56, 64, **72**, 86, 128]

Tab. 7.3: Použité proměnné a jejich rozsahy při hledání MLP architektury.

Proměnná	Rozsah
Počet plně propojených vrstev	[2,4,12,18,**26**,38]
Počet neuronů v jedné vrstvě	[32,64,**128**,256,500,1024,2048]

Na výpisu 7.2, lze vidět vložení klasifikačních plně propojených vrstev do CNN architektury. Kompletní kód pro vytvoření a trénování navržených architektur je sdílen v GitHub repozitáři[3] nebo v elektronické příloze práce. Bylo vytvořeno konzolové uživatelské rozhraní pro trénování těchto modelů. Volbou parametrů je pak určeno, jaký model má být natrénován. Příklad užití je přiložen v daném adresáři.

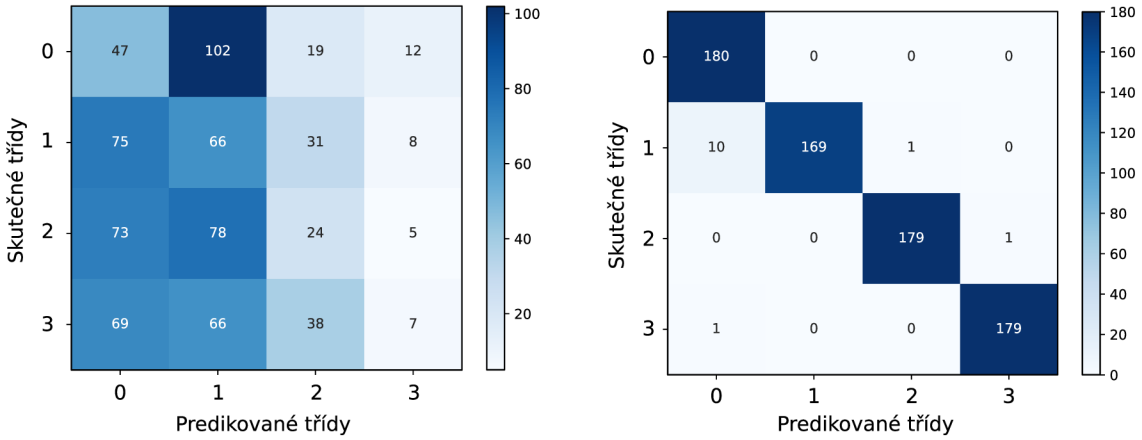
Parametry nalezené nejlepší architektury jsou v tabulce 7.2 vyznačeny tučně. Velikost filtrů byla hledána zvlášť, a to pro každý konvoluční blok zvlášť. Rozmezí hledaných velikostí filtrů bylo od 1 do 9. Nalezené nejvhodnější velikosti konvolučních bloků od prvního po poslední blok jsou 9, 5, 5, 6, 7 a 8. Po každém konvolučním bloku a každé plně propojené vrstvě je použita vrstva dropout s četností 0,1. Jako optimalizátor byl zvolen algoritmus ADAM, ztrátová funkce byla použita sparse categorical crossentropy a aktivační funkce byla vybrána ReLU. Trénování probíhalo na 50 epoch při velikosti dávek 32.

Volba proměnných a jejich rozsah u hledané MLP architektury popisuje tabulka 7.3. Volba hyper-parametrů u MLP architektury zůstala stejná jako u konvoluční architektury. Nejlepší architektura pak byla tvořena z 26 vrstev a každá vrstva byla tvořena ze 128 neuronů.

7.2 Výsledky modelů

V této části budou představeny výsledky navržených architektur popsaných výše. Bude uvedena jejich přesnost, úplnost a jejich f1-skóre, které zavádí kompromis mezi přesností a úplností. Dále budou uvedeny matice záměn jednotlivých modelů.

[3] `https://github.com/xmatus33/SITM_model_trainer`



(a) Vícevrstvá perceptronová síť.

(b) Konvoluční síť.

Obr. 7.2: Výsledné matice záměn.

Tabulka 7.4 popisuje dosažené výsledky obou navržených architektur. Je možno vidět, že perceptronová síť nedosahuje použitelné úrovně přesnosti, resp. síť se nepovedlo natrénovat vůbec. Čistá vícevrstvá perceptrovoná síť není schopná se z nepředzpracovaného náměru naučit vzorec vedoucí ke správné klasifikaci. Na obrázku 7.2 lze vidět matice záměn obou architektur. Pro zapojení do útoku SITM, tak byla vybrána navržená konvoluční síť, která dosahuje velmi dobrých výsledků.

Tab. 7.4: Výsledky navržených architektur.

Architektura	Přesnost	Úplnost	F1-skóre
Konvoluční	0,99	0,99	0,99
Perceptronová	0,24	0,24	0,19

Závěr

V této práci byla popsána problematika postranních kanálů, jejich definice, vysvětlení, použití a jejich druhy. Následovala část týkající se hlubokého učení, jeho principu a popisu stěžejních součástí. Třetí kapitola spojovala dvě již představené problematiky do jedné a sice útoky postranními kanály za pomoci hlubokého učení. Tato problematika je poměrně mladá a rychle se vyvíjející. Byl zmapován aktuální stav problematiky a byly představeny veřejně dostupné datasety. Kapitola 4, představila útok SITM, jeho kroky a aplikaci na symetrickou šifru AES-128.

Dále byla prozkoumána možnost analýzy pomocí korelačního koeficientu a její aplikace pro vytváření modelů hlubokého učení pro útoky postranními kanály. V rámci této kapitoly byla vytvořena aplikace pro efektivní realizaci této analýzy pomocí maticového výpočtu.

Následně byl představen útok na veřejně dostupný dataset ADCAD, kde byla navržena vlastní architektura modelu hlubokého učení. Model s touto architekturou byl následně natrénován a pomocí metriky entropie odhadu porovnán s dvěma referenčními modely.

Poslední kapitola se zaměřila na vylepšení útoku SITM pomocí metod hlubokého učení. Trénování probíhalo nad náměry pořízených v laboratořích univerzity VUT v Brně. Na těchto náměrech byla ukázána možnost zapojení hlubokého učení do procesu útoku SITM. Klasifikace náměrů tak nemusí probíhat ručně, ale je vykonána automaticky pomocí konvoluční neuronové sítě. Průměrování náměrů lze tak snížit z 1024 měření na čtyři měření. Tím dochází nejen ke zkrácení času potřebného k vykonání útoku, ale i k možnosti realizace útoku v podmínkách, kde by to bez hlubokého učení nebylo možné.

Literatura

[1] HALEVI, Tzipora a Nitesh SAXENA. Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios. International Journal of Information Security. 2015, 14(5), 443–456.

[2] HOSPODAR, Gabriel, Benedikt GIERLICHS, Elke DE MULDER, Ingrid VERBAUWHEDE a Joos VANDEWALLE. Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering. Springer, 2011, 1(4), 293–302.

[3] MARTINÁSEK, Zdeněk. Kryptoanalýza postranními kanály. Brno. Dizertační práce. Vysoké učení technické v Brně.

[4] KOCHER, Paul C., NEAL, Koblitz, ed. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. Advances in Cryptology – CRYPTO '96. Berlin, Heidelberg: Springer, 1996, 104–113. ISBN 978-3-540-68697-2.

[5] MAYER-SOMMER, Rita, KOC, Çetin K. a Christof PAAR, ed. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards. Cryptographic Hardware and Embedded Systems – CHES 2000. Berlin, Heidelberg: Springer, 2000, 78–92. ISBN 978-3-540-44499-2.

[6] MANGARD, Stefan, Elisabeth OSWALD a Thomas POPP. Power analysis attacks: Revealing the secrets of smart cards. Springer Science & Business Media, 2008(31).

[7] National Security Agency. TEMPEST: A Signal Problem [online]. 2007 [cit. 2023-05-12]. Dostupné z: `https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf`

[8] TSANG, James C., Jeffrey A. KASH a David P. VALLETT. Picosecond imaging circuit analysis. IBM J. Res. Dev. 2000, 44, 583–604.

[9] FERRIGNO, J. a M. HLAVAC. When AES blinks: Introducing optical side channel. Information Security, IET. 2008, 2, 94–98.

[10] RIJMEN, Vincent, Vincent RIJMEN a Bruce SCHNEIER, JACQUES, Jean, ed. The Block Cipher Rijndael. In: Smart Card Research and Applications. Springer Berlin Heidelberg, 2000, 277–284. ISBN 978-3-540-44534-0.

[11] TRAUTMANN, Jens, Arthur BECKERS, Lennert WOUTERS, Stefan WILDERMANN, Ingrid VERBAUWHEDE a Jürgen TEICH. Semi-Automatic Locating of Cryptographic Operations in Side-Channel Traces. IACR Transactions on Cryptographic Hardware and Embedded Systems. Ruhr-Universität Bochum, 2022(1), 345–366.

[12] KOCHER, Paul C., Joshua JAFFE a Benjamin JUN. Differential Power Analysis. Annual International Cryptology Conference. 1999.

[13] BRIER, Eric, Christophe CLAVIER a Francis OLIVIER. Correlation power analysis with a leakage model. In: International workshop on cryptographic hardware and embedded systems. Springer, 2004, 16–29.

[14] CHARI, Suresh, Josyula RAO a Pankaj ROHATGI. Template attacks. International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2002, 13–28.

[15] SCHINDLER, Werner. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. Journal of Mathematical Cryptology. 2008, 2(3), 291–310.

[16] CORON, Jean-Sébastien a Louis GOUBIN. On Boolean and Arithmetic Masking against Differential Power Analysis. In: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems: CHES '00. 7. Berlin, Heidelberg: Springer-Verlag, 2000, 231–237. ISBN 354041455X.

[17] RIVAIN, Matthieu a Emmanuel PROUFF. Provably Secure Higher-Order Masking of AES. Cryptology ePrint Archive, Paper, 2010(441).

[18] OULADJ, Maamar a Sylvain GUILLEY. SCA Countermeasures. In: Side-Channel Analysis of Embedded Systems: An Efficient Algorithmic Approach. Cham, 2021, 35–46. ISBN 978-3-030-77222-2.

[19] DING, Adam, Liwei ZHANG, Yunsi FEI a Pei LUO. A Statistical Model for Higher Order DPA on Masked Devices. Cryptology ePrint Archive, 2014(433).

[20] DE MEYER, Lauren, Oscar REPARAZ a Begül BILGIN. Multiplicative masking for AES in hardware. In: IACR Transactions on Cryptographic Hardware and Embedded Systems. 2018, 431–468.

[21] NASSAR, Maxime, Youssef SOUISSI, Sylvain GUILLEY a Jean-Luc DANGER. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2012, 1173–1178.

[22] KUTZNER, Sebastian a Axel POSCHMANN. On the security of RSM-presenting 5 first-and second-order attacks. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. Springer, 2014, 299–312.

[23] VON WILLICH, Manfred. A technique with an information-theoretic basis for protecting secret data from differential power attacks. In: Cryptography and Coding: 8th IMA International Conference Cirencester. 8. UK: Springer, 2001, 44–62.

[24] FUMAROLI, Guillaume, Ange MARTINELLI, Emmanuel PROUFF a Matthieu RIVAIN. Affine masking against higher-order side channel analysis. In: Selected Areas in Cryptography: 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers 17. Springer, 2011, 262–280.

[25] SUTSKEVER, Ilya, James MARTENS, George DAHL a Geoffrey HINTON. On the Importance of Initialization and Momentum in Deep Learning. Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28: ICML'13. Atlanta, GA, USA, 2013, III–1139–III–1147.

[26] ABIODUN, Oludare Isaac, Aman JANTAN, Abiodun Esther OMOLARA, Kemi Victoria DADA, Nachaat AbdElatif MOHAMED a Humaira ARSHAD. State-of-the-art in artificial neural network applications: A survey. Heliyon. Elsevier, 2018, 4(11).S

[27] KROSE, Ben a Patrick van der SMAGT. An introduction to neural networks. 2011.

[28] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep Learning. MIT Press, 2016.

[29] AGGARWAL, Charu C. Neural Networks and Deep Learning: A Textbook. Springer Cham, 2018. ISBN 978-3-319-94463-0.

[30] SHIRISH, Nitish, Jorge NOCEDAL, Ping Tak PETER, Dheevatsa MUDIGERE a Mikhail SMELYANSKIY. On large-batch training for deep learning: Generalization gap and sharp minima. 5th International Conference on Learning Representations. 2017.

[31] NIELSEN, Michael A. Neural Networks and Deep Learning. Determination Press, 2015.

[32] LYDIA, Agnes a Sagayaraj FRANCIS. Adagrad—an optimizer for stochastic gradient descent. Int. J. Inf. Comput. Sci. 2019, 6(5), 566–568.

[33] SHI, Naichen a Dawei LI. Rmsprop converges with proper hyperparameter. International conference on learning representation. 2021.

[34] KINGMA, Diederik P. a Jimmy BA. Adam: A Method for Stochastic Optimization. CoRR. 2015, abs/1412.6980.

[35] KIRANYAZ, Serkan, Onur AVCI, Osama ABDELJABER, Turker INCE, Moncef GABBOUJ a Daniel J. INMAN. 1D convolutional neural networks and applications: A survey. Mechanical Systems and Signal Processing. 2021, 151. ISSN 0888-3270.

[36] ALBAWI, Saad, Tareq Abed MOHAMMED a Saad AL-ZAWI. Understanding of a convolutional neural network. 2017 international conference on engineering and technology (ICET). IEEE, 2017, 1–6.

[37] JI, Shuiwang, Wei XU, Ming YANG a Kai YU. 3D Convolutional Neural Networks for Human Action Recognition. IEEE Computer Society, 2013, 35(1), 221–231. ISSN 0162-8828.

[38] TRAN, Du, Lubomir BOURDEV, Rob FERGUS, Lorenzo TORRESANI a Manohar PALURI. Learning Spatiotemporal Features With 3D Convolutional Networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2015.

[39] KUBOTA, Takaya, Kota YOSHIDA, Mitsuru SHIOZAKI a Takeshi FUJINO. Deep learning side-channel attack against hardware implementations of AES. Microprocessors and Microsystems, 2021, 87. ISSN 0141-9331.

[40] LI, Huizhong, Jingdian MING a Yongbin ZHOU. Assessment of Addition-Chain-Based Masked S-Box Using Deep-Learning-Based Side-Channel Attacks. Security and communication networks. London: Hindawi, 2022, 1–11. ISSN 1939-0114.

[41] PROUFF, Emmanuel, Remi STRULLU, Ryad BENADJILA, Eleonora CAGLI a Cecile DUMAS. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. Cryptology ePrint Archive. 2018(053).

[42] STANDAERT, François-Xavier, Tal G. MALKIN a Moti YUNG, JOUX, Antoine, ed. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Advances in Cryptology - EUROCRYPT 2009. Berlin, Heidelberg: Springer, 2009, 443–461.

[43] MASSEY, J.L. Guessing and entropy. Proceedings of 1994 IEEE International Symposium on Information Theory. 1994. ISSN 0-7803-2015-8.

[44] PANOFF, Max, Honggang YU, Haoqi SHAN a Yier JIN. A Review and Comparison of AI-Enhanced Side Channel Analysis. 2022. New York, NY, USA: Association for Computing Machinery, 2022, 18(3).

[45] PICEK, Stjepan, Guilherme PERIN, Luca MARIOT, Lichao WU a Lejla BATINA. SoK: Deep Learning-based Physical Side-channel Analysis. Cryptology ePrint Archive, 2021(1092).

[46] GASPAR, Lubos, Gaëtan LEURENT a François-Xavier STANDAERT. Hardware implementation and side-channel analysis of lapin. Cryptographers' Track at the RSA Conference. Springer, 2014, 206–226.

[47] LECUN, Yann, Corinna CORTES a Christopher J.C. BURGES. The mnist database of handwritten digits.

[48] ASCADv2. GitHub [online]. Agence nationale de la scurit des systmes d'information (ANSSI), 2021 [cit. 2023-05-12]. Dostupné z: `https://github.com/ANSSI-FR/ASCAD`

[49] BHASIN, Shivam, Dirmanto JAP a Stjepan PICEK. AES_HD. GitHub [online]. 2018 [cit. 2023-05-12]. Dostupné z: `https://github.com/AESHD/AES_HD_Datase`

[50] BHASIN, Shivam, Dirmanto JAP a Stjepan PICEK. AES_HD dataset - 50 000 traces [online]. AISyLab repository, 2020 [cit. 2023-05-12].

[51] BHASIN, Shivam, Dirmanto JAP a Stjepan PICEK. AES_HD dataset - 500 000 traces [online]. AISyLab repository, 2020 [cit. 2023-05-12].

[52] AES_HD_MM. CHEST [online]. 2014 [cit. 2023-05-12]. Dostupné z: `https://chest.coe.neu.edu/?currentpage=POWER_TRACE_LINK&software=ptmasked`

[53] AES_RD. GitHub [online]. 2009 [cit. 2023-05-12]. Dostupné z: `https://github.com/ikizhvatov/randomdelays-traces`

[54] DPA_V2 [online]. T. TELECOM ParisTech SEN research group, 2010 [cit. 2023-05-12]. Dostupné z: `http://www.dpacontest.org/v2/`

[55] DPA_V4.1 [online]. T. TELECOM ParisTech SEN research group, 2013 [cit. 2023-05-12]. Dostupné z: `http://www.dpacontest.org/v4/`

[56] MARTINÁSEK, Zdeněk. Counter measure techniques for cryptographic algorithms eliminating power analysis attacks. Brno, 2020. Habilitační práce. Brno University of Technology.

[57] DPA_V4.2 [online]. T. TELECOM ParisTech SEN research group, 2014 [cit. 2023-05-12]. Dostupné z: http://www.dpacontest.org/v4/42_doc.php

[58] CHES CTF 2018. Riscure [online]. 2018 [cit. 2022-12-04]. Dostupné z: https://chesctf.riscure.com/2018/news

[59] Portability Dataset [online]. Aisylab Datasets, 2020 [cit. 2023-05-12]. Dostupné z: http://aisylabdatasets.ewi.tudelft.nl/

[60] BHASIN, Shivam, Anupam CHATTOPADHYAY, Annelie HEUSER, Dirmanto JAP, Stjepan PICEK a Ritu Ranjan SHRIVASTWA. Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis. Cryptology ePrint Archive, 2019(661).

[61] MARTINASEK, Zdenek, Jan HAJNY a Lukas MALINA. Optimization of Power Analysis Using Neural Network. 2014, 94–107. ISBN 978-3-319-08301-8.

[62] MARTINASEK, Zdenek, Lukas MALINA a Petr DZURENDA. Profiling power analysis attack based on MLP in DPA contest V4.2. 2016 39th International Conference on Telecommunications and Signal Processing (TSP). 2016, 223–226.

[63] MARTINASEK, Zdenek, Lukas MALINA a Krisztina TRASY. Profiling Power Analysis Attack Based on Multi-layer Perceptron Network. 2015.

[64] CAGLI, Eleonora, Cécile DUMAS a Emmanuel PROUFF. Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures – Profiling Attacks without Pre-Processing. Cryptology ePrint Archive. 2017, 740.

[65] MAGHREBI, Houssem, Thibault PORTIGLIATTI a Emmanuel PROUFF. Breaking Cryptographic Implementations Using Deep Learning Techniques. Cryptology ePrint Archive. 2016, (921).

[66] ZAID, Gabriel, Lilian BOSSUET, Amaury HABRARD a Alexandre VENELLI. Methodology for Efficient CNN Architectures in Profiling Attacks – Extended Version. Cryptology ePrint Archive, 2019(803).

[67] WOUTERS, Lennert, Victor ARRIBAS, Benedikt GIERLICHS a Bart PRENEEL. Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks. Ruhr-Universität Bochum, 2020(3), 147–168.

[68] DAS, Debayan, Anupam GOLDER, Josef DANIAL, Santosh GHOSH, Arijit RAYCHOWDHURY a Shreyas SEN. X-DeepSCA: Cross-Device Deep Learning Side Channel Attack. 2019 56th ACM/IEEE Design Automation Conference (DAC). 2019, 1–6.

[69] WU, Lichao, Guilherme PERIN a Stjepan PICEK. I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. Cryptology ePrint Archive, 2020(1293).

[70] RIJSDIJK, Jorai, Lichao WU, Guilherme PERIN a Stjepan PICEK. Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems. Ruhr-Universität Bochum, 2021(3), 677–707.

[71] BAKER, Bowen, Otkrist GUPTA, Nikhil NAIK a Ramesh RASKAR. Designing neural network architectures using reinforcement learning. ArXiv preprint arXiv:1611.02167. 2016.

[72] YUANYUAN, Zhou a Standaert FRANÇOIS-XAVIER. Eep learning mitigates but does not annihilate the need of aligned traces and a generalized ResNet model for side-channel attacks. Journal of Cryptographic Engineering. 2019(10), 85–95.

[73] MASURE, Loïc a Rémi STRULLU. Side Channel Analysis against the ANSSI's protected AES implementation on ARM. Cryptology ePrint Archive, 2021(592).

[74] KARAYALCIN, Sengim a Stjepan PICEK. Resolving the Doubts: On the Construction and Use of ResNets for Side-channel Analysis. Cryptology ePrint Archive, 2022, 2022(963).

[75] JIN, Minhui, Mengce ZHENG, Honggang HU a YU. An enhanced convolutional neural network in side-channel attacks and its visualization. ArXiv preprint arXiv:2009.08898. 2020.

[76] BHASIN, Shivam, Jakub BREIER, Xiaolu HOU, Dirmanto JAP, Romain POUSSIER a Siang Meng SIM. SITM: See-In-The-Middle Side-Channel Assisted Middle Round Differential Cryptanalysis on SPN Block Ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems. 2019, 2020(1), 95–122.

[77] PARK, Jonghyun, Hangi KIM a Jongsung KIM. Improved See-In-The-Middle Attacks on AES. Information Security and Cryptology – ICISC 2021. Cham: Springer International Publishing, 2022, 271–279. ISBN 978-3-031-08896-4.

[78] TILLICH, Stefan, Christoph HERBST a Stefan MANGARD. Protecting AES software implementations on 32-bit processors against power analysis. Applied Cryptography and Network Security: 5th International Conference, ACNS 2007. Zhuhai, China: Springer, 2007, 5, 141–157.

[79] SCHRAMM, Kai a Christof PAAR, POINTCHEVAL, David, ed. Higher Order Masking of the AES. Topics in Cryptology – CT-RSA 2006. Berlin, Heidelberg: Springer, 2006, 208–225. ISBN 978-3-540-32648-9.

[80] BREIER, Jakub, Dirmanto JAP a Shivam BHASIN. SCADPA: Side-Channel Assisted Differential-Plaintext Attack on Bit Permutation Based Ciphers. Cryptology ePrint Archive. 2017, 1166.

[81] BREIER, Jakub, Dirmanto JAP, Xiaolu HOU a Shivam BHASIN. On Side Channel Vulnerabilities of Bit Permutations in Cryptographic Algorithms. IEEE Transactions on Information Forensics and Security. 2020, 15, 1072–1085.

[82] BREIER, Jakub a Xiaolu HOU. SITM Computation Example. 2023.

[83] MARTINÁSEK, Zdeněk, Vlastimil ČLUPEK, Václav ZEMAN a Petr SYSEL. Základní metody diferenciální proudové analýzy. Elektrorevue. 2013, 15(1), 9-19.

[84] CPA_Implementation. GitHub [online]. 2021 [cit. 2023-05-12]. Dostupné z: `https://github.com/djtfoo/CPA_Implementation`

[85] dpa-on-aes. GitHub [online]. 2021 [cit. 2023-05-12]. Dostupné z: `https://github.com/nvietsang/dpa-on-aes/blob/master/main_cpa.py`

[86] NumPy [online]. 2006 [cit. 2023-03-19]. Dostupné z: `https://numpy.org/`

[87] DWORKIN, Morris, Elaine BARKER, James NECHVATAL, James FOTI, Lawrence BASSHAM, E. ROBACK a James DRAY. Advanced Encryption Standard (AES). Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2001.

[88] MARTINASEK, Zdenek, Felix IGLESIAS, Lukas MALINA a Josef MARTINASEK. Crucial pitfall of DPA Contest V4.2 implementation. Security Comm. Networks. 2016, 9, 6094–6110.

[89] Keras. Chollet, Francois and others. 2015. [cit. 2022-09-09] Dostupné z: `https://keras.io`

[90] JUPYTER TEAM. JupyterLab [online]. Project Jupyter, 2015 [cit. 2023-05-15]. Dostupné z: `https://docs.jupyter.org/en/latest/#`

Seznam symbolů a zkratek

ASCAD Advanced Side-Channel Analysis Dataset

CNES Centre National d'Études Spatiales (Francouzská vesmírná agentura)

CNN Convolutional Neural Network (Konvoluční neuronová sítě)

CPA Correlation Power Analysis (Analýza korelačním koeficientem)

DDT Difference Distribution Table (Tabulka distribuce rozdílů)

DL Deep Learning (Hluboké učení)

DL-SCA Deep Learning Side-Channel Attack (Kryptoanalýza postranních kanálů pomocí hlubokého učení)

DNN Deep Neural Network (Hluboká neuronová síť)

DPA Differential Power Analysis (Diferenciální proudová analýza)

EM Electromagnetic (Elektromagnetické)

GD Gradien Descent (Gradientní sestup)

GPU Graphics Processing Unit (Grafická karta)

CHES Conference on Cryptographic Hardware and Embedded Systems

MNIST Modified National Institute of Standards and Technology

ML Machine Learning (Strojové učení)

MLP Multi-Layer Perceptrons (Vícevrstvá perceptronová síť)

NN Neural Network (Neuronová síť)

NSA National Security Agency

PA Power Analysis (Proudová analýza)

PICA Picosecond Imaging Circuit Analysis (Pikosekundová zobrazovací obvodová analýza)

PoI Points of Interest (Body zájmu)

RSM Rotating S-Box Masking (Maskování rotací S-boxu)

SCA Side-Channel Analysis (Kryptoanalýza postranních kanálů)

SCADPA Side-Channel Assisted Differential Plaintext Attack

SGD Stochastic Gradient Descent (Stochastický Gradientní Sestup)

SITM See-In-The-Middle

SPA Simple Power Analysis (Jednoduchá proudová analýza)

SPN Substitution-permutation network

XOR Exclusive or (Výlučné nebo)

A Obsah elektronické přílohy

Elektronická příloha obsahuje veškerý kód použitý v rámci této diplomové práci. Obsahuje tři hlavní části a to podle kapitol 5, 6 a 7. Pro sdílení kódu byly využity GitHub repozitáře[1] a GitHub Gists[2]. Elektronická příloha je lokální verzí těchto GitHub repozitářů pořízenou ke dni vydání práce. Elektronická příloha je dostupná na stránkách VUT[3].

Tématicky je příloha členěna do adresářů, dle odpovídajících repozitářů. V každém z nich se nachází `README.md` soubor, který popisuje k čemu daný repozitář slouží. Samostatný soubor `Generate_DDT.py` je rychlou ukázkou způsobu generace tabulky distribuce rozdílů pro AES-128. Soubor `train_ASCAD.ipynb` je pak prostředím Jupyter LAB, které sloužilo pro trénování a validaci ASCAD modelů.

```
/ ..... kořenový adresář přiloženého archivu
├── CPA ..... Python CPA maticová implementace
│   ├── cpa.py
│   ├── dpa_parser.py
│   ├── example_traces_300.npy
│   ├── index_file_striped.txt
│   ├── plotting.py
│   ├── README.md
│   └── tables.py
├── SITM_model_trainer ..... trénování modelů pro SITM útok
│   ├── dataset
│   │   └── README.md
│   ├── README.md
│   ├── train_model.py
│   └── trainlib.py
├── SITM_using_deep_learning ..... realizace SITM útoku s NN
│   ├── example_diff_trace.npy
│   ├── README.md
│   └── SITM.py
├── Generate_DDT.py ..... ukázka generování DDT
└── training_ASCAD.ipynb ..... trénování a validace ASCAD
```

[1] `https://github.com/xmatus33`

[2] `https://gist.github.com/xmatus33`

[3] `https://www.vut.cz/studenti/zav-prace/detail/151258`