FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Demonstrace metod analýzy postranních kanálů
Student:	Marek Bizík
Vedoucí:	Ing. Jiří Buček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Vyberte vhodnou šifru (např. AES).
Proveďte měření postranního kanálu na platformě ChipWhisperer Nano.
Demonstrujte vybrané metody útoku postranním kanálem a navzájem je porovnejte:
- korelační odběrovou analýzu (CPA),
- útok pomocí šablon (template attack),
- aspoň jednu vybranou metodu strojového učení.

Použijte jazyk Python s využitím knihovny Numpy a dalších vhodných knihoven.
K porovnání využijte metriky jako je PGE (částečná entropie odhadu), počet nutných měření a délku výpočtu.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 6. února 2023 v Praze.

Bakalářská práce

DEMONSTRACE METOD ANALÝZY POSTRANNÍCH KANÁLŮ

Marek Bizík

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Jiří Buček, Ph.D.
11. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Především bych chtěl poděkovat mému vedoucímu práce za jeho trpělivost, podporu a cenné rady, které mi pomohly překonat mnoho překážek a dosáhnout lepších výsledků.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 .....

Abstrakt

V této bakalářské práci je zkoumáno a porovnáváno několik metod útoku postranním kanálem na šifrovací algoritmus AES. Práce se zaměřuje na tři základní metody útoku: korelační odběrovou analýzu, útok pomocí šablon a metodu strojového učení, konkrétně náhodný les. Cílem práce je analyzovat a porovnat účinnost těchto metod na platformě ChipWhisperer Nano s použitím programovacího jazyka Python a jeho knihoven.

V praktické části je provedena implementace experimentů pro každou metodu útoku. Účinnost metod je zhodnocena na základě zvolených metrik, jako je částečná entropie odhadu, počet nutných měření a délka výpočtu. Počet nutných měření je vyhodnocen na 32 u korelační odběrové analýzy, 17 u útoku pomocí šablon a 17 u metody náhodného lesa. Výsledky této práce ukazují, že metoda strojového učení je nejúčinnější v metrikách částečné entropie odhadu, avšak v délce výpočtu má nejhorší výsledky. V délce výpočtu má nejlepší výsledky útok pomocí šablon.

Tato práce přispívá k lepšímu porozumění účinnosti jednotlivých metod útoku postranním kanálem a může sloužit jako základ pro další výzkum v oblasti zabezpečení šifrování.

Klíčová slova analýza postranních kanálů (SCA), porovnání metod SCA, útoky na šifrování, AES (Advanced Encryption Standard), ChipWhisperer

Abstract

In this bachelor's thesis, several side-channel attack methods on the AES encryption algorithm are examined and compared. The focus of the study is on three basic attack methods: Correlation Power Analysis, Template Attack, and a machine learning method, specifically Random Forest. The aim of the work is to analyze and compare the effectiveness of these methods on the ChipWhisperer Nano platform, using the Python programming language and its libraries.

In the practical part, the implementation of experiments for each attack method is carried out. The effectiveness of the methods is evaluated based on selected metrics, such as Partial Guessing Entropy, the number of necessary measurements, and computation time. The number of necessary measurements is evaluated to be 32 for Correlation Power Analysis, 17 for Template Attack, and 17 for the Random Forest method. The results of this work show that the machine learning method is the most effective in Partial Guessing Entropy metrics, but it has the worst results in terms of computation time. The Template Attack has the best results in computation time.

This work contributes to a better understanding of the effectiveness of individual side-channel attack methods and can serve as a basis for further research in the field of encryption security.

Keywords Side channel analysis (SCA), Comparison of SCA methods, Attacks on encryption, AES (Advanced Encryption Standard), ChipWhisperer

Seznam zkratek

AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DPA	Differential Power Analysis
PGE	Partial Guessing Entropy
POI	Point of Interest
SCA	Side-Channel Attack
SPA	Simple Power Analysis

Kapitola 1
Úvod

1.1 Úvod

Šifrování dat se stalo v dnešní době nezbytností pro ochranu citlivých informací před neoprávněným přístupem. Mezi nejčastěji používané šifrovací algoritmy patří AES (Advanced Encryption Standard), který se vyznačuje vysokou úrovní bezpečnosti a rychlostí zpracování dat. Nicméně, při použití AES šifrování mohou být data stále ohrožena útoky pomocí postranních kanálů, které umožňují útočníkům získat tajné klíče použité pro šifrování například pomocí měření spotřeby energie nebo elektromagnetického záření.

Toto téma bylo vybráno z důvodu aktuálnosti bezpečnostních hrozeb na šifru AES pomocí postranních kanálů. Vzhledem k rostoucímu významu zabezpečení v oblasti informačních technologií je nezbytné tyto metody zkoumat, aby bylo možné lépe chránit citlivé údaje a systémy.

V této práci se nejprve podrobněji vysvětlí princip šifrování pomocí AES a problematika postranních kanálů. Následně budou popsány jednotlivé metody útoků a vytvořeny experimenty pro porovnání jejich úspěšnosti. V závěru budou prezentovány výsledky a zhodnocení efektivity jednotlivých metod.

Cílem této bakalářské práce je porovnat tři různé metody útoků na AES pomocí postranních kanálů. Konkrétně se jedná o metodu analýzy korelace napětí (CPA), metodu template attack a metodu využívající strojové učení. Měření probíhá na platformě ChipWhisperer, což je open-source platforma, která umožňuje měření a analýzu postranních kanálů na mikrokontrolerech a dalších zařízeních.

Výsledky této práce mohou přispět k lepšímu porozumění zranitelnosti AES vůči útokům pomocí postranních kanálů a k lepšímu návrhu bezpečnějších systémů pro zabezpečení dat.

Kapitola 2
Teoretická část

2.1 Šifry a jejich důležitost

2.1.1 Šifra AES

AES je symetrická bloková šifra. Šifrování i dešifrování probíhá pomocí stejného klíče. Tento klíč může mít různé délky, podle použité varianty. Jednotlivé varianty jsou AES-128, AES-192 a AES-256, lišící se primárně ve velikostech klíče v bitech, jak je specifikováno v názvu. V této práci budeme pracovat výhradně s verzí AES-128, neboli s verzí, jež má klíč velikosti 128 bitů.

Práce je zaměřená na šifrování a útok na něj. Na dešifrování nebudeme útočit a nebude pro naše experimenty důležité, proto v této práci princip dešifrování nebudeme popisovat. Ze šifrování bude popsána pouze část relevantní pro tuto práci.

2.1.1.1 Popis šifry AES-128

Vstupní text se rozděluje na bloky, které mají velikost 128 bitů. Stejně velký je i výstupní blok a vnitřní Stav šifry. Stav je reprezentován jako matice 4x4 bytů, kde každý sloupec je jedno další slovo (jedno slovo je informace délky 32 bitů). Vstupní text do bloku označíme P, a klíč označíme K.

Na začátku je blok vstupního textu zkopírován do Stavu. Poté proběhne inicializační přidání rundovního klíče (operace AddRoundKey). Následně probíhají jednotlivé rundy, přičemž rund je celkem 10. V každé rundě (kromě poslední) proběhnou po sobě jdoucí operace SubBytes, ShiftRows, MixColumns a AddRoundKey.

S klíčem se provádí expanze. Expanze klíče vygeneruje klíč o délce 44 slov. Každá runda totiž potřebuje 4 další slova (rundovní klíče) na provedení AddRoundKey, a k tomu probíhá před rundami inicializační AddRoundKey. Expandovaný klíč má tedy celkem 11 rundovních klíčů k_i, kde $i \in \{0, 1, 2, \ldots, 10\}$. První 4 slova expandovaného klíče jsou totožná s původním klíčem, tudíž $k_0 = K$.

Operace AddRoundKey přidává rundovní klíč do Stavu S' bitovou operací XOR Stavu S a rundovního klíče k_i: $S' = (S \oplus k_i)$.

Operace SubBytes zajišťuje nelinearitu šifry pomocí substituce jednotlivých bytů Stavu podle substituční tabulky (S-box): $S' = Sbox(S)$. [1]

Výsledek první posloupnosti operací, konkrétně AddRoundKey a SubBytes se tedy dá zapsat jako: $S' = Sbox(P \oplus K)$.

Pseudokód celého šifrování lze vidět v obrázku 2.1, ovšem dále AES popisovat nebudeme, jelikož další operace nejsou důležité pro tuto práci.

```
AES-128(byte in[16], byte out[16], word w[44])
    begin
        byte state[4, 4]
        state = in
        AddRoundKey(state, w[0, 3])
        for round = 1 step 1 to 9
            SubBytes(state)
            ShiftRows(state)
            MixColumns(state)
            AddRoundKey(state, w[round*4, (round+1)*4-1])
        end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[40, 43])
    out = state
    end
```

■ **Obrázek 2.1** Pseudo kód šifry AES-128 [1]

2.2 Analýza postranních kanálů

Analýza postranních kanálů je metoda kryptoanalýzy, která se zaměřuje na získání informací o šifrovacím klíči z vedlejších účinků, které vznikají během šifrování nebo dešifrování dat. Tyto vedlejší účinky mohou zahrnovat spotřebu energie, elektromagnetické záření, dobu zpracování a další fyzikální projevy, které jsou spojeny s konkrétními operacemi prováděnými šifrovacím algoritmem. V této práci se zabýváme výhradně útoky analyzujícími spotřebu energie.

SCA se stala nedílnou metodou kryptoanalýzy, především proto, že útoky založené na SCA nezávisí na slabostech samotného šifrovacího algoritmu, ale na jeho implementaci. To znamená, že i šifry, které jsou považovány za bezpečné z hlediska matematické a teoretické kryptoanalýzy, mohou být zranitelné vůči útokům postranními kanály, pokud jejich implementace není navržena tak, aby minimalizovala vedlejší účinky, které lze použít k získání informací o klíči. [2]

2.2.1 Rozdělení útoků dle profilace

Útoky postranními kanály se mohou dělit podle různých kritérií, my v této práci používáme hlavně rozdělení na profilované a neprofilované.

1. **Profilované útoky:** U profilovaných útoků má útočník přístup k identickému či podobnému zařízení, jako je to, na které útočí. Útok je tak rozdělen na dvě části: profilovací část a samotný útok. Při profilovací části útočník ze zařízení, které má pod kontrolou, získá data pomocí postranních kanálů, zatímco na něm probíhá šifrování se známým klíčem. To mu umožňuje analyzovat tato data oproti použitému klíči a sestavit nějaký profil nebo model toho, jak se dané zařízení chová. Ten má poté k dispozici při samotném útoku, kde útočí na zařízení, u kterého již použitý klíč neví a chce ho zjistit pomocí postranních kanálů. Mezi tyto útoky se řadí například útoky pomocí šablon a útoky pomocí metod strojového učení.

2. **Neprofilované útoky:** U neprofilovaných útoků útočník nevyužívá nebo nemá k dispozici podobné či podobně se chovající zařízení. Útok je postaven výhradně na analýze dat naměřených již na samotném zařízení, na které se útočí. Mezi tyto útoky patří například diferenciální odběrová analýza, či korelační odběrová analýza.

2.2.2 Měřená data

Pokud provádíme měření, je potřeba specifikovat, co měříme a co získáme. Jak jsme uváděli, měří se proudová spotřeba v průběhu času na daném zařízení, které v našem případě provádí šifrování. Z toho nám vznikne matice $\mathbf{X}$ o m řádcích a n sloupcích. Počet řádků vyjadřuje počet průběhů, které jsme měřili, neboli kolikrát jsme nechali dané zařízení šifrovat, zatímco měříme jeho odběr. Počet sloupců vyjadřuje počet časových okamžiků, ve kterých jsme naměřili odběr daného zařízení. Dále máme k dispozici všechny vstupní texty, které byly použity u jednotlivých průběhů. Stejná data získáme i při profilovací fázi profilovaných útoků, zde máme navíc k dispozici i použité klíče u jednotlivých stop.

Při provádění měření musíme určit, jak budeme postupovat s výběrem klíče a vstupního textu použitých pro šifrování. Pokud provádíme měření, pomocí kterého chceme již útočit na dané zařízení, můžeme si vybírat pouze vstupní text. Klíč je daný a pro nás neznámý, chceme ho zjistit. Pokud bychom zde pro měření jednotlivých stop použili vždy stejný vstupní text, dostaneme mezi jednotlivými stopami pouze malý rozptyl, u kterého navíc nebude závislost na vstupních datech ani klíči, jelikož se v průběhu nemění. Tato neměnnost je nežádoucí, což budeme diskutovat později. Pro takové útoky (většinou s málo získanými stopami a se stejným nebo podobným vstupním textem) se často používá jednoduchá odběrová analýza [3]. V této práci budeme při měření stop, na které budeme útočit, využívat různých vstupních textů.

U profilovaných útoků přibývá profilovací fáze. V profilovací fázi máme možnost si zvolit jak vstupní text, tak klíč. Zde existují dva přístupy, model *zvoleného vstupního textu* a model *známého vstupního textu* [4]. V rámci těchto modelů řešíme, na základě jakých vstupních textů vytvoříme profil tohoto útoku. V případě modelu *zvoleného vstupního textu* máme možnost zvolit si, jaké vstupní texty budou použity při samotném útoku. Je to teoreticky výhodnější model, jelikož můžeme sestavit profil pouze na tomto zvoleném vstupním textu. Druhý model *známého vstupního textu* předpokládá, že si nemůžeme zvolit použitý vstupní text při útoku, ale alespoň ho známe. V tomto případě musíme sestavit profil, pomocí kterého budeme schopni útočit na zařízení s jakýmkoli vstupním textem. V této práci si zvolíme ten omezenější model, model *známého vstupního textu*, jelikož chceme zajistit větší generalizaci a replikovatelnost útoků. Zde tedy volíme vždy různý vstupní text a různý klíč. Obecně chceme, aby byl útok srovnatelně účinný na všechny možné klíče. Pokud bychom nechali stejný klíč při profilování, může být výsledný profil zaměřený pouze na tento klíč a nefungovat na ostatních.

2.2.3 Model úniku dat

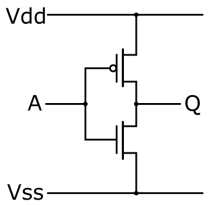
Při provádění útoků postranními kanály potřebujeme sestavit model úniku dat, na základě jehož budeme při samotném útoku odhadovat skutečné úniky dat. [4] Vezmeme v potaz nějakou vnitřní hodnotu kryptografické operace, která je výsledkem funkce vstupního textu a tajného klíče. Tato vnitřní hodnota je náchylná k úniku dat, a operace s touto hodnotou se dá odhalit pomocí postranních kanálů. V této práci se budeme zabývat útoky analyzujícími spotřeby energie, které získáme pomocí postranních kanálů. V tom případě potřebujeme nějak odhadnout, jak bude souviset vnitřní hodnota s reálnou spotřebou energie. Tomu říkáme model spotřeby energie.

Funkce použitá pro vnitřní hodnotu musí tedy záviset jak na vstupním textu, tak na tajném klíči. Pro tyto účely můžeme zvolit například výstup operace SubBytes v první rundě šifrování AES. Zde víme ze sekce 2.1.1.1, že to je výsledek operace XOR na vstupní text a tajný klíč (inicializační operace AddRoundKey) nahrazen hodnotou z tabulky S-box (operace SubBytes v první rundě): $VH = Sbox(P \oplus K)$, kde VH je vnitřní hodnota. Zde vidíme, že pokud bychom zjistili tuto vnitřní hodnotu, stačí nám vyhledat index této hodnoty v tabulce S-box, a následně provést XOR tohoto indexu se vstupním textem a již dostaneme tajný klíč: $K = Sbox^{-1}(VH) \oplus P$.

Únik dat stanovíme na základě spotřeby energie, která je závislá na hodnotě zpracovávaných dat (a tedy vnitřních hodnotách). Pro útoky tohoto typu je často nutné namapovat hodnoty dat zpracovávaných napadeným zařízením na hodnoty spotřeby energie. Zvláště se zaměříme

na model Hammingovy váhy, který se běžně používá pro útoky analýzou spotřeby energie [5]. Hammingova váha informačních dat je dána počtem jedničkových bitů v těchto datech. Jak tato hodnota souvisí se spotřebou energie je třeba vysvětlit pomocí CMOS obvodů.

CMOS obvody jsou základem moderních integrovaných obvodů a jsou široce používány v kryptografických zařízeních. Tyto obvody se skládají z komplementárních sad tranzistorů (hradel), viz obrázek 2.2. V klidovém stavu, kdy neprobíhají žádné výpočty, mají tzv. statickou spotřebu energie, kdy je jeden z páru tranzistorů vždy v nevodivém stavu, proto je spotřeba energie minimální. Pokud dochází ke změně stavu (0 na 1 nebo 1 na 0), mají tzv. dynamickou spotřebu energie. V tomto stavu je spotřeba energie mnohem větší, což je způsobeno hlavně tím, že dochází k nabíjení nebo vybíjení parazitních kapacit hradla a zároveň dochází ke krátkodobému zkratu. Ten je způsoben tím, že na malou ale nezanedbatelnou dobu jsou oba tranzistory vodivé. Obecně je tedy dynamická spotřeba významně větší, než ta statická, a závisí na konkrétních datech, která jsou tímto obvodem zpracovávána. Zároveň je důležitý fakt, že spotřeby při změnách stavu z 0 na 1 a z 1 na 0 nejsou totožné. Například přechod z 0 na 1 může mít větší spotřebu než ten opačný. Je to způsobeno hlavně rozdílnými vlastnostmi tranzistorů v hradle, které způsobují rozdílnou spotřebu energie, a zároveň parazitní kapacity mezi výstupem hradla a napájecím napětím obecně nejsou stejné jako mezi výstupem hradla a zemí. [3]



Obrázek 2.2 CMOS Invertor

Model Hammingovy váhy se obvykle používá, pokud útočník nemá žádné informace o síti, nebo pokud nezná po sobě jdoucí hodnoty dat pro nějakou známou část sítě. Pokud by je znal, mohl by využít Hammingovy vzdálenosti, která počítá počet přechodů, které, jak jsme popsali, jsou velice důležité pro určování spotřeby energie CMOS obvodů. V případě modelu Hammingovy váhy útočník předpokládá, že spotřeba energie je úměrná právě počtu nastavených jedničkových bitů ve zpracovávaných hodnotách dat. Model Hammingovy váhy je často spojován se skutečnou spotřebou energie, ačkoli vztah mezi nimi může být slabý. Jak bylo popsáno, přechod z 0 na 1 může mít u hradla větší spotřebu než ten opačný, z čehož vyplývá možnost použití modelu Hammingovy váhy. Pro tu by v tomto případě mělo platit, že čím větší je Hammingova váha dat, tím větší mají spotřebu energie.

2.2.4 Použité metody analýzy

Existuje více metod analýzy postranních kanálů, které se liší podle způsobu, jakým získávají a zpracovávají informace z postranních kanálů. V této práci jsme již vysvětlili rozdíl mezi profilovanými a neprofilovanými.

Mezi neprofilovanými budeme implementovat jednu metodu, a to je korelační odběrová analýza, která je v zadání této práce. Tato metoda byla navrhnuta jako vylepšení do té doby používané metody DPA, diferenciální odběrové analýzy [2]. CPA je obecně účinnější metoda, která tratí pouze na faktu, že je náročnější ji implementovat [5].

Mezi profilované můžeme zařadit zbývající dvě metody, které máme v zadání a budeme je implementovat. To je útok pomocí šablon a metoda strojového učení. Útok pomocí šablon byl navržen v roce 2002 jako vylepšení dosavadních útoků na postranní kanály [6]. Jejich přínos je v zavedení profilovací fáze, díky níž dokáže tato metoda vykonávat útoky s velmi omezeným

počtem průběhů, což může být až od jednoho průběhu. Tento přístup prolamuje různá opatření kryptografických implementací, která neumožňují získat velké množství průběhů spotřeby zařízení, na které se útočí. Taková opatření jsou velice účinná v kontextu klasických metod analýzy, jako je DPA či CPA. Ovšem útoky pomocí šablon zavádějí šablony, které lze získat z jakéhokoli zařízení, které bude mít stejný či podobný průběh spotřeby pro stejná data a klíč.

2.2.4.1 Korelační odběrová analýza (CPA)

Korelační odběrová analýza se zaměřuje na analýzu korelace mezi měřenými hodnotami spotřeby energie a hypotetickými hodnotami, které by vznikly při zpracování různých částí šifrovacího klíče. Tato metoda je založena na předpokladu, že existuje statistická závislost mezi skutečnými hodnotami a hodnotami odvozenými z klíče.

Pro sestavení hypotetických hodnot se musí stanovit model úniku dat, který byl představen v sekci 2.2.3.

Samotný útok vyžaduje znalost vstupního textu a naměřené hodnoty odběru daného zařízení při šifrování, matici $\mathbf{X}$. V případě, že útočíme vždy na jeden byte, se pro každé šifrování vytvoří pro každou možnou hodnotu daného bytu klíče hypotéza o vnitřní hodnotě, které změříme Hammingovu váhu a tu uložíme. Tak vznikne matice $\mathbf{Y}$, s 256 sloupci, pro každý možný klíč jeden sloupec, a m řádky. Z těchto dvou matic chceme zjistit, která hypotéza je ta správná. To provedeme pomocí měření míry korelace mezi nimi, která nám stanoví, která hypotéza je nejpravděpodobnější. Jelikož předpokládáme lineární závislost, je pro výpočet korelace možné použít například Pearsonův korelační koeficient:

$$\rho_{\mathbf{X},\mathbf{Y}} = \frac{\operatorname{cov}(\mathbf{X},\mathbf{Y})}{\sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}} \tag{2.1}$$

Kde $\mathbf{X},\mathbf{Y}$ jsou naše matice, $\operatorname{cov}(\mathbf{X},\mathbf{Y})$ je jejich kovariance a $\sigma_{\mathbf{X}},\sigma_{\mathbf{Y}}$ jsou jejich směrodatné odchylky.

Určování korelace pomocí této rovnice by pro nás však bylo neefektivní, jelikož my chceme znát hodnotu korelace po každém analyzovaném časovém průběhu odběru spotřeby. V případě použití této rovnice bychom však museli po každém časovém průběhu počítat korelaci znovu, aniž bychom výpočet mohli zjednodušit využitím již vypočtených hodnot. V této práci chceme časté výsledky (po každém průběhu), jelikož je chceme řádně analyzovat a následně vyhodnotit pomocí později specifikovaných metrik popsaných v sekci 2.4.

Výše popsaná rovnice se dá upravit do následující podoby určující bodový odhad [7]:

$$\mathrm{r}_{X,Y} = \frac{l\sum_{i=1}^{l} x_i y_i - \sum_{i=1}^{l} x_i \sum_{i=1}^{l} y_i}{\sqrt{l\sum_{i=1}^{l} x_i^2 - \left(\sum_{i=1}^{l} x_i\right)^2}\sqrt{l\sum_{i=1}^{l} y_i^2 - \left(\sum_{i=1}^{l} y_i\right)^2}}. \tag{2.2}$$

Kde l je počet zpracovaných časových průběhů, $r_{X,Y}$ je bodový odhad korelace mezi jednotlivými sloupci matice, x_i je i-tý řádek matice $\mathbf{X}$ a y_i je i-tý řádek matice $\mathbf{Y}$. Následně při samotném výpočtu po přidání každé nové stopy a hypotetických odhadů klíče je stačí přidat mezi průběžné součty $\sum_{i=1}^{l} x_i$ a $\sum_{i=1}^{l} y_i$, jejich druhé mocniny mezi součty $\sum_{i=1}^{l} x_i^2$ a $\sum_{i=1}^{l} y_i^2$ a jejich součin do $\sum_{i=1}^{l} x_i y_i$. Následně se výsledná korelace jednoduše vypočítá dosazením do vzorce.

Z výpočtu korelace nám vznikne korelační matice $\mathbf{Z}$ o n řádcích a 256 sloupcích, kde každá buňka $z_{i,j}$ udává, jakou mírou koreluje i-tý sloupec z matice $\mathbf{X}$ s j-tým sloupcem matice $\mathbf{Y}$.

Následně vezmeme z každého sloupce nejvyšší hodnotu korelace, čímž získáme 256 hodnot určujících maximální míru pro každou hypotézu klíče. Tento proces opakujeme, dokud výsledné hodnoty neuhádnou správný byte.

2.2.4.2 Útok pomocí šablon

Útok pomocí šablon je profilovaný druh útoku na šifrování pomocí informací získaných z postranního kanálu. Jedná se o útok založený na tom, že každý průběh spotřeby energie se dá aproximovat pomocí vícerozměrného normálního rozdělení. Toto rozdělení určíme pomocí matice kovariancí $\mathbf{C}$ a vektoru středních hodnot v. Pro obecnou matici $\mathbf{M}$ s vícerozměrným normálním rozdělením určíme matici $\mathbf{C}$ jako kovariance sloupců matice $\mathbf{M}$, takže nám vznikne čtvercová matice se stranou rovnou počtu sloupců matice $\mathbf{M}$. Vektor v obsahuje průměrné hodnoty sloupců, tudíž má délku rovnou počtu sloupců matice $\mathbf{M}$. Tento pár $(\mathbf{C}, v)$ je nazýván šablona. [3]

Metod, které používají různé hodnoty v matici $\mathbf{M}$ je více, každopádně vždy budeme útočit samostatně na jednotlivé byty klíče. Základní možnost je vzít všechny bytové páry vstupních dat a klíče, a pro každý tento pár naměřit několik průběhů spotřeby, které nám budou sloužit jako matice $\mathbf{M}$, ze které uděláme šablonu. Takto nám vznikne 256^2 šablon pro různé páry vstupních dat a klíče. Následně když útočíme na nějaký průběh, u kterého neznáme klíč, vypočítáme hustotu pravděpodobnosti pro každou šablonu s tímto průběhem. U šablony s největší hustotou pravděpodobnosti zjistíme její klíč, a získáme tak odhad klíče, který byl použitý při šifrování s tímto průběhem spotřeby. Tento postup je ovšem výpočetně velice náročný a v našem případě, kdy předpokládáme, že zařízení unikají data pomocí Hammingovy váhy, toto nemusíme podstupovat. Jelikož když budeme mít 2 hodnoty, se kterými bude pracovat obvod, a budou mít stejnou Hammingovu váhu, tak by měly mít i stejnou spotřebu, kterou my měříme. Takže šablona, která by charakterizovala průběh, ve kterém by bylo operováno s číslem stejné Hammingovy váhy jako v jiné šabloně, by poté pravděpodobně přikládala podobnou pravděpodobnost průběhu s tou samou Hammingovou váhou, na který by útočila, jako ta druhá šablona. Proto v této práci budeme vytvářet pouze 9 šablon, každá pro jinou hodnotu Hammingovy váhy. Navíc v našem případě útočíme na vnitřní hodnotu šifry AES, takže tyto šablony budou rozděleny podle Hammingovy váhy této vnitřní hodnoty.

Tudíž v naší práci budeme mít matice $\mathbf{M}_i$ pro $i \in \{0, 1, 2, \ldots, 8\}$, kde každá matice $\mathbf{M}_i$ obsahuje všechny průběhy z matice $\mathbf{X}$ s Hammingovou váhou vnitřní hodnoty rovnou i. Z těchto matic již získáme příslušné kovarianční matice $\mathbf{C}_i$ a vektory středních hodnot v_i výše popsaným postupem.

Jak je zřejmé, počet prvků matice $\mathbf{C}$ roste kvadraticky s počtem sloupců matice $\mathbf{M}$, proto chceme počet těchto sloupců ideálně redukovat. Proto zavádíme takzvané POI (*points of interest*), body zájmu. To jsou body, po kterých požadujeme, aby obsahovaly co nejvíce informací o funkci, na kterou útočíme [8]. To je v našem případě Hammingova váha vnitřní hodnoty. Pro tento účel použijeme míru rozptylu napříč jednotlivými vektory v_i v časových okamžicích. POI vybereme jako časové okamžiky, ve kterých pozorujeme největší rozptyl, který indikuje, že v tomto okamžiku by mohla probíhat operace s vnitřní hodnotou v průbězích.

Zároveň časové průběhy spotřeby energie jsou získávány pomocí osciloskopů s mnohdy vysokou vzorkovací frekvencí. To má za následek, že může vzniknout více časových bodů, které nesou totožné informace o spotřebě. Při výběru POI toto chceme ošetřit, abychom neměli v POI redundantní informace. Proto zavedeme *vzdálenost*, která udává, v jaké nejbližší vzdálenosti se mohou každé dva POI vyskytovat.

2.2.4.3 Metoda strojového učení

Jako vybranou metodu ze strojového učení jsme se rozhodli vybrat náhodný les z několika důvodů. Jelikož jako první ze zkoušených metod strojového učení, které jsme se rozhodli vyzkoušet na základě prezentované úspěšnosti v této práci [9], což byla metoda k-nejbližších sousedů, metoda rozhodovacích stromů a metoda logistické regrese, poskytoval významnější výsledky. Také proto, že je náhodný les označován za obecně známý pro svou schopnost dobře fungovat i s původními hodnotami hyperparametrů a je uveden jako první mezi používanými metodami strojového učení v analýze postranních kanálů [10, 11]. Na základě těchto okolností jsme se dohodli s vedoucím práce na použití náhodného lesa jako vybrané metody strojového učení.

Náhodný les

Metoda náhodného lesa je jeden z algoritmů strojového učení, který využívá shluk více stromových klasifikátorů, tzv. rozhodovacích stromů, pro dosažení lepšího výkonu klasifikace. V kontextu útoku na postranní kanál použijeme metodu náhodného lesa jako klasifikátor pro odhad jednoho bytu na základě naměřených dat z postranních kanálů, neboli klasifikátor, jenž bude odhadovat důležitost 256 hodnotám.

Náhodný les funguje na základě tzv. bootstrapingu, kdy z trénovacích dat vytvoří pomocí náhodného výběru s opakováním skupiny dat libovolné velikosti (zpravidla stejně velké jako původní data, ale mohou být i větší či menší) pro každý z rozhodovacích stromů. Každý tento rozhodovací strom se následně naučí na těchto datech. Když potom náhodný les dostane data, která má klasifikovat, nechá je klasifikovat každý strom samostatně, a následně se rozhoduje podle majoritní skupiny převládajících odhadů. [12]

Následující postup popisuje, jak může být metoda náhodného lesa použita pro útok na postranní kanál:

1. **Příprava základních dat:** Nejdříve připravíme matici **X** z měření postranního kanálu, která obsahuje časové průběhy proudových spotřeb a vektor odpovídajících hodnot klíče pro daný byte.

2. **Výběr příznaků:** K matici **X** dále přidáme další příznak (sloupec) bytů vstupního textu odpovídajících klíči, který chceme získat. Tento příznak byl přidán z důvodu, že tento byte vstupního textu ovlivňuje hodnotu odpovídajícího bytu vnitřní hodnoty, kterou se snaží model predikovat, tudíž má o jeden příznak více informací. Data poté rozdělíme na tréninkový a validační set.

3. **Ladění hyperparametrů:** Určíme hyperparametry, které budeme chtít ladit. To bude konkrétně `max_depth`, udávající maximální hloubku stromů, `n_estimators`, udávající počet stromů v lese, což jsou populární hyperparametry [10]. Pro jednotlivé hyperparametry určíme rozsahy, a sestavíme všechny jejich možné kombinace. Pro každou takovou kombinaci hyperparametrů na tréninkovém setu natrénujeme klasifikátor náhodného lesa. Pomocí každého takhle natrénovaného náhodného lesa predikujeme byte klíče na validačním datasetu. Následně posoudíme výkon klasifikátoru jako přesnost klasifikace, neboli procento úspěšně uhodnutých bytů.

4. **Extrahování bytu klíče:** Natrénovaný klasifikátor náhodného lesa s největší přesností klasifikace použijeme na testovacích datech popsaných v 2.4. Zde máme vždy k dispozici několik průběhů se stejným tajným klíčem. Natrénovaný klasifikátor funguje tak, že nám rovnou zobrazí nejpravděpodobnější predikci bytu pro každý průběh, ale my ho využijeme tak, že se od něj dozvíme pravděpodobnosti všech možných hodnot bytu pro každý průběh. Pokud tyto pravděpodobnosti dáme dohromady pro více průběhů se stejným tajným klíčem, očekáváme, že predikce bude konvergovat ke správnému uhodnutí klíče.

5. **Extrahování klíče:** Následně proces extrakce bytu klíče opakujeme pro všechny byty klíče, hyperparametry už necháváme stejné.

6. **Evaluace metody:** Na závěr tuto metodu vyhodnotíme pomocí metrik v sekci 2.4 a postupech v ní uvedených.

2.3 Měření časových průběhů spotřeb

Pro měření průběhů spotřeb na cílovém zařízení potřebujeme mít samotné cílové zařízení. Toto zařízení často vyžaduje externí zdroj hodinového signálu, který musíme dodat. Dále obvykle vyžaduje externí napájecí zdroj.

Pro měření elektrických signálů v čase se obvykle používá digitální vzorkovací osciloskop. Jeho nevýhoda je, že dokáže měřit pouze napětí elektrického signálu, zatímco my pro analýzu potřebujeme změřit spotřebu energie zařízení. K tomu se využívá obvod pro měření spotřeby energie, který obvykle umístí malý rezistor o odporu od 1Ω do 50Ω na napájecí zdroj cílového zařízení. Pokles napětí na tomto rezistoru je úměrný proudu v tomto zařízení, pokud uvažujeme konstantní napětí, je pokles tohoto napětí úměrný i spotřebě energie na tomto zařízení. [3]

V provedení tohoto měření se spolehneme na platformu ChipWhisperer.

2.3.1 Platforma ChipWhisperer

ChipWhisperer je open-source platforma pro výzkum postranních kanálů a provádění útoků postranním kanálem. Platforma byla vytvořena firmou NewAE Technology Inc. a zahrnuje řadu hardwarových zařízení, softwarových nástrojů a výukových materiálů pro učení a výzkum v oblasti postranních kanálů [13].

Pro práci s ChipWhisperer je k dispozici knihovna ChipWhisperer, která je napsána v jazyce Python a poskytuje rozhraní pro komunikaci s platformou, nastavení experimentů, získávání a analýzu dat a provádění útoků postranním kanálem. Knihovna také obsahuje příklady a výukové materiály pro různé kryptografické algoritmy a metody útoku postranním kanálem.

Jednotlivé desky použité v této práci budou popsány v praktické části.

2.4 Použité metriky pro porovnání

Pro důvěryhodnost evaluace jednotlivých metod je potřeba provést n sad měření. V každé sadě i naměříme data do matice $\mathbf{X}_i$. V rámci každé matice $\mathbf{X}_i$ použijeme stejný klíč, rozdílný napříč maticemi. Pro měření všech stop v průběhu měření použijeme navzájem různé vstupní texty, pro zajištění obecnosti výsledků.

2.4.1 PGE (částečná entropie odhadu)

Metoda částečné entropie odhadu poskytuje kvantitativní ukazatel toho, jak blízko jsme k získání správného klíče, což je užitečné pro hodnocení účinnosti různých útoků a pro posouzení úspěchu útoků v průběhu času. Pro výpočet částečné entropie odhadu se nejprve získají pravděpodobnosti jednotlivých možností (například v případě určování jednoho bytu klíče to bude 256 možností) na základě dostupných informací z postranních kanálů a jejich analýzy podle vybrané metody. Výsledek PGE je počet možností, které mají větší pravděpodobnost, než ta možnost, o které víme, že je správná. Čím nižší je částečná entropie odhadu, tím menší je neurčitost spojená s odhadem klíče a tím blíže jsme k získání správného klíče. V této práci budeme měřit PGE pro každý byte klíče zvlášť, jako je to obvyklé. Tudíž ideální hodnota je 0, která znamená, že jsme uhádli ten správný byte, a nejhorší je 255, kde útok upředností všechny byty před tím správným.

V praxi se částečná entropie odhadu může použít k určení, kdy je útok na postranní kanál úspěšný a kdy je potřeba zvýšit počet měření nebo použít jinou strategii pro zlepšení odhadu klíče. [14]

Při výpočtu PGE je potřeba brát v potaz, co se stane, pokud máme více bytů se stejnou hodnotou pravděpodobnosti. Pokud tato situace nastane, ale ten správný byte není mezi stejnými byty, nemá to na PGE metriku vliv. Problém nastává, pokud má ten správný byte stejnou hodnotu pravděpodobnosti, jako alespoň jeden jiný byte. V tom případě podává PGE zavádějící výsledky. Například pokud útok vyhodnotí pravděpodobnost všech bytů jako $\frac{1}{256}$, bude PGE 0, ale o správném bytu nic nevíme. Proto je v této práci PGE implementována pro tento krajní případ jako průměr pravděpodobností těchto bytů se stejnou hodnotou, pokud by jejich pravděpodobnosti byly seřazeny za sebou.

Zároveň budeme PGE vyhodnocovat u každé metody po útoku na každou další stopu (kromě začátku CPA, jak je diskutováno v sekci 3.3.1). U každé metody se pravděpodobnosti jednotlivých možností bytu klíče získávají různě. Tyto hodnoty nemusí být přímo pravděpodobnosti, ale například míra korelace. U každé je však možnost tyto hodnoty získat, a to vždy po analýze každé další stopy. Na základě toho dokážeme určit, jak se PGE mění s průběhem přidávaných stop. Toto vyhodnocení probíhá u každé sady zvlášť, a konečná hodnota se bere jako průměr z výsledků jednotlivých sad.

2.4.2 Počet nutných měření

Počet nutných měření je důležitou metrikou při hodnocení účinnosti útoků postranními kanály. Tato metrika ukazuje, kolik měření je třeba provést, aby byl útok úspěšný a klíč byl odhalen s určitou úrovní pravděpodobnosti. K provedení tohoto výpočtu potřebujeme získat údaje, které nám budou říkat u jednotlivých měření, kolik jsme potřebovali analyzovat stop, dokud jsme neuhádli správně klíč.

Zde je postup vedoucí ke spočítání počtu nutných měření:

1. Byte klíče necháme predikovat zvolenou metodou v n sadách měření. V každé sadě necháváme vyhodnotit vždy další průběh, dokud se PGE nerovná 0 (předpokládáme, že se s zvětšujícím počtem analyzovaných průběhů blíží PGE k 0, pokud ne, metoda by nebyla funkční, a museli bychom vyzkoušet jiný přístup).

2. Počet analyzovaných průběhů, než dostaneme PGE rovno 0, v sadě i, bytu k, zaznamenáme do matice $\mathbf{P}$ na místo $\mathbf{P}_{k,i}$.

3. Předchozí kroky opakujeme i s ostatními byty klíče. Tím nám vznikne matice o 16 řádcích a n sloupcích. Z této matice vytvoříme vektor p: $p = \begin{bmatrix} \max(\mathbf{P}_{:,1}) \\ \max(\mathbf{P}_{:,2}) \\ \vdots \\ \max(\mathbf{P}_{:,n}) \end{bmatrix}$, neboli jsou v něm uloženy maximální hodnoty ze sloupců matice $\mathbf{P}$. Sloupce této matice reprezentují minimální počty nutných průběhů pro každý byte klíče. Tudíž pokud vezmeme maximum, víme, že jsme s tímto počtem dokázali prolomit daný klíč.

4. Vektor p vzestupně uspořádáme, abychom ho mohli analyzovat.

5. Na závěr určíme percentil, který nám bude říkat, kolik je potřeba průběhů, abychom správně určili klíč v určitém procentu případů. Tento percentil vynásobíme s délkou vektoru p, což je n, a jeho výsledkem získáme pozici v tomto vektoru. Na této pozici leží hodnota, kterou zaokrouhlíme na celé číslo nahoru, abychom měli logicky správný výsledek, kolik potřebujeme průběhů. Tento percentil vypočítáme pro více hodnot pro lepší názornost rozložení dat, ale jako výsledný počet nutných měření budeme brát 99% percentil.

2.4.3 Délka výpočtu

Délka výpočtu představuje časovou náročnost potřebnou k provádění útoku postranním kanálem a závisí na různých faktorech, jako jsou zvolená metoda útoku, dostupné výpočetní prostředky a kvalita získaných dat. Délka výpočtu je důležitým parametrem při hodnocení efektivity útoku, jelikož úspěšný útok s kratším časem výpočtu představuje větší hrozbu pro zabezpečení šifrovaných systémů.

Délka výpočtu se povahou zásadně liší u profilovaných a neprofilovaných útoků. Profilované útoky mají dvě fáze, zatímco neprofilované mají pouze jednu. To je činí z tohoto pohledu těžko porovnatelné. Proto toto porovnání rozdělíme na tyto dvě fáze.

Při útočící fázi budeme útočit postupně na všechny byty klíče, přičemž změříme, jak dlouho bude celkem trvat tento útok. To je určeno jednak tím, jak je daná metoda rychlá v počítání pravděpodobností po každém průběhu, ale zároveň to ovlivňuje i fakt, kolik potřebuje analyzovat průběhů, aby prolomila klíč, protože obecně útočíme na byte klíče, dokud nemáme PGE rovno 0. Tudíž čím rychleji metoda určí klíč, konkrétně jeho byty, tím rychlejší bude. Celkový čas převedeme na průměrný čas pro jednu sadu, čímž získáme průměrný čas pro získání jednoho klíče.

To si musíme uvědomit hlavně u metody CPA, která by v reálném útoku fungovala jiným způsobem. V reálném útoku bychom určili počet průběhů, které budeme měřit, například podle metriky počtu nutných měření. Pro tento počet průběhů bychom rovnou spočítali korelaci u jednotlivých odhadů klíčů, a odhady s největší korelací bychom určili jako výsledný klíč. To znamená, že rychlost útoku by se podstatně změnila (v závislosti na počtu nutných měření). U metody útoku pomocí šablon a u metody náhodného lesa bychom i při reálném útoku počítali pravděpodobnosti odhadů pro všechny klíče, které následně sčítáme. Rozdíl v rychlosti by v takovém případě nastával kvůli tomu, že nemusíme po každém průběhu zjišťovat hodnotu PGE. To je časově nenáročná operace o časové složitosti $O(256) = O(1)$, neboli konstantní.

Při profilovací fázi měříme, za jak dlouho metoda vytvoří profil, pomocí kterého poté útočí. V případě útoku pomocí šablon je to doba, za jakou se vytvoří šablony, a v případě metody náhodného lesa je to doba, za jakou se natrénuje model. To budeme měřit pro všechny byty klíče dohromady. Výslednou metriku t budeme vztahovat k počtu průběhů na nichž budeme profilovat, na kterých je tato doba závislá. Tudíž vzorec pro výpočet této metriky t využijeme vzorce $t = \frac{doba\_celkova}{pocet\_prubehu}$.

Další důležitý aspekt délky výpočtu je ladění hyperparametrů u metody náhodného lesa. Tento proces obvykle trvá dlouhou dobu. Tuto hodnotu také určíme, ale nebudeme ji v této práci s ničím porovnávat.

Kapitola 3
Praktická část

V praktické části této bakalářské práce se zaměříme na aplikaci teoretických znalostí získaných v předchozích částech práce. Naše experimenty budou prováděny na platformě ChipWhisperer Nano (a později i na platformě ChipWhisperer Lite), která nám poskytne potřebné prostředky pro měření postranního kanálu a analýzu získaných dat.

3.1 Použité nástroje a knihovny

Pro úspěšné provedení experimentů a analýzu dat jsme využili následující nástroje a knihovny.

3.1.1 Python

Jako základní programovací jazyk pro implementaci algoritmů útoku postranním kanálem a analýzu dat jsme zvolili Python. Jeho snadná čitelnost, flexibilita a široká podpora knihoven nám umožnila efektivně pracovat s našimi daty a algoritmy. Především Chipwhisperer Python API, které umožňuje v Pythonu ovládat platformu Chipwhisperer, bylo hlavní motivací pro volbu Pythonu.

3.1.1.1 Numpy

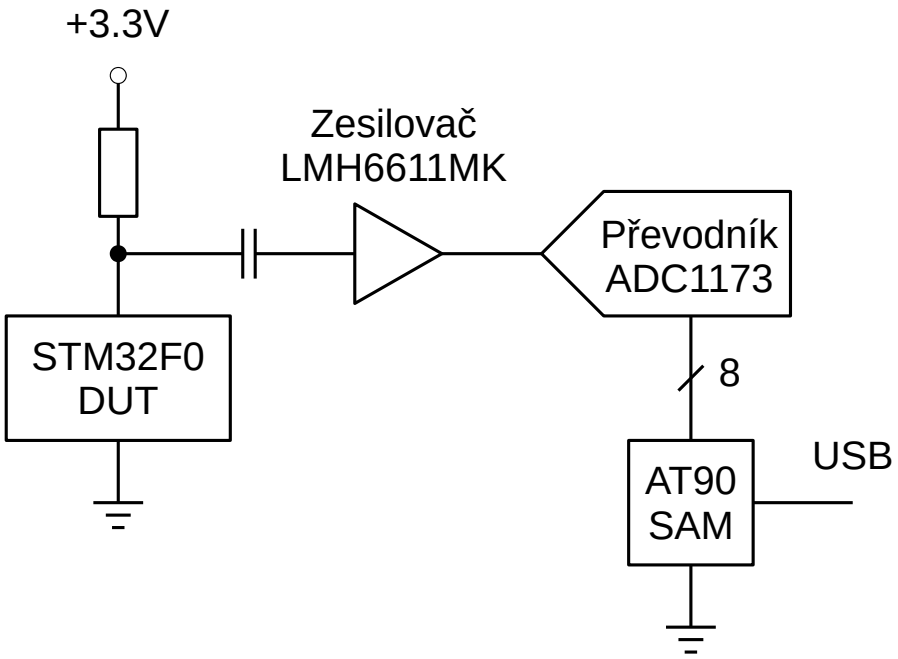
Tato knihovna představuje základní balík pro vědecké výpočty v Pythonu. Poskytuje podporu pro práci s mnohorozměrnými poli a nástroje pro lineární algebru. Numpy byla použita pro efektivní manipulaci s měřenými daty a výpočty během analýzy postranních kanálů.

3.1.1.2 Scikit-learn

Tato knihovna poskytuje jednoduché a efektivní nástroje pro analýzu dat a strojové učení. V rámci praktické části jsme využili Scikit-learn pro implementaci a vyhodnocení metod strojového učení, jako je například Náhodný les.

3.1.1.3 Matplotlib

Matplotlib je knihovna pro tvorbu vizualizací a grafů v Pythonu. V naší práci jsme ji využili pro vizualizaci a porovnání výsledků jednotlivých útoků postranním kanálem.



Obrázek 3.1 Schéma platformy ChipWhisperer Nano

3.1.1.4 SciPy

SciPy je open-source softwarová knihovna určená pro vědecké a technické výpočty v jazyce Python. V naší práci jsme ji využili primárně pro počítání vícerozměrných normálních distribucí při útoku pomocí šablon.

3.1.2 Použité desky ChipWhisperer

V rámci naší práce jsme použili ChipWhisperer Nano a ChipWhisperer Lite.

3.1.2.1 Chipwhisperer Nano

ChipWhisperer Nano je kompaktní verze ChipWhisperer platformy, která je zaměřena na zjednodušení procesu měření postranního kanálu a provádění útoků na jednoduchých mikrokontrolerech.

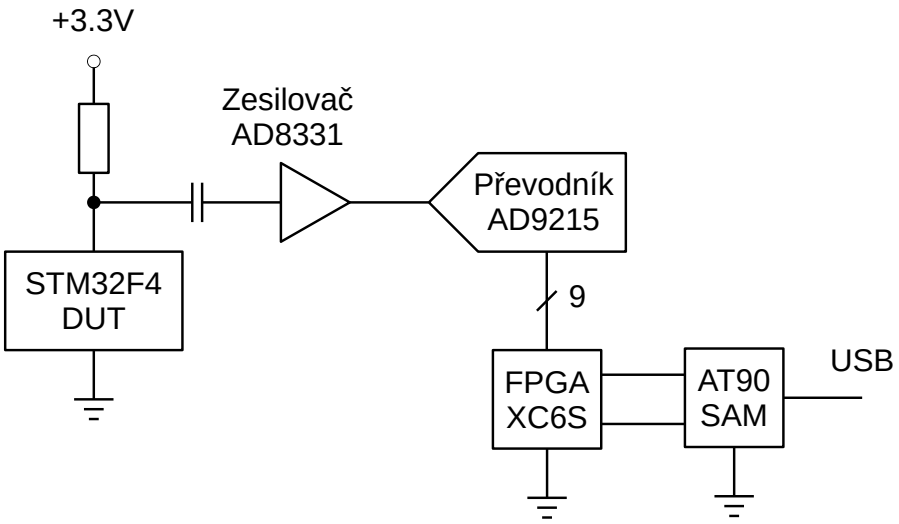
ChipWhisperer Nano se skládá z integrovaného osciloskopu pro měření napěťových odběrů, generátoru hodinových signálů pro synchronizaci měření a cílového zařízení, na kterém je implementována šifra. Platforma umožňuje měření postranních kanálů, jako jsou napěťové odběry, které vznikají během provádění kryptografických operací na cílovém zařízení.

Cílové zařízení je *STM32F0*. Schéma desky je znázorněno na obrázku 3.1.

3.1.2.2 Chipwhisperer Lite

ChipWhisperer Lite je další varianta platformy ChipWhisperer od NewAE Technology Inc., která je navržena pro pokročilejší výzkum postranních kanálů a experimenty s útoky postranním kanálem. ChipWhisperer Lite nabízí rozšířené možnosti a flexibilitu ve srovnání s ChipWhisperer Nano, což umožňuje zkoumat složitější a sofistikovanější útoky a obranné mechanismy.

ChipWhisperer Lite se skládá z odděleného osciloskopu a cílové desky. Osciloskop umožňuje vyšší přesnost měření a větší rozsah nastavení, jako je šířka pásma, vzorkovací frekvence a roz-



Obrázek 3.2 Schéma platformy ChipWhisperer Lite

lišení. Cílová deska je navržena tak, aby byla snadno vyměnitelná, což umožňuje testovat různé mikrokontrolery a kryptografické implementace.

ChipWhisperer Lite je vhodnou volbou pro pokročilejší výzkum a experimenty v oblasti postranních kanálů, když je potřeba vyšší přesnost měření, flexibilita a rozšířené možnosti nastavení.

Cílové zařízení je *STM32F4*. Schéma desky je znázorněno na obrázku 3.2.

V naší práci jsme nakonec použili tuto desku pro všechna provedená měření. A to právě z důvodu vyšší přesnosti měření, která je způsobená několika faktory. Například použití lepšího 10-bitového analogově-digitálního převodníku, který umožňuje až 105 MS/s (milionů vzorků za sekundu), oproti 8-bitovému převodníku desce Nano, který umožňuje maximálně 20 MS/s.

3.1.3 Hardware

V této části bakalářské práce se zaměříme na notebook, který byl použit jako hardwarová platforma pro měření postranních kanálů. Popíšeme specifikace notebooku relevantní v naší práci, jeho využití během výzkumu a nevýhody, které přinesl.

Specifikace notebooku:

- Procesor: Intel(R) Core(TM) i5-10300H CPU @ 2,50 GHz
- Operační paměť: 16 GB
- Úložiště: 512 GB SSD
- Operační systém: Windows 10

Využití notebooku během výzkumu:
Během výzkumu byl notebook použit pro následující účely:

1. **Akvizice dat:** Notebook byl použit k zaznamenávání a ukládání dat z měření postranních kanálů. Byl připojen k hardwarové platformě ChipWhisperer Nano a ChipWhisperer Lite a poskytoval řízení a synchronizaci měření.
2. **Analýza dat:** Notebook byl použit k analýze dat z postranních kanálů, neboli provedení všech útoků podle zmiňovaných metod.

Jak se ukázalo, použitý hardware měl svoje omezení, a to především ve velikosti operační paměti, jak bude ukázáno v sekci 3.3.3.

3.2 Proces měření

V této sekci popisujeme postup měření na platformě ChipWhisperer, který jsme využili pro získání dat potřebných pro analýzu a porovnání jednotlivých metod útoku postranním kanálem. Platforma ChipWhisperer nám poskytla nezbytné prostředky pro měření a analýzu dat z postranních kanálů. Provedení měření proběhlo následovně:

1. **Příprava prostředí:** Nejprve jsme připravili prostředí pro měření, což zahrnovalo nastavení ChipWhisperer platformy, připojení potřebných kabelů a propojení s počítačem. Zároveň jsme si připravili Python notebooky, které ovládaly ChipWhisperer pomocí ChipWhisperer Python API.

2. **Nastavení parametrů měření:** Před zahájením měření jsme nastavili parametry, které ovlivňovaly měření na platformě ChipWhisperer. Tyto parametry zahrnovaly například nastavení jakou desku budeme konkrétně používat a jakou cílovou desku budeme používat. Dále jsme nastavili parametry pro použitou šifru (AES) a nahráli potřebný firmware na cílovou desku, který implementuje šifru AES a šifrování vstupních dat pomoci ní.

3. **Měření:** Po nastavení parametrů jsme provedli samotné měření. Během měření platforma ChipWhisperer zaznamenávala data z postranního kanálu z cílové desky, čímž měřila její odběr při provádění šifrovacích operací. Počet měření závisel na požadované přesnosti výsledků a zvolené metodě útoku. Pro měření slouží funkce `capture_trace` z knihovny ChipWhisperer, které se předává kromě informací o desce a cílové desce i klíč, který bude použit na šifrování, a vstupní text, který se bude šifrovat. Funkce poté vrací už samotnou naměřenou stopu. Klíč a vstupní text jsme získávali z třídy `ktp.Basic()` z knihovny ChipWhisperer, která poskytuje klíče a vstupní texty pro tyto účely.

4. **Ukládání a zpracování dat:** Získaná data jsme uložili pomocí třídy `project` dostupné z knihovny ChipWhisperer a následně zpracovávali v Pythonu pomocí knihoven Numpy. Data byla dále analyzována a vizualizována pomocí dalších knihoven, jako jsou Scikit a Matplotlib.

5. **Opakování pro všechny sad:** Toto opakujeme pro všechny sady, které chceme naměřit.

   Měření probíhalo efektivně ve dvou fázích. Za prvé jsme potřebovali naměřit průběhy, na které budeme útočit, což jsou data která popisujeme v sekci 2.4. K tomu potřebujeme naměřit n sad měření, která obsahují k průběhů, kde v každé sadě je pro těchto k průběhů použit stejný klíč. Vstupní texty jsou vždy různé. Počet sad se ustálil na hodnotě 4000 a počet průběhů pro každou sadu na 60. K hodnotě 60 jsme došli po několika experimentech na všech modelech, kolik vlastně je potřeba průběhů maximálně, nejvíce průběhů se ukazovalo, že bude potřeba pro korelační odběrovou analýzu, podle níž jsme s určitou rezervou zvolili 60 průběhů. Reálné výsledky související s tímto výběrem jsou poté v sekci 4.1.2. Celkem jsme tedy pro útok a jeho evaluaci naměřili 240 000 průběhů. Toto měření trvalo přibližně 3,5 hodiny.

   V druhé fázi jsme potřebovali naměřit průběhy pro profilované útoky pro profilovací fázi. U těch byl vždy jiný vstupní text a jiný klíč a měří se vlastně pouze jedna sada, jelikož ke všem průběhům je přistupováno stejně. Pro tyto účely bylo naměřeno 300 000 průběhů. Toto měření trvalo přibližně 6 hodin. Jak jde vidět, měření těchto průběhů trvalo nepoměrně déle, než měření v první fázi. Náš odhad je, že implementace AES na cílovém zařízení si ukládá výsledek expanze klíče, tudíž když je použit znova, nemusí ho znovu expandovat. Naopak když vždy dostane nový klíč, musí ho vždy nejdříve expandovat, což může zapříčinit toto zpomalení. Odpověď na tuto otázku může být předmětem zkoumání v jiné práci.

3.3 Realizace jednotlivých metod útoku

Měření časových průběhů proudových spotřeb probíhalo na platformě ChipWhisperer. Nejdříve jsme měřili na platformě ChipWhisperer Nano, ovšem později bylo rozhodnuto po domluvě s vedoucím této práce, že všechna měření budou probíhat na platformě ChipWhisperer Lite, a to z důvodů nekonzistence měření průběhů mezi jednotlivými relacemi měření, které měly za důsledek nemožnost realizace metody útoku pomocí metody strojového učení, jak je popsáno v sekci 3.3.3. Po zjištění této skutečnosti bylo tedy všechno měření prováděno pomocí platformy ChipWhisperer Lite, tudíž i všechny výsledky jsou na základě těchto měření. Pro to jsme se rozhodli z toho důvodu, abychom všechny výsledky mohli mezi sebou porovnávat.

3.3.1 Realizace korelační odběrové analýzy (CPA)

Pro korelační oděrovou analýzu jsme implementovali rovnici určující bodový odhad korelace. Počítali jsme korelaci mezi bodovými okamžiky průběhů a Hammingovou váhou odhadu klíčů. Z toho vyplývá, že tuto metodu jsme nemohli použít pouze na jeden průběh, jelikož bychom počítali korelaci vždy mezi dvěma samostatnými čísly. Ovšem jak jsme zjistili, nemohli jsme počítat korelaci ani pro dva či tři průběhy. Důvodem bylo, že vzniklé korelační matice měly příliš mnoho nedefinovaných hodnot, což nám znemožňovalo je analyzovat a vyhodnotit nejpravěpodobnější klíč. Tyto nedefinované hodnoty se tvoří na základě dělení nulou ve jmenovateli rovnice, což se u takhle málo čísel objevuje dle našeho pozorování statisticky častěji, než když je jich více. Matematický aparát pro odůvodnění tohoto pozorování jsme v této práci nevyvinuli.

Poté, co jsme vypočítali korelační koeficienty pro daný byte klíče, provedli jsme samotnou analýzu. Pro každý nově přidaný průběh jsme vypočítali odhady spotřeb všech možných hodnot bytu klíče (256 hodnot). Tyto odhady spotřeby jsme porovnali s příslušnými průběhy a vypočítali pro ně korelační koeficienty. Tento proces jsme opakovali pro další průběhy, dokud nebyla maximální korelace u správného bytu. Tento proces se opakoval pro všechny byty klíče.

3.3.2 Realizace útoku pomocí šablon

Pro realizaci útoku pomocí šablon jsme sestavili třídu v Pythonu s názvem `POI`. Tato třída má na starosti analýzu specifikovaného projektu. Zahrnuje rozdělování průběhů do skupin podle Hammingovy váhy vnitřní hodnoty jednotlivých bytů klíče, čímž vytváří matice $\mathbf{M}_i$. Na základě těchto matic poté počítá vektory středních hodnot a kovarianční matice, které tvoří jednotlivé šablony. Tato třída tedy zajišťuje celou profilovací fázi.

V průběhu samotného útoku, po sestavení šablon pro daný byte klíče, jsme pro každý nově přidaný průběh sestavovali odhady spotřeb pro všechny možné hodnoty bytu klíče, tedy 256 hodnot. Pro odhad spotřeby jsme využili příslušnou šablonu a vytvořili z ní vícerozměrné normální rozdělení pomocí třídy `scipy.stats.multivariate_normal` z knihovny `scipy`. Následně jsme vypočítali hustotu pravděpodobnosti pro příslušný odhad spotřeby a tento výpočet přidali k sumě pravděpodobností pro jednotlivé hodnoty bytu klíče. Tento proces jsme opakovali pro další průběhy, až dokud PGE nebyla rovna 0. Celý proces se opakoval pro všechny byty klíče.

V průběhu našeho experimentování jsme se snažili nastavit vzdálenost mezi POI, jak bylo popsáno v teoretické části, avšak jakákoliv vzdálenost jiná než 0 vedla k horším výsledkům. S různým počtem POI jsme experimentovali a nejlepší výsledky jsme získali pro 20 POI.

Narazili jsme na problém s délkou trvání útoku. Zatímco profilovací fáze trvala přibližně 90 sekund pro 300 000 průběhů, útočící fáze trvala 17 minut pro 4000 sad po 60 průbězích pro každý byte. Tento problém jsme řešili implementací cache, která si ukládá již určené třídy `multivariate_normal`. Tyto třídy pak počítají hustotu pravděpodobnosti pro další průběhy. Předpokládáme, že tato třída ukládá nějaké mezivýpočty nutné pro výpočet této hodnoty, které

může použít i pro pozdější výpočty hustot pravděpodobností dalších průběhů. Ačkoliv naše teorie není potvrzena, zavedení tohoto typu cachování nám umožnilo zrychlit útočící fázi 16,6krát.

3.3.3 Realizace metody náhodného lesa

Pro realizaci metody útoku pomocí náhodného lesa byla použita knihovna `scikit-learn` a z ní konkrétně třída `sklearn.ensemble.RandomForestClassifier`, která implementuje klasifikátor náhodného lesa.

Metodu náhodného lesa jsme zkoušeli jako druhou ze zde probíraných metod, hned po metodě CPA, která dosahovala přesvědčivých výsledků. Naproti tomu zde metoda náhodného lesa měla nekonzistentní výsledky. Když byly naměřeny průběhy na ChipWhisperer Nano, které se rozdělily na trénovací a validační, byl model schopný dosáhnout určitých ne nevýznamných výsledků na validačních datech. Problém nastával v nasazení tohoto modelu na predikci klíče u průběhů, které byly naměřeny v jakémkoli jiném sezení. V takovém případě byl tento model nepoužitelný a jeho úspěšnost (v odhadování bytu klíče, jak bylo zmíněno zde 3) se pohybovala od 0,2 do 0,4 %. Jelikož klasifikujeme 256 hodnot, pravděpodobnost, že se náhodně trefíme do té správné hodnoty je $\frac{1}{256} \doteq 0,0039$. Z toho vyplývá, že na těchto jiných datech měl až horší úspěšnost než kdybychom prováděli náhodnou predikci. Následně jsme tuto situaci zkoušeli odůvodnit různými hypotézami, jako například, že se při ladění hyperparametrů tento model nakonec přeučil ku prospěchu validačních dat. Proto jsme vyzkoušeli brát testovací data ze stejného sezení, ovšem úspěšnost na těchto testovacích datech byla velice podobná té na validačních, tudíž přizpůsobení validačním datům neprobíhalo. V rámci řešení tohoto problému jsme vyzkoušeli i různé postupy normalizace příznaků za účelem vyhlazení rozdílů mezi měřeními v různých sezeních. Tyto úpravy nepomohly, naopak měl model horší úspěšnost v rámci i jednoho sezení. Důvodem této nekonzistence bylo nakonec nedostatečně přesné měření průběhů způsobené platformou ChipWhisperer Nano. Z tohoto důvodu jsme se rozhodli provádět všechna následující měření na platformě ChipWhisperer Lite.

Při trénování modelů bylo zjištěno, že je zapotřebí velké množství stop, aby byly predikce naučených modelů účinné. Zároveň se však se zvětšujícím počtem průběhů zvětšuje i časová a prostorová složitost jak učení modelů, tak získávání dat. Nakonec se počet průběhů z časových důvodů ustálil na hodnotě 300 000, které se měřily přibližně 6 hodin, jak bylo diskutováno v 3.2. Na těchto datech byly laděny hyperparametry, a nakonec i naučen výsledný model.

Při vyladění hyperparametrů modelu do takové úrovně, že byl tento model schopný predikovat testovací data s úspěšností až 20 %, ukázaly výsledky funkce `feature_importances_` (významnosti příznaků), dostupné u třídy použité k sestavení modelu náhodného lesa, že vliv takto přidávaného bytu vstupního textu na učení modelu není významný. Proto byl pro pozdější učení tento příznak odebrán. Zároveň tento objev vedl k rozhodnutí provést výběr příznaků, z nichž se bude model učit. Byl zvolen způsob filtrační metody, který v principu vyhazuje příznaky, jež mají nízký rozptyl a jsou tedy téměř konstantní. Zde byl určen nízký rozptyl mezi průměry skupin stop rozdělených podle Hammingovy váhy vnitřní hodnoty daného průběhu, což eliminuje ty příznaky, které jsou méně závislé na vnitřní hodnotě, a tudíž je zde předpoklad, že vnitřní hodnotu nebudou významněji ovlivňovat. Tento postup byl popsán jako výběr POI v sekci 2.2.4.2, a pojí se s ním i výběr minimální vzdálenosti mezi jednotlivými POI, který byl přidán jako další parametr k ladění. Kolik příznaků zachovat bylo také přidáno jako parametr k ladění. Ukázalo se, že graf hodnot variance těchto skupin průběhů a graf významnosti příznaků jsou si navzájem velmi podobné, což ukazuje, jakým způsobem si model dokáže vybrat příznaky sám (lze vidět v grafu 3.4). Tato oční inspekce byla podpořena i výpočtem korelace mezi těmito daty, která dosáhla hodnoty 0,7, což už můžeme označit za silnější korelaci. Ovšem pokud proběhne výběr předem, je model účinější, jeho účinnost na testovacích datech stoupne až k 50 %, a z důvodu menšího počtu příznaků probíhá učení daleko kratší dobu.

3.3.3.1 Ladění hyperparametrů

Jako hyperparametry pro model náhodného lesa jsme tedy zvolili `max_depth` a `n_estimators`, jak bylo popsáno v sekci 3. K tomu jsme chtěli zjistit, jaké budou nejlepší hodnoty počtu vybraným POI, a jakou stanovit délku mezi nimi. K tomu jsme využili `model_selection.ParameterGrid` z `sklearn`, který slouží pro sestavení všech různých kombinací hyperparametrů.

Při ladění jsme narazili na různé problémy, ten hlavní byl problém s operační pamětí. Ladění se z tohoto důvodu několikrát zastavilo, jelikož programu došla paměť, kterou by mohl alokovat. Stávalo se to při tréninku modelů s větší maximální hloubkou a větším počtem stromů v lese, které mají velký vliv na velikost modelu. Tento problém jsme vyřešili zavedením vymazání modelu vždy na konci iterace pomocí `del`, a manuální aktivace `garbage collectoru`, který sbírá všechny vymazané objekty z paměti. V takovém nastavení již ladění hyperparametrů mohlo proběhnout. Ladění probíhalo na našem hardwaru 20 hodin.

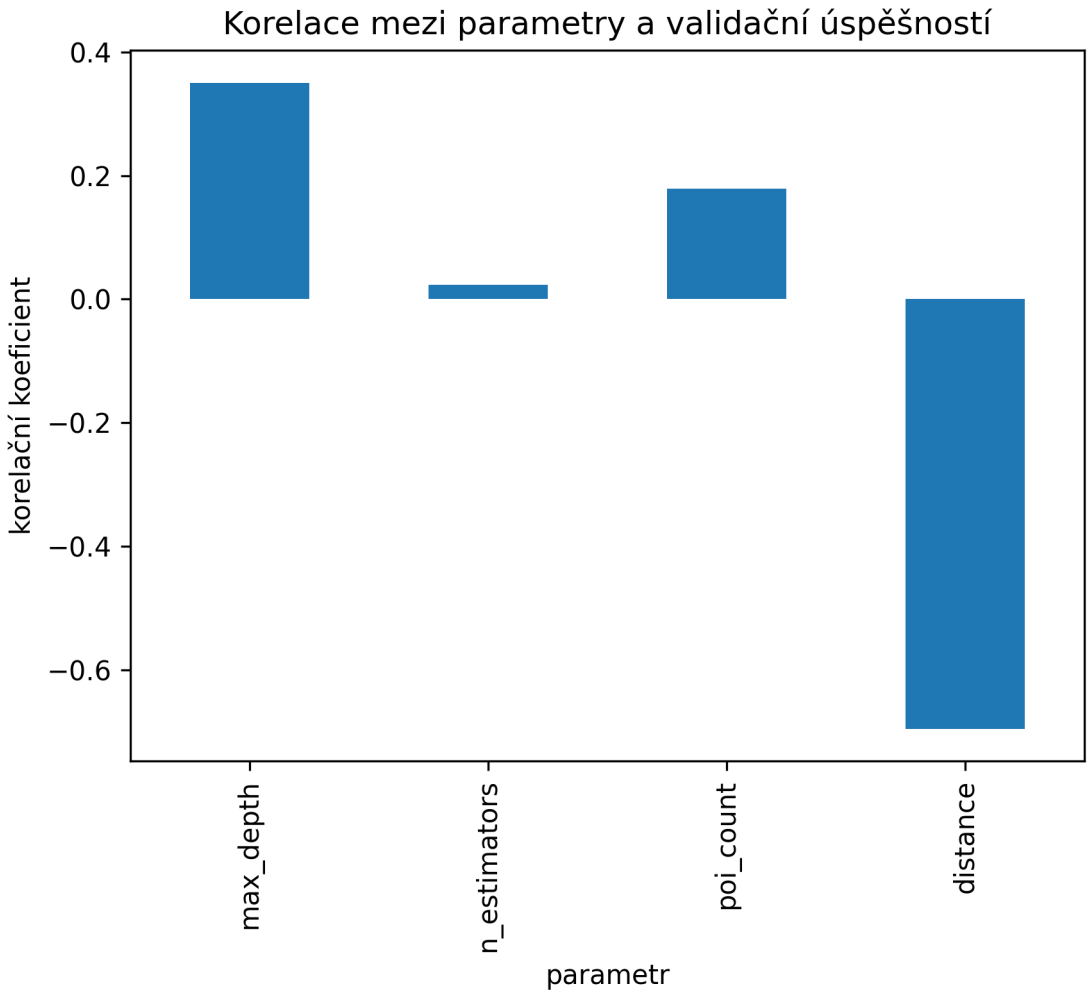
Modelem s největším úspěchem byl nakonec náhodný les s hodnotami hyperparametrů v tabulce 3.1. V případě `max_depth` a `n_estimators` to byla největší hodnota z rozsahu. Zároveň jsme změřili korelaci mezi jednotlivými parametry a hodnotou úspěšnosti modelu na validačních datech, jak lze vidět v grafu 3.3, nejdůležitějším parametrem je maximální hloubka, zatímco největší zápornou korelaci má vzdálenost, tudíž nejlepších výsledků dosahoval model když se vzdálenost vůbec nepoužívala, neboli měla nulovou hodnotu. Poté byl celkem důležitý parametr počtu POI, zatímco počet lesů měl korelaci blízkou nule, takže jeho hodnota byla pro výsledný model nejméně důležitá.

Hyperparametr	**Hodnota**
`max_depth`	16
`n_estimators`	225
POI	25
vdálenost	0

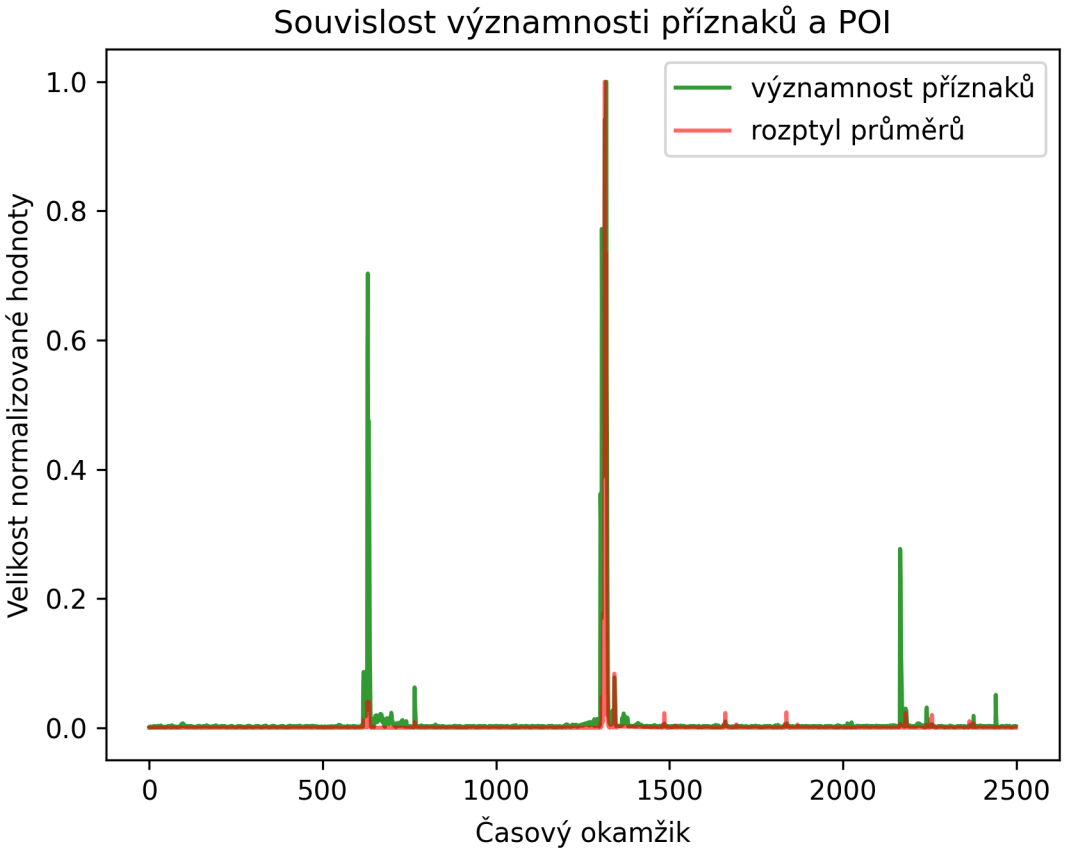
Tabulka 3.1 Výsledná nejlepší kombinace hyperparametrů

Problém nastal opět u paměti. Když jsme se pokusili zvětšit hodnotu `max_depth`, nebylo již možné z důvodu nedostatku paměti natrénovat lepší model. Proto jsme nakonec zůstali u těchto hyperparametrů jako konečných. Když jsme tedy tento model znova natrénovali, abychom ho mohli evaluovat na námi zvolených metrikách, a zároveň si ho uložit, zjistili jsme, že po uložení má na disku 18 GB. Tento model slouží jako klasifikátor pro jeden byte. Kdybychom tedy chtěli uložit všechny modely, abychom pomocí nich mohli po načtení určit celý klíč, potřebovali bychom na disku prostor přibližně 282 GB.

Zároveň učení takového modelu trvalo přibližně 6 min a 23 s. To nebyla velká časová zátěž, proto jsme se rozhodli modely nijak neukládat, ale nechat je naučit se vždy před samotným útokem.



Obrázek 3.3 Korelace mezi jednotlivými parametry a úspěšností modelu na validačních datech



Obrázek 3.4 Souvislost významnosti příznaků s POI, které jsou vybírány podle rozptylu mezi skupinami podle Hammingovy váhy vnitřní hodnoty. U obou pozorování jsme provedli normalizaci hodnot do intervalu $[0, 1]$, abychom je mohli porovnat

Kapitola 4
Výsledky

4.1 Porovnání účinnosti metod útoku

4.1.1 PGE

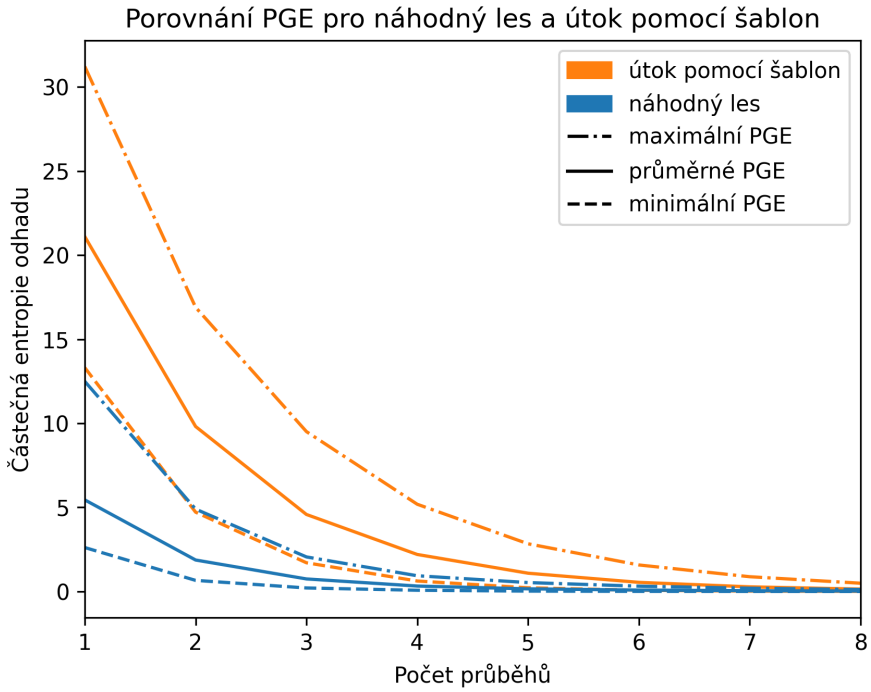
Metriku PGE jsme v určitých variantách naměřili u všech metod útoků. Nebyli jsme schopni ji naměřit u útoku CPA v nižších rozsazích naměřených stop (3 průběhy a méně), z důvodu limitace samotné metody a limitace použité rovnice, tak jak bylo diskutováno v 3.3.1. U ostatních dvou metod, které jsou založeny na předpřípravě modelu, jsme mohli měřit PGE i s jednou stopou, a to poměrně úspěšně. Nejúčinější byla v tomto ohledu měření PGE pomocí jedné stopy metoda strojového učení, Náhodný les. Poté byla tedy metoda šablon, jak lze vidět v tabulce 4.1.

Metoda	**PGE**
Útok pomocí šablon	21,1
Náhodný les	5,45

Tabulka 4.1 Výsledky PGE na jedné stopě pro jednotlivé metody

Poté jsme měřili závislost hodnot PGE na počtu zpracovaných průběhů. V případě CPA jsme pro tyto účely použili rovnici 2.2, která nám umožňuje získávat mezivýsledky, aniž bychom museli vždy celý výsledek přepočítat. Jak již bylo zmíněno, toto měření u CPA proběhlo až od čtyř stop. Proto, a také proto, že zbývající dvě metody jsou profilované, a předpokládají se od nich odlišné výsledky, porovnáváme v grafu 4.1 tyto dvě, útok pomocí šablon a náhodný les. Zde jsou použity tři indikátory k zachycení průběhu křivek PGE pro jednotlivé byty, a to maximální PGE, průměrné PGE a minimální PGE u jednotlivých metod. Lze vidět, že náhodný les si vedl podstatně lépe než útok pomocí šablon. Jeho maximální křivka, která určuje nejhorší výsledek napříč byty, se překrývá s minimální křivkou útoku pomocí šablon, dokonce pro první průběh má nižší PGE.

Porovnání s CPA lze vidět v grafu 4.2, kde jak již bylo řečeno, CPA začíná až od 4. průběhu. Nicméně lze pozorovat, že výsledky profilovaných a neprofilovaných útoků jsou diametrálně odlišné.



■ **Obrázek 4.1** Porovnání maximální, průměrné a minimální hodnoty metriky PGE u náhodného lesa a útoku pomocí šablon. Tyto profilované metody byly profilovány na 300 000 průbězích. Počet průběhů v grafu na ose x vyjadřuje počet průběhů ve fázi samotného útoku

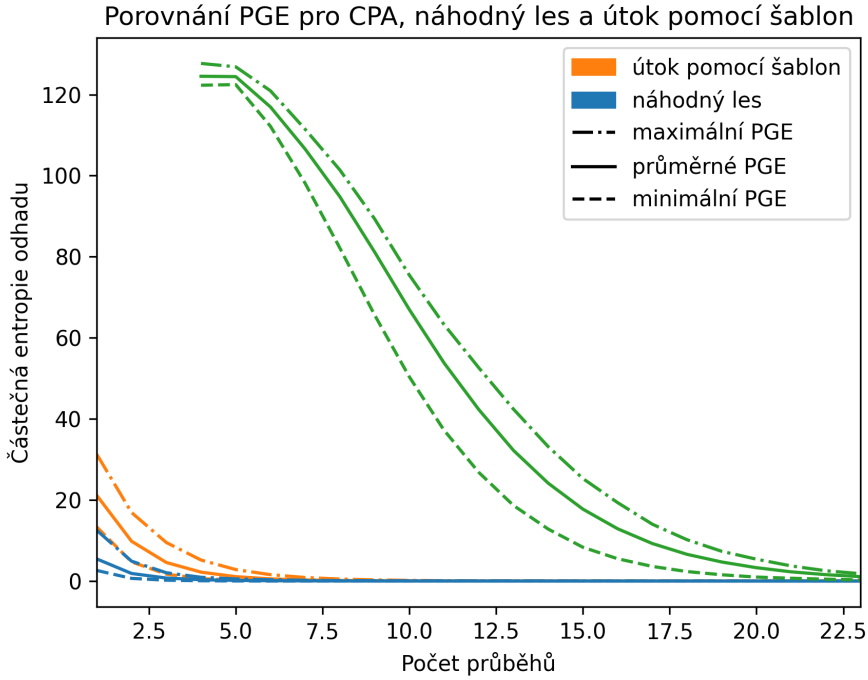
4.1.2 Počet nutných měření

Nutný počet průběhů nám říká, kolik potřebujeme u daného útoku průběhů, abychom měli určitou jistotu, že pomocí něho získáme klíč. Tudíž počítáme pro jakou hodnotu dosáhneme námi požadovaný percentil. Výsledek určujeme pomocí více percentilů, pro větší míru představy o rozložení dat na základě těchto výsledků. K výpočtu této metriky nám slouží sada dat udávající u každého modelu, kolik bylo u každého klíče nutno naměřit průběhů. Výsledky pro různé percentily jsou v tabulce 4.2. Podobných výsledků dosáhly metody útok pomocí šablon a náhodný les. U nich potřebujeme analyzovat 17 průběhů, abychom měli 99% jistotu uhádnutí klíče, zatímco u CPA to bude 32. Zde se opět ukazuje, že rozdíl mezi profilovanými a neprofilovanými útoky je veliký. Zatímco profilované útoky nemají tak veliký rozdíl, u každého percentilu je jejich vzájemné pořadí různé.

■ **Tabulka 4.2** Srovnání metod útoku podle nutného počtu průběhů

Percentil \ Metoda	CPA	Útok pomocí šablon	Náhodný les
95%	27	14	12
99%	32	17	17
99,9%	39	21	27

Z těchto dat můžeme získat i histogram 4.3, který o něco lépe graficky znázorní úspěšnost



Obrázek 4.2 Porovnání maximální, průměrné a minimální hodnoty metriky PGE u CPA, náhodného lesa a útoku pomocí šablon. Zobrazené profilované metody (náhodný les a útok pomocí šablon) byly profilovány na 300 000 průbězích. Počet průběhů v grafu na ose x vyjadřuje počet průběhů ve fázi samotného útoku

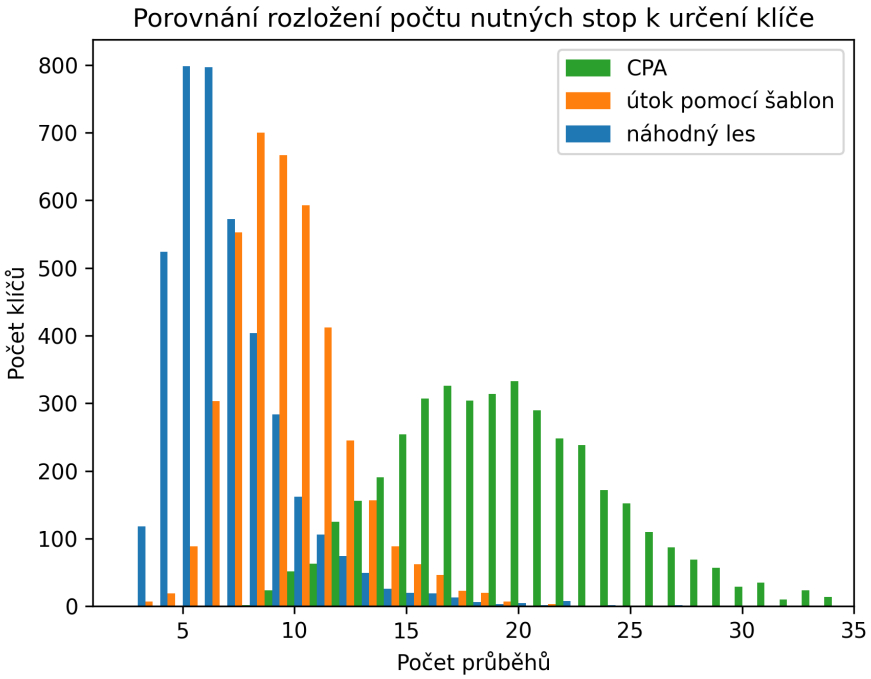
jednotlivých modelů v rámci této metriky. V tomto histogramu můžeme pozorovat rozložení dat udávajících, kolik bylo nutno u každého klíče naměřit průběhů, abychom ho uhádli, tudíž čím je rozložení obecně více vlevo, tím lépe. Zde lze vidět, že náhodný les má nejpříznivější rozložení soustředěné nejvíce z metod k začátku. Poté je na tom podobně útok pomocí šablon a na konci je metoda CPA.

4.1.3 Délka výpočtu

Délku výpočtu jsme určovali zvlášť pro útočící fázi a profilovací fázi. Útočící fázi mají všechny metody, které jsme v této práci vyzkoušeli. Tuto hodnotu jsme určili jako průměrný čas k prolomení jednoho klíče. Výsledky lze vidět v tabulce 4.3, ve které vidíme, že nejlépe si vedl útok pomocí šablon, který v průměru prolomil klíč přibližně 28krát rychleji, než náhodný les. To i přes fakt, že náhodný les potřebuje v průměru méně průběhů k prolomení klíče, jak vyplývá z předchozí metriky počtu nutných měření. Domníváme se, že útok pomocí náhodného lesa je zpomalen z důvodu nedostatku paměti. Jak by si tento útok vedl při dostatečném množství operační paměti, může být námět pro budoucí práce. Korelační odběrová analýza je přibližně 2krát rychlejší než náhodný les a 14krát pomalejší než útok pomocí šablon. CPA je zpomalena průběžných vypočítáváním korelací, jak jsme diskutovali v teoretické části.

V profilovací fázi jsme měřili dobu nutnou na vytvoření profilu v přepočtu na jeden průběh. Výsledky jsou v tabulce 4.4. Zde lze vidět, že profilování útoku pomocí šablon bylo rychlejší než profilování útoku pomocí náhodného lesa. Tento rozdíl není tak signifikantní jako v útočící fázi.

Poté jsme měřili dobu strávenou laděním hyperparametrů, což probíhalo pouze u náhodného lesa. Je to hodnota, kterou musíme brát v potaz u metod strojového učení, a v závislosti na přesnosti jakou chceme získat pro ideální kombinaci hyperparametrů může ladění probíhat velice



Obrázek 4.3 Porovnání nutného počtu průběhů k určení klíče u všech použitých metod

Metoda	**Čas [s]**
Korelační odběrová analýza (CPA)	3,475
Útok pomocí šablon (template attack)	0,245
Metoda strojového učení (náhodný les)	6,962

Tabulka 4.3 Výsledky průměrné doby potřebné k prolomení jednoho klíče při samotném útoku pro jednotlivé metody

dlouho v našem případě to bylo 20 hodin, s tím, že by model potřeboval ještě přesnější ladění, ale v tom nás už limitovala velikost paměti.

Metoda	**Čas [ns]**
Útok pomocí šablon (template attack)	4,727
Metoda strojového učení (náhodný les)	20,427

Tabulka 4.4 Výsledky délky výpočtu při profilování v přepočtu na jeden průběh pro jednotlivé metody

Kapitola 5
Závěr

5.1 Závěr

V rámci této bakalářské práce jsme se zaměřili na problematiku šifrování a postranních kanálových útoků. Cílem práce bylo provést měření postranního kanálu na platformě ChipWhisperer Nano a porovnat vybrané metody útoku postranním kanálem, jako jsou korelační odběrová analýza (CPA), útok pomocí šablon (template attack) a vybranou metodu strojového učení, konkrétně metodu náhodný les.

V teoretické části jsme se zabývali základními koncepty šifrování a jeho významem v ochraně informací. Dále jsme se seznámili s vybranou šifrou a platformou ChipWhisperer. Podrobně jsme prozkoumali jednotlivé metody útoku postranním kanálem a metriky, které jsme použili k jejich porovnání.

V praktické části jsme se věnovali návrhu a implementaci experimentů na platformě ChipWhisperer. Pomocí jazyka Python jsme realizovali jednotlivé metody útoku a zhodnotili jsme jejich účinnost na základě zvolených metrik.

Výsledky naší práce ukazují, že každá metoda má svoje specifické využití. Při útoku pomocí CPA potřebujeme naměřit všechna data již na samotném přípravku, na který útočíme. To může být omezujícím, pokud máme například omezený čas na měření, nebo omezený počet měření. Samotná analýza nasbíraných dat je poté průměrně rychlá a nejméně účinná. Potřebujeme 32 průběhů na prolomení klíče.

Útok pomocí šablon umožňuje naměřit data, podle kterých vytvoříme danou šablonu, na jiném přípravku, než na který útočíme. Zde musíme mít plný přístup, a dopředu znát všechny klíče. Následně není potřeba měřit takové množství dat na cílovém přípravku, jako u CPA. Celkově je tento útok nejrychlejší v porovnání jak profilovací fáze, tak útočící fáze. Pro prolomení klíče potřebujeme analyzovat 17 průběhů.

Útok pomocí metody strojového učení, v této práci konkrétně zvoleného náhodného lesa, umožňuje přistupovat k problému SCA šifry jako k problému strojového učení. Ve výsledku se tato metoda projevila jako srovnatelná z pohledu počtu nutných průběhů, která musíme naměřit na cílovém přípravku, abychom prolomili daný klíč, s útokem pomocí šablon. Tomu předchází dlouhé učení a optimalizace jednotlivých modelů, společně s nutností naměřit velké množství dat, aby měly modely takovou úspěšnost. Zároveň se ukázalo, že aby bylo možné využít tuto metodu, bylo zapotřebí použít přesnějšího měřícího přípravku než ChipWhisperer Nano, jelikož jeho výsledky nebyly natolik konzistentní mezi měřeními v různých podmínkách, aby se na nich dal vybudovat úspěšný model.

Tato zjištění přispívají k lepšímu porozumění účinnosti jednotlivých metod útoku postranním kanálem a mohou sloužit jako základ pro další výzkum v oblasti zabezpečení šifrování.

V rámci této práce jsme identifikovali několik omezení a oblastí pro budoucí výzkum. Všechna

měření probíhala na zařízeních, na kterých nebyly použity obranné mechanismy proti útokům postranními kanály. Účinnost námi použitých metod by v takovém případě byla odlišná.

Budoucí výzkum by se mohl zaměřit na testování metod SCA proti možným výše zmíněným ochranám. Zároveň by se mohl zaměřit na návrh a implementaci nových ochran a opatření pro ochranu proti SCA.

Shrnutím dosažených výsledků a zkušeností získaných při řešení této bakalářské práce můžeme konstatovat, že jsme úspěšně splnili stanovené cíle a přispěli k lepšímu pochopení problematiky postranních kanálových útoků a jejich efektivity v kontextu zabezpečení šifrování.

Bibliografie

1. DWORKIN, Morris; BARKER, Elaine; NECHVATAL, James; FOTI, James; BASSHAM, Lawrence; ROBACK, E.; DRAY, James. *Advanced Encryption Standard (AES)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards a Technology, Gaithersburg, MD, 2001. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.197.
2. KOCHER, Paul; JAFFE, Joshua; JUN, Benjamin. Differential Power Analysis. In: WIENER, Michael (ed.). *Advances in Cryptology — CRYPTO' 99*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, s. 388–397. ISBN 978-3-540-48405-9.
3. MANGARD, Stefan; OSWALD, Elisabeth; POPP, Thomas. *Power analysis attacks*. Springer New York, NY, 2007. Dostupné také z: https://link.springer.com/book/10.1007/978-0-387-38162-6.
4. STANDAERT, François-Xavier; KOEUNE, François; SCHINDLER, Werner. How to Compare Profiled Side-Channel Attacks? In: ABDALLA, Michel; POINTCHEVAL, David; FOUQUE, Pierre-Alain; VERGNAUD, Damien (ed.). *Applied Cryptography and Network Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 485–498. ISBN 978-3-642-01957-9.
5. BRIER, Eric; CLAVIER, Christophe; OLIVIER, Francis. Correlation Power Analysis with a Leakage Model. In: JOYE, Marc; QUISQUATER, Jean-Jacques (ed.). *Cryptographic Hardware and Embedded Systems - CHES 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 16–29. ISBN 978-3-540-28632-5.
6. CHARI, Suresh; RAO, Josyula R.; ROHATGI, Pankaj. Template Attacks. In: KALISKI, Burton S.; KOÇ, çetin K.; PAAR, Christof (ed.). *Cryptographic Hardware and Embedded Systems - CHES 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, s. 13–28. ISBN 978-3-540-36400-9.
7. SOCHA, Petr; MIŠKOVSKÝ, Vojtěch; KUBÁTOVÁ, Hana; NOVOTNÝ, Martin. Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches. In: *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2017, s. 184–189. Dostupné z DOI: 10.1109/DDECS.2017.7934563.
8. CAGLI, Eleonora. *Feature Extraction for Side-Channel Attacks*. 2018. Dostupné také z: https://theses.hal.science/tel-02494260. Theses. Sorbonne Université.
9. EDMONDS, Jack; MOON, Tyler. *Machine Learning-Based Side-Channel Analysis on the Advanced Encryption Standard*. 2021. Dostupné také z: https://scholarcommons.scu.edu/elec_senior/61. Dis. pr.

10. TRITHIPKAIWANPON, Thitiya; TAETRAGOOL, Unchalisa. Sensitivity Analysis of Random Forest Hyperparameters. In: *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2021, s. 1163–1167. Dostupné z DOI: 10.1109/ECTI-CON51831.2021.9454885.

11. SONG, Shijie; CHEN, Kaiyan; MA, Jiawei. An overview of profiling side channel analysis based on machine learning. *IOP Conference Series: Materials Science and Engineering*. 2019, roč. 563, č. 3, s. 032029. Dostupné z DOI: 10.1088/1757-899X/563/3/032029.

12. BREIMAN, Leo. Sv. 45. Springer Science a Business Media LLC, 2001. Č. 1. Dostupné z DOI: 10.1023/a:1010933404324.

13. 2020. Dostupné také z: https://www.newae.com/chipwhisperer.

14. STANDAERT, François-Xavier; MALKIN, Tal G.; YUNG, Moti. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: JOUX, Antoine (ed.). *Advances in Cryptology - EUROCRYPT 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 443–461. ISBN 978-3-642-01001-9.