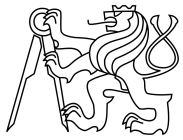
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vliv obrany hardwarové implementace AES vůči fault-injection útokům na její odolnost před útoky rozdílovou odběrovou analýzou
Student:	Martin Mašek
Vedoucí:	Dr.-Ing. Martin Novotný
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Na FPGA implementujte vybrané metody obrany vůči fault-injection útokům pro šifru AES. Prozkoumejte vliv těchto obran na odolnost vůči útokům rozdílovou odběrovou analýzou. Porovnejte odolnost hardwarové implementace šifry AES s obranami a bez obran. Konkrétní cílovou platformu vyberte po konzultaci s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. října 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalářská práce

Vliv obrany hardwarové implementace AES vůči fault-injection útokům na její odolnost před útoky rozdílovou odběrovou analýzou

Martin Mašek

Vedoucí práce: Dr.-Ing. Martin Novotný

15. února 2018

Poděkování

Chtěl bych poděkovat Dr.-Ing. Martinu Novotnému za námět při výběru tématu, jeho cenné rady a odborný dohled při vypracování bakalářské práce. Také děkuji Mgr. Lucii Baumrukové za pomoc při gramatické kontrole práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. února 2018 .....

Odkaz na tuto práci

Mašek, Martin. *Vliv obrany hardwarové implementace AES vůči fault-injection útokům na její odolnost před útoky rozdílovou odběrovou analýzou.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Bakalářská práce se zabývá vlivem obrany vůči fault-injection útokům na šifrovací algoritmus AES implementovaný na FPGA Spartan 3E. Je prozkoumáno, zda tyto obvody pro detekci chyb mají nějaký dopad na odolnost před útoky rozdílovou odběrovou analýzou. V rámci práce byla vytvořena hardwarová implementace AES v jazyce VHDL a program ve skriptovacím jazyku Matlab pro provedí DPA útoku. Z tohoto základního návrhu bylo odvozeno dalších 5 variant obsahujících informační redundanci pro detekci chyb v průběhu chodu programu. U každé varianty byl zjištěn počet průběhů spotřeby, při kterých se úspěšně prolomí šifra a získá šifrový klíč. Výsledky neodhalily nějaký vliv na proveditelnost útoku.

Klíčová slova AES, rozdílová odběrová analýza, DPA, FPGA, fault-injection, obrana vůči útokům

Abstract

The bachelor thesis deals with the inluence of prevention against fault-injection attacks on the AES cryptographic algorithm implemented on the Spartan 3E FPGA. It is investigated whether these fault detection circuits have any impact on resistance to differential power analysis attacks. In the course of the thesis, hardware implementation of AES in VHDL was developed and a program was written for DPA attack in the Matlab scripting language. Additional 5 variants were derived from the primary desing containing information redundancy for error detection during runtime. The number of patterns of consumption was found to successfully break the cipher and obtain the cipher key for each variation. The results did not reveal any greater impact on the feasibility of the attack.

Keywords AES, differential power analysis, DPA, FPGA, fault-injection, defense against attacks

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době je čím dál důležitější počítačová bezpečnost. Zatímco dříve byl problém utajení a šifrování dat jen doménou odborníků, dnes se touto otázkou musí zabývat i koncový uživatel. Poté, co se internet rozšířil do celého světa a někteří jedinci si bez něj nedokáží představit být byť jen jediný den, stal se tento problém velmi důležitým, ne-li jedním z hlavních diskutovaných témat informatiky.

Kryptografické systémy musí zajistit utajení informací před neoprávněnými uživateli, zaručit autentičnost dat, aby příjemce zprávy byl schopný ověřit jejího odesílatele a obstarat datovou integritu, zda zpráva přišla v původním znění a nebyla po cestě nijak modifikována. Tomu mají dopomoci stále složitější a komplexnější šifrovací systémy a algoritmy v nich použité. Analýzou těchto algoritmů se zjišťuje jejich bezpečnost proti případným útokům.

Dosavadní metody kryptoanalýzy se převážně soustředily na odhalení slabiny v matematické podstatě, např. frekvenční analýzou. Prolomit algoritmus tímto způsobem v rozumném čase se stává velice obtížné, a proto se do popředí kryptoanalýzy dostávají útoky, které se snaží využít informací, které unikají přímo z fyzické implementace systému během běhu kryptografického algoritmu, např. skrze postranní kanály, což je souhrnný název pro jakoukoliv výměnu informací mezi zabezpečeným systémem a jeho okolím.

V této práci se budu zabývat rozdílovou odběrovou analýzou (differential power analysis, DPA), při které se sleduje spotřeba energie v zařízení, kdy se využívá toho, že tato spotřeba je přímo závislá na právě prováděné operaci. Cílem této analýzy je zjistit použitý šifrový klíč na základě velkého počtu naměřených proudových spotřeb, které byly v konstantním čase zaznamenány pro průběh operace šifrování (nebo dešifrování) pro různá vstupní data.

Cílem této práce je zjistit, zda se změní odolnost hardwarové implementace šifry Advanced Encryption Standard (AES) v programovatelném hradlovém poli (FPGA) pokud šifru doplníme o protiopatření proti poruchám, které jsou založeny na většinou jednoduchém přidání nějaké redundantní informace pro odhalení chyb v přenosu dat. Cílem je také zjistit, jestli tato opatření

budou mít vliv na efektivnost odběrové analýzy a zda bude takto upravený kryptografický systém poté více zranitelný.

Práce je rozdělena do několika kapitol. Úvodní kapitola slouží jako seznámení s algoritmem AES, jeho popisem a k jakým operacím v něm dochází. Tato kapitola dále obsahuje sekci o typech postranních kanálů a zvláště pak analýzu proudového kanálu. Součástí jsou i informace, co je diferenciální proudová analýza a jak v ní využít tzv. korelační koeficient. V závěru úvodní kapitoly je i soupis použitých variant obran vůči fault-injection útokům.

Druhá kapitola se zabývá analýzou použité desky FPGA a způsobem, jakým byl proveden útok DPA. Součástí je i analýza obran a jejich následné zabudování do implementace algoritmu. Třetí kapitola pak detailně popisuje realizaci celého systému a obsahuje vybrané části VHDL kódu. Součástí jsou i bloková schémata jednotlivých částí a stavové diagramy konečných automatů v řadičích systému. Následuje kapitola o testování jednotlivých částí návrhu a ověření jejich korektního fungování. V poslední kapitole je popsán průběh DPA a porovnání odolnosti jednotlivých variant.

Kapitola 1

Úvod do problematiky

1.1 Algoritmus AES

AES je jedním z nejpoužívanějších šifrovacích algoritmů, kterým se mimo jiné zabezpečují Wi-Fi sítě. V roce 2001 byl americkým Národním Institutem Standardů a Technologií (NIST)[1] zvolen za šifrovací standard a nahradil tím v té době již nevyhovující DES. Bloková šifra, která se v tomto algoritmu používá, se jmenuje Rijndael, podle svých autorů Joana Daemena a Vincenta Rijmena. Detailní popis lze najít třeba v [2] nebo [3]. Velikost zpracovávaného bloku je pevně daná na 128 bitů (16 bytů), avšak délka klíče se může zvolit varianta se 128, 192 nebo 256 bity. Při použití klíče o 128 bitech se šifruje v deseti rundách (iteracích). Každá z rund obsahuje 4 operace - SubBytes, ShiftRows, MixColumns a AddRoundKey. AddRoundKey je použito ještě před první rundou a naopak poslendí runda neobsahuje MixColumns. Schéma rund je uvedeno na obrázku 1.1. 16 bytů vstupního řetězce je rozděleno do matice 4x4 (tabulka 1.1), kterou nazýváme stavem či stavovou maticí. Každá operace má svou odpovídající inverzní operaci, která se používá při dešifrování.

Tabulka 1.1: Stavová matice

A_0	A_1	A_2	A_3
A_4	A_5	A_6	A_7
A_8	A_9	A_{10}	A_{11}
A_{12}	A_{13}	A_{14}	A_{15}

1.1.1 SubBytes

SubBytes je jediná nelineární transformace prováděná během šifrování. Každý byte vstupního řetězce je nahrazen jiným podle předem definované vyhledávací tabulky, tzv. S-Box. Je to nejdůležitější krok celého výpočtu, který zajišťuje bezpečnost proti případným útokům.

1.1.2 ShiftRows

Byty stavové matice se cyklicky posouvají po řádcích doleva. Každý řádek se posouvá o různý počet pozic. První zůstává nezměněn, druhý o jednu, třetí o dvě a čtvrtý o tři pozice. Pro představu slouží tabulka 1.2.

Tabulka 1.2: Stavová matice po posunu řádků

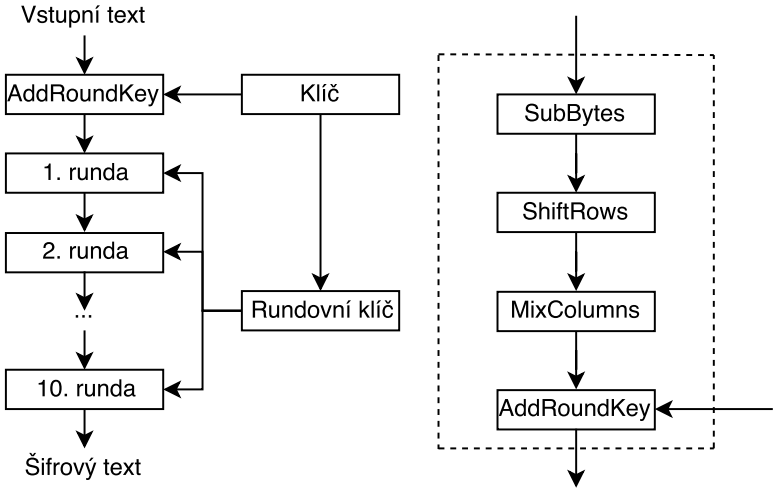
A_0	A_1	A_2	A_3	žádný posun
A_5	A_6	A_7	A_4	posun o 1 doleva
A_{10}	A_{11}	A_8	A_9	posun o 2 doleva
A_{15}	A_{12}	A_{13}	A_{14}	posun o 3 doleva

1.1.3 MixColumns

Předposledním krokem každé rundy je MixColumns, který zaručuje dostatečnou náhodnost ve výpočtu. Postupně se vezme každý sloupec stavové matice, které se pomocí speciálního polynomu mezi sebou pronásobí, přičemž se počítá v tělese $GF(2^8)$. Výsledné byty jsou tedy mezi sebou ovlivněny těmi vstupními.

1.1.4 AddRoundKey

Nejjednodušší ze všech použitých operací. Bitová operace xor mezi 16 byty stavu a k nim odpovídajících 16 bytů expandovaného klíče. Pro každou rundu se v KeySchedulde odvodí jiný rundovní klíč z klíče vstupního, přičemž v první rundě se použije právě ten vstupní. Popis KeySchedulde najdete v [4].



Obrázek 1.1: Vlevo schéma algoritmu AES, vpravo schéma jedné rundy

1.2 Postranní kanály

Koncem 20. století se dospělo k závěru, že moderní kryptografické algoritmy jsou založeny na pevných matematických základech [5], které by konvenčním způsobem kryptoanalýzy měly být neprolomitelné [6]. Proto se upustilo od hledání slabin v matematickém modelu a více se zaměřilo na problémy implementační - tedy jakým způsobem je šifra napsána a jak jsou jednotlivé části poskládány za sebou. Hlavně jde ale o využití informací, které vyzařují z fyzické implementace systému, tedy například elektromagnetické záření takového zařízení. Posléze se objevily způsoby, jak tímto postupem odhalit klíč a zprávu rozluštit. Postranní kanál je definován jako nežádoucí výměna informací mezi kryptografickým modulem a jeho okolím [7].

Analýza postranních kanálů z hlediska získaných dat se rozděluje na jednoduchou a diferenciální [8]. Pokud jsou data zpracována jen prostým pozorováním a vyhodnocena bez použití speciálních výpočetních metod, jedná se o analýzu jednoduchou (Simple Analysis, SA). Pro úspěšné prolomení šifry pomocí takovéto analýzy je pro útočníka většinou nutné znát detailní znalost implementovaného algoritmu a kryptografického zařízení. Při útoku je nutné, aby v zařízení, na které je prováděn útok, existovala závislost mezi tajným klíčem a proudovou spotřebou. Typickým příkladem je útok na implementace asymetrického šifrovacího algoritmu RSA (Rivest Shamir Adleman)[9], u kterého lze pozorovat rozdíl ve spotřebě proudu mezi operacemi násobení a umocnění.

Jestliže však použijeme určité statistické a matematické techniky a pokud si dokážeme celý výpočet určitým způsobem zautomatizovat, lze v takovém případě mluvit o analýze diferenciální (Differential Analysis, DA). Cílem je získat tajný šifrovací klíč na základě velkého počtu naměřených hodnot, které byly zaznamenány během procesu šifrování pro různá vstupní data. Hlavní výhodou je pak to, že útočník nemusí mít příliš velkou znalost o zařízení a šifrovacím algoritmu, stačí mu pouze informace o tom, který algoritmus byl použit.

Druhů postranních kanálů je hned několik. Každý typ je založen na konkrétní měřitelné informaci, kterou většinou bývá nějaká fyzikální veličina, jenž jsme schopni změřit.

Typy postranních kanálů jsou (převzato z [10]):

- elektromagnetický (electromagnetic) – v důsledku průchodu proudu obvodem vzniká v jeho okolí změřitelné střídavé elektromagnetické pole,
- výkonový nebo proudový (power) – velikost energie spotřebované v systému je přímo závislá na druhu právě prováděné operace,
- časový (timing) – založen na předpokladu, že doba výpočtu je úměrná délce vstupních dat a délce použitého klíče,

- chybový (fault) – záměrné posílání špatných dat a využití chybových hlášení,
- optický (optical) – pokaždé, když tranzistor v obvodu změní svou hodnotu, vyzáří do okolí několik fotonů, které se dají posléze změřit,
- akustický (acoustic) – byl zjištěn například u klávesnice na bankomatu, kdy lze podle nahrání zvuku tlačítek zpětně zjistit PIN kód,
- a další.

1.3 Proudová analýza

Většina moderních zařízení mají jako základní stavební prvek logická hradla, která jsou tvořena tranzistory. Elektrony proudí v obvodu, spotřebovávají energii a produkují elektromagnetické záření. Spotřeba takové energie se dá změřit, například osciloskopem. Poprvé byla proudová analýza představena v roce 1999 P. Kocherem v [7].

1.3.1 Jednoduchá proudová analýza

Jednoduchá proudová analýza (SPA, Simple power analysis) je technika, při níž se naměřená data spotřeby přímo interpretují během měření. Lze tím třeba zjistit, jaké operace se právě provádějí a do jakého místa bychom měli směřovat náš útok. Velmi náchylná na tento typ analýzy je například šifra DES. Nevýhodou SPA je nutná detailní znalost použitého algoritmu a kryptografického zařízení.

1.3.2 Diferenciální proudová analýza

Diferenciální proudová analýza (DPA, Differential power analysis) je populárnější typ proudové analýzy. Není u ní nutné detailně znát daný kryptografický modul, je jen postačující vědět, který je použit. Nalezení šifrovacího klíče je založeno na změření velkého počtu změn průběhů při šifrování různých vstupních datových bloků. Klíč může být odhalen i v případě zaznamenání průběhů s vysokým šumem. Pro úspěšné provedení DPA je nutné fyzicky vlastnit kryptografické zařízení během provádění analýzy a vědět jaká vstupní nebo výstupní data byla v daný okamžik použita.

Obecné schéma DPA je následující podle [11]. Nejprve se zvolí vstupní hodnoty (tzv. vnitřní hodnota), tedy známá vstupní data a neznámý šifrovací klíč, který chci analýzou vyluštit.

Poté se změří (např. osciloskopem) průběh proudové spotřeby při šifrování (či dešifrování, záleží na zvoleném algoritmu) těchto vstupních hodnot a výsledky se zaznamenají.

Následně se vytvoří tabulka hypotéz, jakých hodnot by mohl klíč nabývat (pro jeden byte je to 0-255). K tomu se musí určit závislost mezi proudovou spotřebou a vnitřní hodnotou.

Nakonec se vezmou naměřené hodnoty průběhů s těmi hypotetickými a statisticky se porovnají. Tam, kde vyjde největší závislost (korelace), najdeme hodnotu klíče.

1.4 Korelační koeficient

Korelační koeficient (Correlation coefficient) patří mezi nejpoužívanější metody, jak změřit závilost mezi dvěma náhodnými proměnnými, je tedy vhodný pro útok DPA [12]. Pro dvě veličiny X a Y je definován jako:

$$\rho = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\sigma_x^2 \sigma_y^2}} \tag{1.1}$$

kde $\mathrm{Cov}(X,Y)$ označuje kovarianci, tedy střední hodnotu součinu odchylek obou náhodných veličin od jejich středních hodnot, a σ_x^2 a σ_y^2 označují rozptyl těchto veličin. Korelace pak může nabývat hodnot od -1 do 1. -1 značí nepřímou závislost, 1 značí přímou závislost a 0 pak značí, že měřené hodnoty jsou na sobě nezávislé, tedy neexistuje statisticky zjistitelná závislost.

V DPA je korelační koeficient použit k určení lineární závislosti mezi dvěma sloupci dvou matic (sloupec h_i s indexy $i = 1, ..., K$ a sloupec t_j s indexy $j = 1, ..., T$), výsledkem je matice, jejíž prvky $r_{i,j}$ jsou vypočítány podle následujícího vztahu:

$$r_{i,j} = \frac{\sum_{d=1}^{n}(h_{d,i} - \overline{h_i})(t_{d,i} - \overline{t_j})}{\sqrt{\sum_{i=1}^{n}(h_{d,i} - \overline{h_i})^2 . \sum_{i=1}^{n}(t_{d,i} - \overline{t_j})^2}} \tag{1.2}$$

Matici H odpovídá matice s vytvořenými hypotézami, která má n řádků (počet naměřených průběhů) a 256 sloupců, pro každou možnou hodnotu bytu jeden. Matice T obsahuje změřené průběhy. Nakonec hodnoty $\overline{h_i}$ a $\overline{t_j}$ značí průměrné hodnoty příslušných sloupců.

1.5 Obrana vůči fault-injection útokům

Útoky postranními kanály jsou dnes vážným problémem při implementaci kryptografických algoritmů. V dnešní době již bylo vyvinuto mnoho opatření proti těmto útokům, avšak stále existuje velmi málo informací o tom, jestli když ochráníme zařízení proti jednomu typu útoku, jaký to bude mít vliv na útok jiným postranním kanálem. V této práci se hlavně soustředíme na zranitelnost vůči fault-injection útokům [13], tedy že do algoritmu jsou přidány systémy na detekci chyb, a jaký to má dopad na efektivnost útoku založeném na DPA.

Obrany jsou založeny na většinou jednoduchém přidání nějaké redundantní informace pro odhalení chyb v přenosu dat [14]. Podle [15] a [16] jsme se v práci zaměřili na 5 takových opatření. Nejjednodušší je přidání jednoho bitu parity, poté dvojitá parita, komplementární parita, kontrola zbytku modulo 3 a Hammingův opravný kód.

- Jednoduchá parita je jeden bit, který se přidá za každý byte výpočtu, jenž udává informaci o počtu jedničkových bitů, pokud se jedná o sudou paritu, či naopak počet nul v bytu pro paritu lichou. V tomto případě se zaměříme na paritu sudou. Například tedy pro byte zapsaný ve dvojkové soustavě jako 01101011 je parita rovna 1.
- Dvojitá parita je v tomto případě opět sudá, ale místo jednoho přidaného bitu to jsou dva. V jednom je uložena informace o počtu jedničkových bitů na lichých pozicích bytu a v tom druhém na sudých. Tedy pro 01101011 to je 10.
- Komplementární parita je také reprezentována dvěma bity a je to společně sudá i lichá parita celého bytu. Pro náš byte 01101011 tedy vychází 01.
- Kontrolu zbytku modulo 3, jak už název napovídá, je výpočet zbytku po dělení třemi. Tedy výsledkem je číslo 0, 1 nebo 2. Pokud tedy náš ukázkový byte 01101011 přepíšeme do desítkové soustavy na 107, tak modulo 3 vychází 2.
- Poslední použitou metodou je výpočet Hammingova kódu. Rozšířený Hammingův kód (8,4) je schopen detekce dvou chyb a opravení jedné chyby. Výpočet probíhá jak je znázorněno v [17] a výsledkem jsou 4 paritní bity. Pro náš byte tedy vychází 0101.

Kapitola 2

Analýza

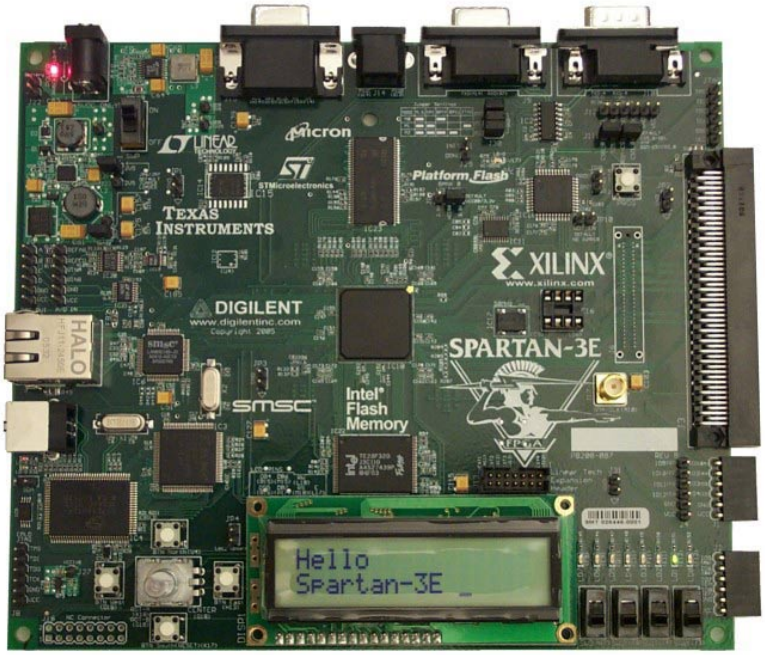
2.1 Výběr FPGA desky

Pro útok pomocí DPA na FPGA je na výběr hned několik platforem. Pro účely této práce byla zvolena deska Digilent Spartan 3E Starter Board (obrázek 2.1). Tato deska byla zvolena především z toho důvodu, že na stejném přístroji prováděli měření i kolegové Ondřej Semerád[18], Lukáš Mazur[19] a Jan Severyn[20], což umožňuje porovnání výsledků této práce s jejich výsledky a nebo na ně přímo navázat. Navíc už byla deska příslušně upravena pro tento druh měření a tedy bylo i ověřeno, že útok je na ní možný provést s odpovídajícími výsledky. Jednou z hlavním výhod byla také přítomnost jumperů pro snadné připojení osciloskopu a také, že deska obsahuje rozhraní RS232 pro komunikaci po sériové lince. Dále byly z desky odstraněny blokovací kondenzátory, které jsou jinak připojené k napájecím pinům. Z důvodů redukce rušení ze síťového adaptéru byl zdroj napětí vyměněn za olověný akumulátor. Místo diferenciální sondy pro měření osciloskopem byl použit po konzultaci s Priv.-Doz. Dr. Amir Moradi AC předzesilovač PA 303 BNC od firmy Langer EMV-Technik, který má zesílení 30 dB.

2.2 Metody útoku rozdílovou odběrovou analýzou

Existují dva přístupy k tomu, jak vytvořit tabulku hypotéz v DPA, a to buď užitím Hammingovy vzdálenosti nebo Hammingových vah. Jak bylo zjištěno, každá varianta se hodí na jiné použití. Při útoku na čipovou kartu je vhodným kandidátem použití Hammingovy váhy a cílit útok na první rundu šifrování. Hammingova váha nám udává počet jedničkových bitů v bytu.

Tento přístup se ukázal jeho nevhodný při pokusu o prolomení šifry AES v poslední rundě šifrování. Bylo zjištěno, že k lepším výsledkům při útoku na FPGA se dojde při použití Hammingovy vzdálenosti, která nám udává rozdíl mezi dvěma hodnotami. Pokud se tedy zaměříme na změnu hodnoty



Obrázek 2.1: Deska Spartan 3E Starter Kit

registru v čase, můžeme přesněji určit změny proudu, které v tu chvíli nastávají [12].

Čistě z praktických důvodů byla zvolena runda poslední, kdy potřebujeme dvě hodnoty v registru, před a po rundě. Hodnotu po poslední rundě máme k dispozici - jde o samotný šifrový text. K hodnotě před poslední rundou se dojde díky inverzním operacím samotného šifrování, které se jinak využívají v dešifrování. Inverzní funkce k AddRoundKey je to samé, protože operace xor je inverzní sama k sobě. V inverzní ShiftRows se otočí směr posunu řádků. Inverze v SubBytes je výpočet podobné tabulky hodnot (provádí se zde hledání inverze hodnoty v tělese $GF(2^8)$ a afinní transformace ve stejném tělese). Jednodušší způsob zjištění inverze pro S-Box je následující - pokud na políčku 41_{16} (řádek 4. sloupec 6.) je v tabulce uloženo číslo 83_{16}, tak v inverzní tabulce bude na políčku 83_{16} (řádek 8. sloupec 3.) uloženo číslo 41_{16}.

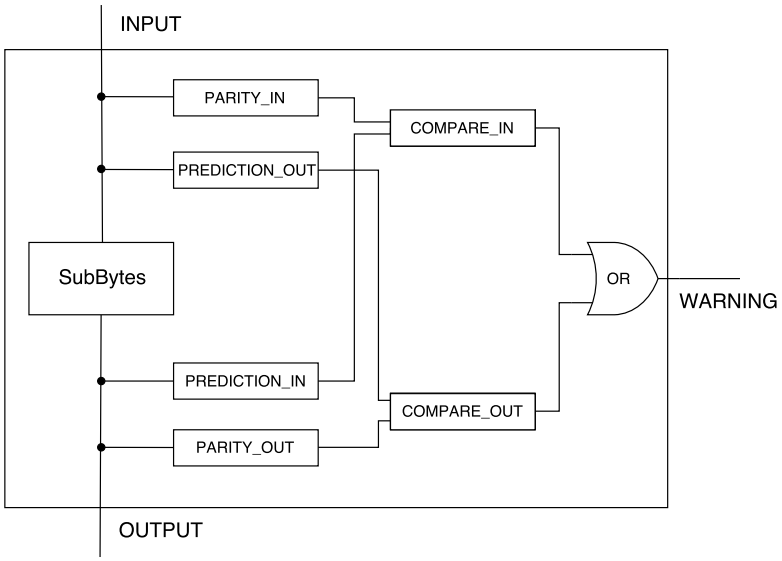
Útoky diferenciální proudovou analýzou se v případě blokových šifer, což AES bloková šifra je, koncentrují na část výpočtu, kdy se v SubBytes mění hodnota přes S-Box. Výstup z této nelineární transformace je místo, kde je rozdíl mezi správnými a špatnými hodnotami hypotéz klíče je největší [16].

2.3 Fault-injection obrany

Obrana proti chybám je založena na přidání nějaké dodatečné informační redundance, tedy bitů navíc do obvodu. Stejně jako kolega Zimmerhakl [21] jsme se v této práci inspirovali v [22]. Hlavní myšlenkou je, že se do celého obvodu a paměti nepřidává žádný drát (signál) navíc pro paritu, nýbrž parita se spočítá až těsně před vstupem do SubBytes.

Do SubBytes se přidají dodatečné obvody pro predikci parity. Ze vstupního bytu se vždy vypočte vstupní parita `PARITY_IN` a také predikce, jakou paritu by měl mít výstupní SubBytes byte, což vypočítá `PREDICTION_OUT`. Po výměně v SubBytes je vypočtena parita, jaká opravdu z funkce vyšla, tedy `PARITA_OUT`, a také predikce hodnoty, která by pro tento byte měla být vstupní, tedy `PREDICTION_IN`. Predikce se určují z tabulek, které jsou předem předpočítány a uloženy v paměti.

Příslušné vstupní a výstupní parity a jejich odpovídající predikce jsou v `COMPARE_IN`/`COMPARE_OUT` pomocí operace xor porovnány a pokud jeden či druhý zjistí změnu, vyšle jedničku do výstupního signálu `WARNING`. Výše uvedené je přehledně znázorněno na obrázku 2.2.



Obrázek 2.2: Obecné schéma pro výpočet parity v SubBytes operaci

Kapitola 3

Realizace

V průběhu práce jsme vytvořili dvě implementace šifry AES a další programy pro metodu DPA. První implementace je v jazyce C a je určena pro čipovou kartu. Tento způsob útoku je na FIT ČVUT vyučován v rámci předmětu MI-BHW[23] a sloužil v této práci čistě jako seznámení se se základy šifrování dat a následného prolamování použitých šifer. Jakým způsobem probíhá DPA na čipové kartě se lze podívat ke kolegům Mazurovi[19], Říhovi[24] a Semrádovi[18].

Druhá stěžejní implementace AES je napsána v jazyce VHDL a je určena pro použití v FPGA. Nejprve jsme vytvořili základní verzi, do které pak byly přidány varianty obrany proti chybám za účelem jejich zranitelnosti proti DPA.

Poslední důležitou komponentou této práce bylo vytvoření programu v jazyce Matlab, který provádí analýzu naměřených dat.

3.1 AES pro FPGA

V této kapitole detailně popíšeme implementovanou šifru AES.

3.1.1 Vstupní rozhraní

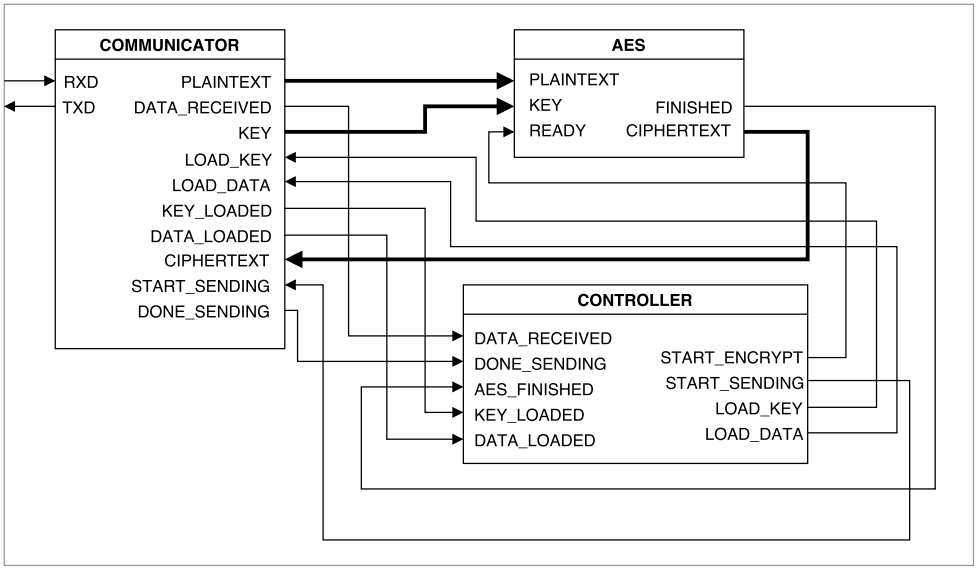
Kód 3.1: Rozhraní TOP

```
entity TOP is
    port (
        CLK     : in  STD_LOGIC;
        RST     : in  STD_LOGIC;
        RXD     : in  STD_LOGIC;
        TXD     : out STD_LOGIC;
        TRIG    : out STD_LOGIC
    );
```

```
end TOP;
```

Vstupní rozhraní s názvem `TOP` má jako vstupní signál hodin `CLK`, jejichž frekvence je pro zlepšení útoku snížena předděličkou 32x, tedy z 50 MHz na 1,5625 MHz. Další vstupní signál je reset, připojený ke tlačítku na desce a signály `RXD` a `TXD` pro komunikace po sériové lince. Výstupní signál `TRIG` využíváme v měření na osciliskopu a značí začátek šifrování.

Do této entity jsou připojeny celkem tři komponenty. `COMMUNICATOR` pro komunikaci na sériové lince, `AES`, ve které najdeme samotnou implementaci šifry, rozdělenou na datovou a řídící cestu, a nakonec `CONTROLLER`, jenž generuje řídící signály pro celé rozhraní. Celá implementace je propojena signálem hodin a resetu, aby všechny registry a čítače pracovaly správně. Obrázek 3.1 znázorňuje zapojení jednotlivých komponent.



Obrázek 3.1: Blokové schéma entity TOP

3.1.2 Rozhraní pro sérivou linku

Kód 3.2: Rozhraní komunikátoru

```
entity COMMUNICATOR is
    port (
        CLK             : in  STD_LOGIC;
        RESET           : in  STD_LOGIC;
        RXD             : in  STD_LOGIC;
        PLAINTEXT       : out STD_LOGIC_VECTOR (0 to 127);
```

```
        KEY             : out STD_LOGIC_VECTOR (0 to 127);
        DATA_RECEIVED   : out STD_LOGIC;
        LOAD_KEY        : in  STD_LOGIC;
        LOAD_DATA       : in  STD_LOGIC;
        KEY_LOADED      : out STD_LOGIC;
        DATA_LOADED     : out STD_LOGIC;
        TXD             : out STD_LOGIC;
        CIPHERTEXT      : in  STD_LOGIC_VECTOR (0 to 127);
        START_SENDING   : in  STD_LOGIC;
        DONE_SENDING    : out STD_LOGIC
    );
end COMMUNICATOR;
```

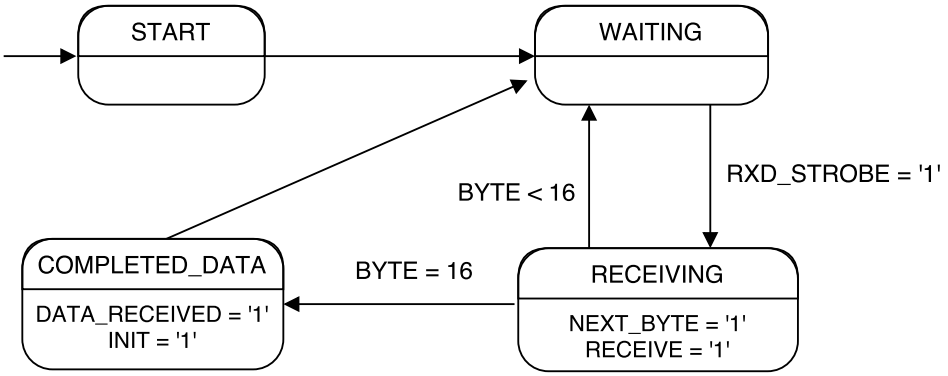
Entita `COMMUNICATOR` slouží pro komunikaci po sériové lince. Její součástí je komponenta `RS232`, která je převzata od Dr.-Ing. Martina Novotného ze stránek [23], a další dvě komponenty pro čtení dat ze sériové linky a zápis na ní (`RECEIVE_DATA` a `TRANSMIT_DATA`). Obě komponenty obsahují jeden konečný automat typu Moore se třemi procesy (stavový registr, kombinační logiku generující následující stav a kombinační logiku na generování výstupu podle aktuálního stavu) a poté i datový registr, do kterého jsou postupně nahrávána buď přijatá data nebo data čekající na odeslání. Nakonec je přítomen ještě čítač aktuálního přijímaného/odesílaného bytu, jelikož po sériové lince můžeme najednou poslat jen 8 bitů. Stavové diagramy pro přijímaní a odeslání dat jsou uvedeny na obrázcích 3.2 a 3.3.

V entitě jsou ještě dva procesy, které slouží pro rozlišení, zda přijatá data jsou klíč nebo už data určená pro šifrování. Tuto obsluhu zajišťují dva vstupní signály (`LOAD_KEY` a `LOAD_DATA`) a dva výstupní signály (`KEY_LOADED` a `DATA_LOADED`). Signál `LOAD_...` nám řekne, jaký typ dat se nyní přijal a kam je máme uložit a signálem `..._LOADED` dáváme odpověď, že se tak stalo.

Protože se při realizaci této práce stávalo, že po resetu se přes sériovou linku poslal první byte neznámých dat, je v `RECEIVE_DATA` čítač nastaven na čítání od 1 do 16, první byte se ignoruje. v `TRANSMIT_DATA` tato skutečnost nenastala, tedy čítač čítá 0 do 15.

3.1.2.1 Přijímání dat

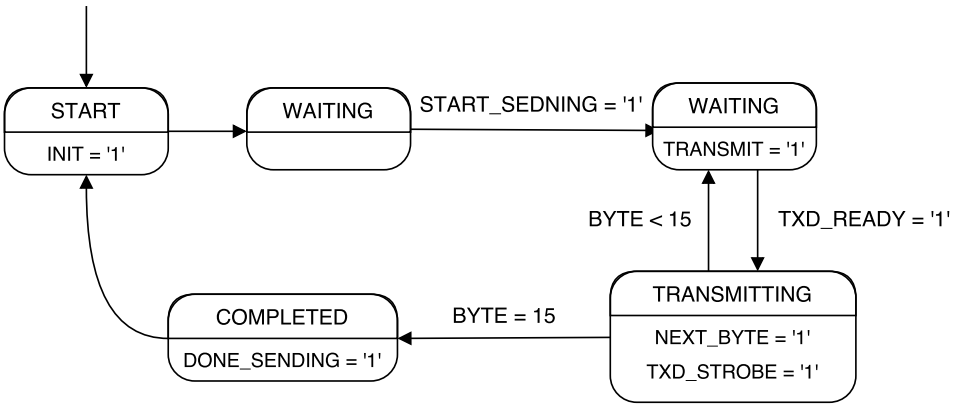
- START - Startovní stav po resetu.
- WAITING - v tomto stavu se čeká, dokud nám sériová linka neoznámí příchod nových dat signálem `RXD_STROBE`.
- RECEIVING - Stav na přijímání nových dat. V tuto chvíli jsou aktivní dva signály. `RECEIVE`, díky němuž se nahrají data z `RXD_DATA` do `DATA` (příslušná pozice se vypočítá v čítači). Dalším aktivním sig-



Obrázek 3.2: Stavový diagram přijímání dat

nálem je `NEXT_BYTE`, který inkrementuje čítač. Pokud už jsme přijali všechny byty, pokračujeme do stavu COMPLETED.

- COMPLETED - Konečný stav po přijetí všech 16 bytů (`BYTE` = 16). Aktivní signál `DATA_RECEIVED` je odeslán do řadiče. Pokud jsou přijaty všechna data, může začít šifrování. Poté se ještě signálem `INIT` inicializuje čítač do 1.



Obrázek 3.3: Stavový diagram odesílání dat

3.1.2.2 Odesílání dat

- START - Startovní stav po resetu. Aktivní signál je `INIT`, který nuluje bytový čítač.

- WAITING - V tomto stavu se čeká, dokud řadič neoznámí, že můžeme začít odesílat data, tedy že jsou data připravena po zašifrování. O tom informuje vstupní signál `START_SENDING`.

- LOADING - Čeká se, dokud sériová linka nepošle signál `TXD_READY`, abychom na ní mohli příslušná data odeslat. Aktivní je signál `TRANSMIT`, který připraví na výstupní signál `TXD_DATA` data z `CIPHERTEXT`. Jaká část ze 128 bitů se použije opět závisí na hodnotě bytového čítače `BYTE`.

- TRANSMITTING - Stav pro odesílání dat. Opět jsou aktivní dva signály - `TXD_STROBE`, který dává signál sériové lince, že může odeslat, a `NEXT_BYTE`, který inkrementuje čítač. Pokud jsme odeslali všechny byty, pokračujeme do stavu COMPLETED, jinak se vracíme do stavu LOADING.

- COMPLETED - Konečný stav po odeslání všech 16 bytů (`BYTE` = 15). Aktivní signál `DONE_SENDING` je odeslán do řadiče, který na jeho základě může signalizovat, že se mohou zažít zpracovávat nová data.

3.1.3 AES

Entita pro šifrování dat algoritmem AES je rozdělena na dvě komponenty, datavou a řídící cestu. Řídící je uložena v entitě `AES_CONTROLLER`, datová v entitě `AES_DATA` a jsou propojeny jen řídícími signály. do entity vstupuje signál `READY`, který startuje samotné šifrování, tedy indikuje, že vstupní data a klíč jsou připraveny. Vstupní data jsou uložena ve 128 bitovém vektoru `PLAINTEXT` a klíč je zase ve vektoru `KEY`. Potom, co doběhne šifrování do poslední rundy a na výstupním signálu `CIPHERTEXT` jsou připravena zašifrovaná data určená pro odeslání po sériové lince, řadič pošle aktivní signál `FINISHED`.

Kód 3.3: Rozhraní AES

```
entity AES is
    port (
        CLK        : in  STD_LOGIC;
        RESET      : in  STD_LOGIC;
        READY      : in  STD_LOGIC;
        PLAINTEXT  : in  STD_LOGIC_VECTOR (0 to 127);
        KEY        : in  STD_LOGIC_VECTOR (0 to 127);
        FINISHED   : out STD_LOGIC;
        CIPHERTEXT : out STD_LOGIC_VECTOR (0 to 127)
    );
end AES;
```

3.1.3.1 Datová cesta entity AES

Hlavní částí této práce je samotný algoritmus AES, který je uložený v entitě `AES_DATA`. Každá operační část algoritmu je rozdělena do samostatných entit. Blokové schéma nalezneme na obrázku 3.4.

Kód 3.4: Rozhraní pro šifrování

```
entity AES_DATA is
    port (
        CLK        : in  STD_LOGIC;
        RESET      : in  STD_LOGIC;
        LOAD       : in  STD_LOGIC;
        ENCRYPT    : in  STD_LOGIC;
        DONE       : in  STD_LOGIC;
        ROUND      : in  STD_LOGIC_VECTOR (0 to 3);
        PLAINTEXT  : in  STD_LOGIC_VECTOR (0 to 127);
        KEY        : in  STD_LOGIC_VECTOR (0 to 127);
        CIPHERTEXT : out STD_LOGIC_VECTOR (0 to 127)
    );
end AES_DATA;
```

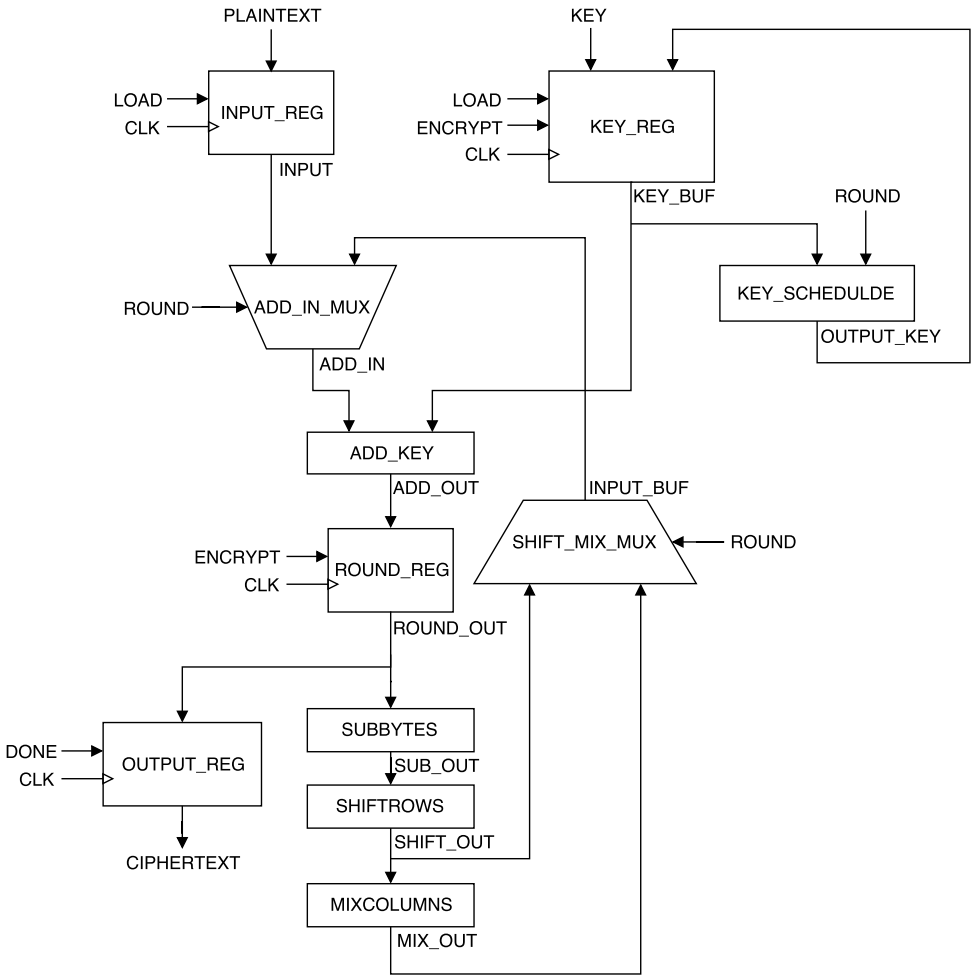
Ve chvíli, kdy jsou data v `PLAINTEXT` připravena na šifrování, přijde signál `LOAD` a data jsou v procesu `INPUT_REG`, jenž funguje jako registr, nahrána do `INPUT`. To samé platí i pro `KEY` a jeho příslušný proces `KEY_REG`, který se nahraje do `KEY_BUF`. Ve chvíli, kdy jsou tedy data připravena ve startovních registrech, přijde z řadiče signál `ENCRYPT` a začně samotné šifrování.

V první rundě se jako vstup do `AddRoundKey` použije `INPUT`, avšak v dalších rundách je jako vstup použit `INPUT_BUF`. Na to je použit proces s názvem `ADD_IN_MUX`, tedy jednoduchý multiplexor, který na základě toho, jaká je aktuální runda, rozhodne, jaký vstup použít. V signálu `ROUND` je uloženo číslo aktuální rundy a čítač rund se nachází v řadiči AES.

Pro účely úspěšného útoku pomocí DPA byl mezi `AddRoundKey` a `SubBytes` vložen registr s názvem `ROUND_REG`, který uchovává hodnotu stavu při šifrování. Pokud je vstupní signál `ENCRYPT` v jedničce, překopíruje vstupní data (v tomto případě výstup z `AddRoundKey` - signál `ADD_OUT`) na výstup (signál `ROUND_OUT`) a pokračuje další runda.

`ROUND_OUT` je vstupním signálem pro entitu `SubBytes`, ze které vystupuje signál `SUB_OUT`, jenž je vstupním signálem pro `ShiftRows`. Z této entity vystupuje signál `SHIFT_OUT`. Ten vstupuje do `MixColums` a vystupuje signál `MIX_OUT`. Entita se `MixColumns` používá v každé rundě až na tu poslední. Pro tuto skutečnost je určený multiplexor s názvem `SHIFT_MIX_OUT`, který určuje jaký signál použít v závislosti na čísle rundy. Výstupem je vnitřní signál `INPUT_BUF`.

Druhým vstupem `AddRoundKey` je signál `KEY_BUF`, ve kterém je uložena hodnota aktuálního rundovního klíče. Nezměněný vstupní klíč se použije jen



Obrázek 3.4: Blokové schéma datové cesty AES

na úplném začátku šifrování, v dalších rundách se použije klíč odvozený z něj. Výpočet zajišťuje entita `KeySchedulde`, která jej výpočítá opět na základě čísla rundy.

Posledním procesem je `OUTPUT_REG` registr, který, pokud je šifrování dokončeno a data jsou připravena, překopíruje hodnotu `ROUND_OUT` do výstupního signálu `CIPHERTEXT`. O tom, zda je vše dokončeno, informuje vstupní signál `DONE`, který přijde z řadiče AES.

3.1.3.2 Jednotlivé entity v AES

AddRoundKey Entita `AddRoundKey` obsahuje jeden jednoduchý proces, kdy provede bitovou operaci xor nad vstupními signály `ADD_IN` a `KEY_IN` a je-

jím výstupním signálem je `ADD_OUT`.

SubBytes Další entita `SubBytes` nahradí vstupní `SUB_IN` 128 signál, rozdělený po 8 bitech, podle nahrazovací tabulky (entita `SBox`). Toto je jediný nelineární prvek celého šifrování a díky tomu se na něj zaměřují útoky.

ShiftRows V `ShiftRows` dochází k jednoduchému posunu aktuálního stavu po řádcích. Ve VHDL jde o jednoduché přečíslování vstupního signálu (pojmenovaný `SHIFT_IN`) na ten výstupní (`SHIFT_OUT`).

MixColumns Složitější je oproti ostatním entitám `MixColumns`. Operace je založena na maticovém násobení v $GF(2^8)$. V procesech `MULTIPLICATION_2X` a `MULTIPLICATION_3X` se předpočítají výsledky pro násobení dvěma, resp. třemi. Na výstupní signál `MIX_OUT` jsou tyto mezivýsledky podle příslušného sloupcového posunu zkombinovány pomocí bitové operace xor.

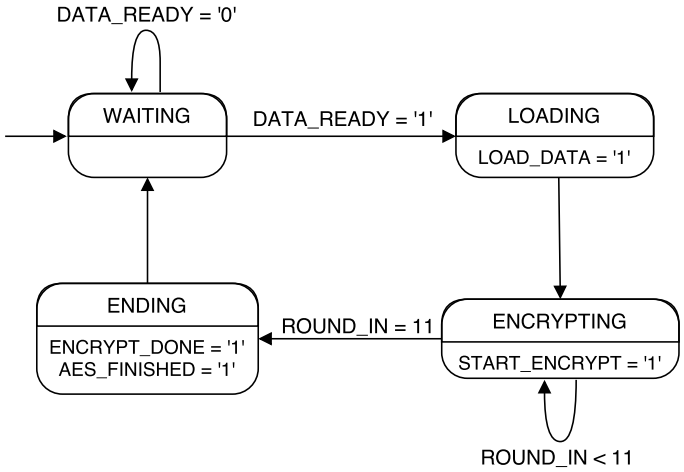
KeySchedulde Poslední entitou je `KeySchedulde`, který, jak už bylo řečeno, počítá aktuální hodnotu rundovního klíče. Pro výpočet je použita jak tabulka hodnot `SBox`, která se využívá v `SubBytes`, tak i tabulka `RCON` obsahující konstanty potřebné během počítání. Opět jde jen o jednoduché xorování příslušných hodnot.

3.1.3.3 Řídící cesta entity AES

Řadič (entita `AES_CONTROLLER`) obsahuje konečný automat typu Moore sloužící k vysílání signálů do a z entity `AES_DATA`. Následuje jeho rozhraní a na obrázku 3.5 vidíme jeho stavový diagram.

Kód 3.5: Rozhraní pro šifrování

```
entity AES_CONTROLLER is
    port (
        CLK           : in  STD_LOGIC;
        RESET         : in  STD_LOGIC;
        DATA_READY    : in  STD_LOGIC;
        LOAD_DATA     : out STD_LOGIC;
        START_ENCRYPT : out STD_LOGIC;
        ENCRYPT_DONE  : out STD_LOGIC;
        AES_FINISHED  : out STD_LOGIC;
        ROUND         : out STD_LOGIC_VECTOR (0 to 3)
    );
end AES_CONTROLLER;
```



Obrázek 3.5: Stavový diagram pro řidící část šifrování AES

Součástí řadiče je čítač, který čítá od nuly do jedenácti a udává číslo aktuální rundy. Ve skutečnosti je rund jen deset, kdy ta jedenáctá je součástí pouze z toho důvodu, aby výstupní registr vydal zašifrovaná data.

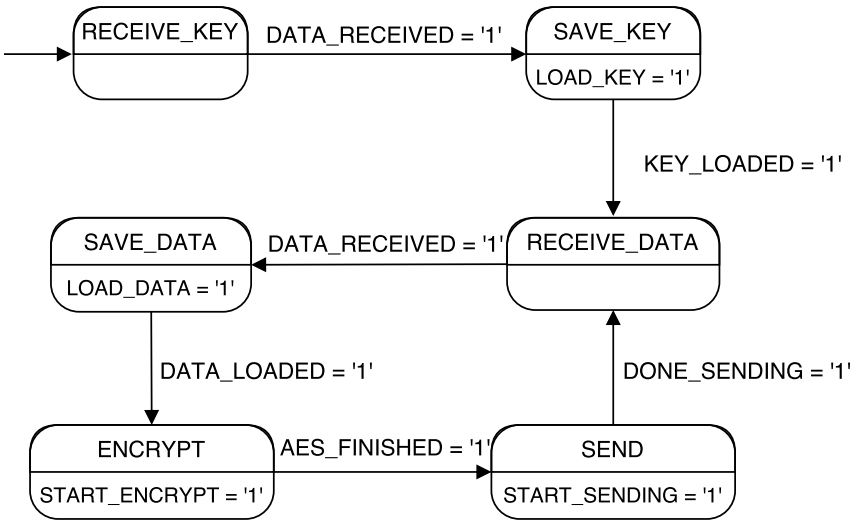
- WAITING - Startovní stav po resetu. Čeká na signál `DATA_READY`, který indukuje připravená data po přijímání ze sériové linky.
- LOADING - Zde se nahrají data ze vstupního signálu `PLAINTEXT` do registru `INPUT_REG` a vstupní klíč `KEY` do registru `KEY_REG`. Datové části to dáváme najevo aktivním signálem `LOAD_DATA`.
- ENCRYPTING - Zde probíhá šifrování. Po celou dobu 10 rund je výstupní signál `START_ENCRYPT` aktivní a čeká se až rundovní čítač dojde k číslu 11.
- ENDING - Poslední stav, kdy doběhlo šifrování, vysílá dva signály. Jeden míří do entity pro šifrování (signál `ENCRYPT_DONE`), jenž připraví výstupní data do signálu `CIPHERTEXT`. Ten druhý dává najevo tu samou skutečnost, ale tentokrát míří do hlavního řadiče, který posléze signál odešle do komunikátoru pro odesílání zašifrovaných dat po sériové lince.

3.1.4 Řadič

Poslední entitou je hlavní řadič celé implementace, který má na starosti jak zpracování řídících signálů ze sériové linky, tak signály o stavu šifrování. Tedy signály vedoucí z entit `COMMUNICATOR` a `AES`. Řadič opět obsahuje jednoduchý konečný automat typu Moore. Stavový diagram je vidět na obrázku 3.6.

Kód 3.6: Rozhraní řadiče

```
entity CONTROLLER is
    port (
        CLK             : in  STD_LOGIC;
        RESET           : in  STD_LOGIC;
        DATA_RECEIVED   : in  STD_LOGIC;
        DONE_SENDING    : in  STD_LOGIC;
        AES_FINISHED    : in  STD_LOGIC;
        KEY_LOADED      : in  STD_LOGIC;
        DATA_LOADED     : in  STD_LOGIC;
        START_ENCRYPT   : out STD_LOGIC;
        START_SENDING   : out STD_LOGIC;
        LOAD_KEY        : out STD_LOGIC;
        LOAD_DATA       : out STD_LOGIC
    );
end CONTROLLER;
```



Obrázek 3.6: Stavový diagram pro řadič

- RECEIVE_KEY - Startovní stav po resetu. Stav pro nahrání nového klíče. Dokud nepřijde od komunikátoru aktivní signál `DATA_RECEIVED`, čekáme v tomto stavu.

- SAVE_KEY - Přes sériovou linku byl přijat klíč, signálem `LOAD_KEY` ho uložíme, odpovědí je signál `KEY_LOADED` pro přechod do dalšho stavu.

- RECEIVE_DATA - V tomto stavu dochází k přijímání dat ze sériové linky. do dalšího stavu se přejde po příchodu signálu `DATA_RECEIVED`, jenž signalizuje příjem všech 16 bytů dat.
- SAVE_DATA - Data přijata, signálem `LOAD_DATA` uložíme do `PLAINTEXT`, jako odpověď přijde signál `DATA_LOADED` a může začít šifrování.
- ENCRYPT - Šifrovací stav. Aktivní výstupní signál `START_ENCRYPT` vysílá dokud se nedokončí šifrování, tj. neproběhne všech 10 rund. Konec šifrování signalizuje vstupní signál `AES_FINISHED`, který přijde z entity `AES`.
- SEND - Stav pro odeslání dat na sériovou linku. Z tohoto stavu se přejde do dalšího až po přijetí vstupního signálu `DONE_SENDING`. Do té doby je vysílán výstupní signál `START_SENDING`. Po dokončení můžeme přijmout další data určená k šifrování.

3.2 Fault-injection obrany

V následující kapitole si popíšeme, jak se zvolené varianty AES zabezpečené proti poruchám liší od základní verze. Komunikace po sériové lince probíhá zcela stejně, tedy celá entita `COMMUNICATOR` se jen překopírovala. To samé se udělalo s entitou `CONTROLLER`. Jediná změna, která v každé variantě nastala, je modifikace entity `SubBytes` v šifrovací části implementace. Přibyl nový signál `WARNING`, který je vyveden až do hlavního rozhraní a připojen na LED diodu desky. Pokud by došlo v průběhu šifrování k nějaké chybě v datech, dioda by ji indikovala.

V `SubBytes` byla entita `SBox` vyměněna za entitu `Parity` (v případě modulo 3 se tato entita jmenuje `Modulo` z důvodu přehlednosti). `Parity` pak obsahuje jeden proces, ve kterém probíhá výpočet parity (vstupní a výstupní), jejich predikce (vstupní a výstupní) a porovnání těchto proměnných. Pokud by nějaké číslo nesedělo, pak by signál `WARNING` signalizoval chybu.

3.2.1 Parita

Jednobitová sudá parita se spočítá jako bitový xor všech bitů jednoho bytu spolu. Číslo vyjde buď 1 pro sudý počet jedničkových bitů nebo 0 pro lichý. Výsledné předpovídané parity jsme si spočítali předem a uložili do dvou tabulek, aby jejich výpočet nezpomaloval průběh šifrování. Paritu vstupního signálu najdeme v proměnné `PARITY_IN`. Parita signálu po výměně SBox je v proměnné `PARITY_OUT`. V `PREDICTION_OUT` je podle příslušné tabulky hodnota predikce parity, která odpovídá vstupní hodnotě, a v `PREDICTION_IN` hodnotě výstupní. `COMPARE_IN` a `COMPARE_OUT` porovnává parity s jejich predikcí pomocí bitové operace xor. `WARNING` se vypočítá bitovou operací or nad porovnávacími proměnnými.

Kód 3.7: Výpočet jednobitové parity

```
PARITY : process (INPUT_SIGNAL, OUTPUT_SIGNAL)
    variable PARITY_IN, PARITY_OUT,
             PREDICTION_IN, PREDICTION_OUT,
             COMPARE_IN, COMPARE_OUT : STD_LOGIC;
begin
    PARITY_IN := INPUT_SIGNAL(0) xor INPUT_SIGNAL(1) xor
                 INPUT_SIGNAL(2) xor INPUT_SIGNAL(3) xor
                 INPUT_SIGNAL(4) xor INPUT_SIGNAL(5) xor
                 INPUT_SIGNAL(6) xor INPUT_SIGNAL(7);
    PARITY_OUT := OUTPUT_SIGNAL(0) xor OUTPUT_SIGNAL(1) xor
                  OUTPUT_SIGNAL(2) xor OUTPUT_SIGNAL(3) xor
                  OUTPUT_SIGNAL(4) xor OUTPUT_SIGNAL(5) xor
                  OUTPUT_SIGNAL(6) xor OUTPUT_SIGNAL(7);

    PREDICTION_IN  := INPUT_PARITY_TABLE (to_integer(unsigned(
                      OUTPUT_SIGNAL(0 to 7))));
    PREDICTION_OUT := OUTPUT_PARITY_TABLE(to_integer(unsigned(
                      INPUT_SIGNAL (0 to 7))));

    COMPARE_IN  := PARITY_IN  xor PREDICTION_IN;
    COMPARE_OUT := PARITY_OUT xor PREDICTION_OUT;

    WARNING <= COMPARE_IN OR COMPARE_OUT;
end process PARITY;
```

3.2.2 Dvojitá parita

Další výpočty parit jsou odvozeny od té jednobitové, takže se jim budeme věnovat jen zkráceně. Ve dvojité paritě se pro vstupní i výstupní paritu počítají dvě hodnoty. `PARITY_IN_EVEN` určuje paritu vstupního bytu na všech jeho lichých pozicích a `PARITY_IN_ODD` paritu na pozicích sudých. K tomu jsou i odpovídající výstupní parity. V tabulce je opět předpočítána hodnota predikcí, která je porovnána s tou reálnou. `WARNING` je bitová operace or nad všemi hodnotami.

3.2.3 Komplementární parita

Komplementární parita je v podstatě stejná jako parita dvojitá, jen nebere sudé a liché indexy, ale sudou a lichou paritu celého bytu. Zbytek je zcela totožný.

3.2.4 Modulo 3

Pro tuto variantu musela být přidána nová tabulka `MODULO3_TABLE`, která udává zbytek po dělení třemi. VHDL jazyk sice podporuje funkci modulo, jenže

jen s čísly dělitelnými 2, což v tomto případě nebylo možné použít. V tomto případě byla také tabulka vyplněna hodnotami integer místo `STD_LOGIC`, jak tomu je v ostatních variantách.

3.2.5 Hammingův kód

Proměnná se rozroste na osm míst, čtyři pro vstupní a čtyři pro výstupní signál. Podle příslušné matice (obrázek 3.7) jsme vyplnili, jaké bity odpovídají jaké paritě a opět jsme si předpočítali tabulky predikcí. `WARNING` je potom výstupní signál 8 proměnných.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrázek 3.7: Matice pro Hammingův kód

Pro výpočet parit byla použita matice na obrázku 3.7 a jak je na něm vidět, matice o velikost 8x4 byla spojena s jednotkovou maticí velikosti 4x4. Výsledné čtyři paritní bity se spočtou jako bitový xor na pozicích, kde je jednička, kromě matice jednotkové. Tedy:

$$P_1 = I(0) \oplus I(1) \oplus I(2) \oplus I(6) \oplus I(7) \tag{3.1}$$
$$P_2 = I(0) \oplus I(3) \oplus I(4) \oplus I(6) \oplus I(7) \tag{3.2}$$
$$P_3 = I(1) \oplus I(3) \oplus I(5) \oplus I(6) \tag{3.3}$$
$$P_4 = I(2) \oplus I(4) \oplus I(5) \oplus I(7) \tag{3.4}$$

3.3 DPA útok

Samotná realizaci útoku DPA byla provedena v programovacím prostředí Matlab, které má svůj vlastní skriptovací jazyk. Program slouží mimo jiné pro analýzu dat a práci s maticemi.

Skript provádějící útok nahraje data získaná z osciloskopu a vytvoří hypotézy, které se v tomto případě odvodily použitím Hammingovy vzdálenosti. Pro urychlení výpočtu byl výpočet zaměřen jen na poslední desátou rundu algoritmu AES. Výsledkem je tedy rundovní klíč, který je použit právě v této rundě. Zjistit si použitý šifrový klíč už nebylo tak těžké. Protože ve všech měřeních byl použit ten samý šifrový klíč, byl na začátek programu přidán předpokládaný výsledek klíče, který má vyjít a na konci se jen kontrolovala správnost porovnáním dvou hodnot. Pro další použití při zvolení jiného klíče by se měla dopsat část kódu, která hlavní klíč vypočítá.

Při vyhodnocování měření bylo zjištěno, že pro úspěšný útok je potřeba nejméně 5000 průběhů o velikosti 1000 vzorků. Bylo tedy naměřeno pro každou

variantu 100000 průběhů, které se můžou rozdělit na 20 nezávislých měření a na každý zaútočit zvlášť. Program byl tedy upraven, aby provedl 20 útoků a pokaždé nahrál odpovídající data ze souborů.

Pro použití útoku na implementaci AES na FPGA byl zvolen přístup přes použití Hammingovy vzdálenosti na poslední rundu šifrování. Zašifrovaná data jsou načtena ze souboru a z nich se vypočítá inverzními operacemi, podobně jako při dešifrování, hodnota, ve které byly, když se nacházely v rundovním registru. Ten je umístěn před operací SubBytes, tedy musím provést opreaci AddRound, InvShiftRows a InvSubBytes. Hammingova vzdálenost je poté vypočtena jako Hammingova váha rozdílu mezi šifrovým textem a hodnotou před desátou rundou.

Vyhodnocení matice hypotéz probíhá následovně. Pomocí funkce se vypočte korelační koeficient pro všechny hypotézy. Protože tyto hodnoty můžou nabývat hodnot od -1 do 1, je z matice vypočtena její absolutní hodnota a poté hledáme na jakém řádku se nachází maximum. Pokud výsledný byte odpovídá hledanému, zmenší se počet průběhů o 25 a výpočet se opakuje. Ve chvíli, kdy byte neodpovídá tomu správnému, přičítá se zpět po jedné, aby se zjistil přesný počet průběhů, díky kterým se dá daný byte klíče prolomit. Z těchto hodnot pro každý byte je vybrán ten největší, aby se zjistil počet průběhů nutných pro prolomení klíče celého (všech 16 bytů).

Kapitola 4

Testování

4.1 Testování AES

Každé testování lze rozdělit na dvě části - verifikaci a validaci. Verifikace je testování v nějakém počítačovém simulátoru. V našem případě to byl simulátor ISim, který je součástí Xilinx ISE Design Suite. Validace poté znamená ověření si správnosti návrhu nahráním do FPGA přípravku.

4.1.1 Verifikace

Protože návrh AES v této práci je rozdělen na jednotlivé entity, bylo možné každou z nich otestovat zvlášť. Tedy pro entity SubBytes, ShiftRows, MixColumns, AddRoundKey a KeySchedulde byly vytvořeny jejich odpovídající testbench a v simulátoru byla ověřena jejich korektnost. Poté, co byla napsána celá implementace AES, bylo možné vytvořit testbench pro tuto entitu a ověřit správnost šifrování.

Poté, co byly vytvořeny zbývající varianty AES, tedy ty obsahující nějakou formu obrany proti chybám, vytvořili jsme testbench jen pro část s entitou `Parity`. Protože zbytek kódu zůstal zachován, usoudili jsme, že testování celé entity AES není zapotřebí. Protože jsme nezjistili, jak správně verifikovat používání sériové linky v simulátoru, její testování probíhalo až formou validace nahrávání dat do přípravku.

4.1.2 Validace

Jak už bylo rečeno, použili jsme desku Spartan 3E Starter Kit, na které probíhala i samotná validace. Nejprve jsme otestovali samotné zasílání dat do a z přípravku po sériové lince. K tomu nám dopomohla skutečnost, že jsem ve svém studiu už desku používal a měl jsem k dispozici napsaný program na používání LCD na této desce.

Poté byly pospojovány všechny entity návrhu do finální verze a mohli jsme začít šifrovat data v přípravku. Pro komunikaci po sériové lince byl použit program Advanced Serial Port Terminal[25]. Do přípravku byl odeslán otevřený text a kontrolován byl text zašifrovaný. Toto bylo nutné udělat pro každou variantu programu a také kdykoliv, když se některý z nich nějak pozměnil. Validace dat přes Terminal probíhala i před každým začátkem měření na osciloskopu.

Kapitola 5

Diferenciální proudová analýza

5.1 Postup měření

Útok na šifru AES na FPGA přípravku probíhal následujícím způsobem. Do přípravku se nejdříve nahrála přes USB rozhranní implementace algoritmu, k čemuž slouží například program iMPACT od společnosti Xilinx, který je stejně jako ISim součástí Xilinx ISE Design Suite. Přípravek se připojí k počítači přes sériovou linku pro komunikaci. Jak je možné zjistit v realizaci, nejprve se musí do přípravku nahrát klíč, který se útokem budeme snažit odhalit. Zašleme tedy prvních 16 bytů použitím programu Advanced Serial Port Terminal. Posléze zašleme dalších 16 bytů pro ověření správnosti šifrování a od přípravku dostaneme jako odpověď zašifrovaná data, jejichž správnost si můžeme ověřit jakýmkoliv programem třetích stran. V tuto chvíli může začít měření spotřeby.

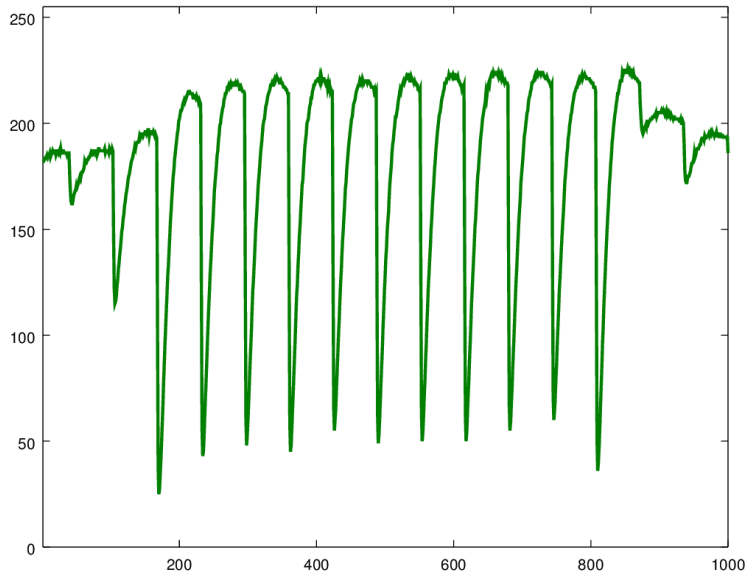
Náš přípravek Spartan 3E Starter Kit byl pro toto speciálně upraven přidáním obvodu s rezistorem, na kterém se měří vnitřní napětí. Deska má též odstraněny blokovací kondenzátory C158 - C175. Přípravek se připojí k osciloskopu přes AC předzesilovač PA 303 BNC. V této práci byl použit osciloskop Agilent MSO7104A. Na něm se zapne bandwith limit, což nám filtruje nechtěný šum a máme díky němu zřetelnější obraz. Pro komunikaci počítače s ociloskopem byl použit program SC Power Measurement, do kterého se zadá číslo portu, na který je přes USB osciloskop připojen. Program slouží i pro ověření funkčnosti sériové linky a správného nastavení osciloskopu, které se dá programem i uložit do počítače a zpětně nahrát. Bylo zjištěno, že pokud se tak provede, sníží se tím počet vzorků průběhu jednoho měření. Následně se může zapnout samotné měření, kdy se specifikuje, jaký počet měření chceme dostat.

5.2 SC Power Measurement

SC Power Measurement vytváří náhodné hodnoty 16 bytů, které odešle do přípravku, ty se v něm zašifrují a odešlou zpátky do počítače. Osciloskop během této operace změří průběh napětí a změřená data odešle také do počítače. Program všechno uloží do příslušných souborů. Otevřené texty lze najít v souboru plaintext.txt, ty zašifrované zase v souboru ciphertext.txt. Na příslušném řádku lze najít odpovídající byty - 20. řádek v jednom souboru odpovídá 20. řádku v souboru druhém. Průběhy spotřeby lze najít v binárním souboru traces.bin, kdy každému průběhu odpovídá tzv. trace, která je v tomto případě dlouhá 1000 vzorků. Po skončení měření je tato hodnota uložena do souboru traceLength.txt. Použitý osciloskop má rozlišení 8 bitů, tedy výsledné vzorky se nachází v intervalu 0-255 (nejvyšší číslo, které osmi byty můžeme reprezentovat je 11111111_2, což odpovídá 255_{10}).

5.3 Útok na FPGA

5.3.1 Základní varianta



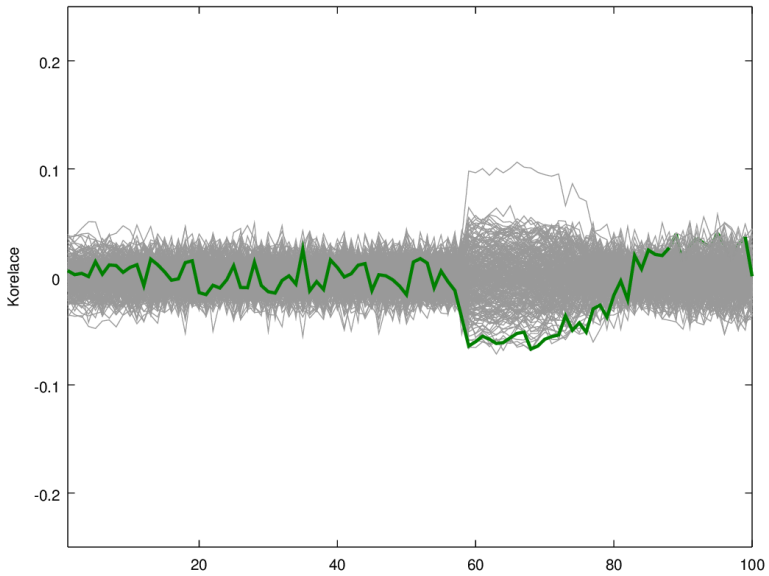
Obrázek 5.1: Vzorek spotřeby pro základní variantu AES, jednotlivé rundy šifrování jsou zřetelně vidět

Implementace této základní varianty odpovídá té, jenž je popsána v kapitole 3.1. Varianta neobsahuje žádné zabezpěčení pro jakékoliv poruše, tedy

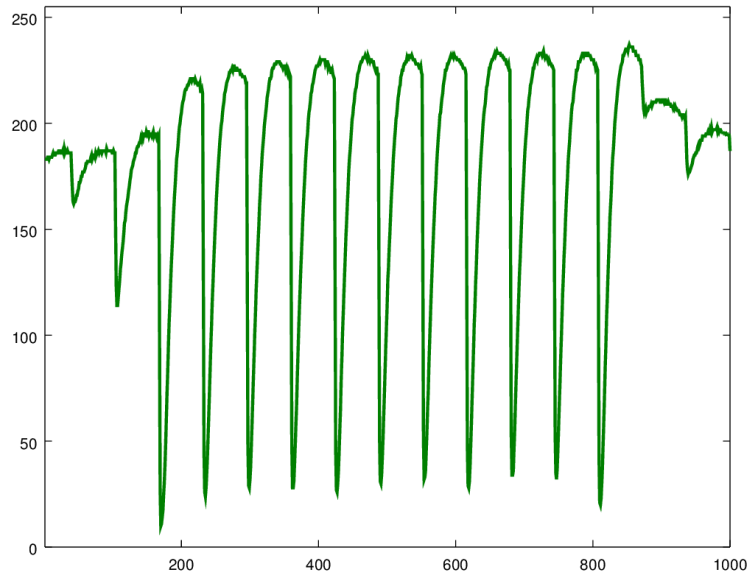
nedochází zde k žádné informační redundanci. Všechny ostatní varianty vychází z této základní a kromě zabezpečení týkající se entity SubBytes, jsou s ní zcela totožné. Ve všech měřeních byl použit stejný šifrový klíč zapsaný v hexadecimální podobě $00112233445566778899AABBCCDDEEFF_{16}$.

Bylo naměřeno 100 000 vzorků spotřeby. Na obrázku 5.2 je vidět průběh korelace pro první byte 10. rundovního klíče základní varianty mezi 750. a 850. vzorkem, kdy nastává čas 10. rundy. Zeleně je zvýrazněn předpokládaný správný byte s hodnotou 36_{16}. Místo, kdy probíhá tato runda víme ze znalosti průběhu šifrování algoritmu AES a lze zjistit i z obrázku 5.1.

Přestože všechny ostatní varianty zcela obsahují tuto základní, nepodařilo se na ní úspěšně prolomit použitý klíč ani při použití velkého množství traců. Avšak na všech zabezpečených variantách se klíč odhalit podařilo a to ve všech případech při použití jen 5000 trace. Za důsledek to může mít jiné zpracování syntézním programem, nemělo by to být použitím či nepoužitím detekujících obvodů, protože kolegům se podařilo prolomit šifru i bez nich. Pro tuto skutečnost bylo rozhodnuto, že odvozené varianty se budou porovnávat jen mezi sebou, nikoliv se základní variantou.



Obrázek 5.2: Průběh korelace pro základní variantu mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

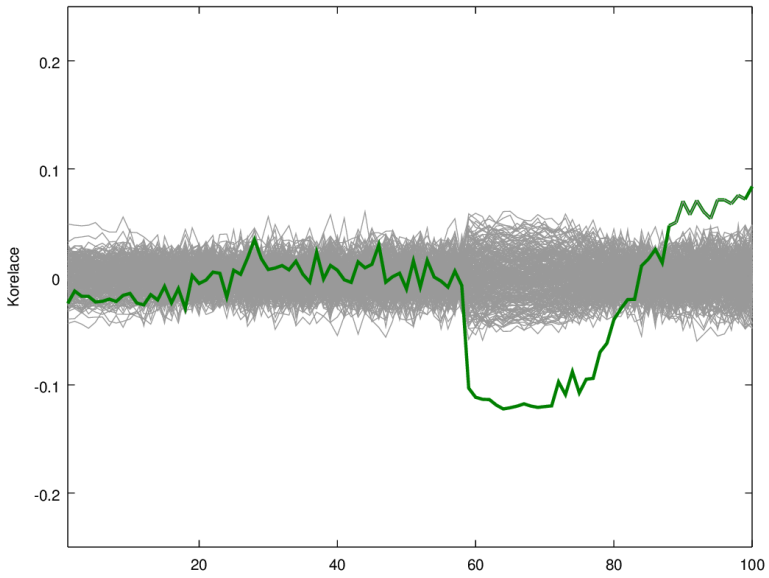


Obrázek 5.3: Vzorek spotřeby pro variantu s jednoduchou paritou

5.3.2 Varianta zabezpečená jednoduchou paritou

Tato varianta obsahuje zabezpečení proti poruchám obvodu výpočtem parity u operace SubBytes. Parita je realizována jako jeden bit, ve kterém se nachází jednička, pokud součet všech jedniček v bytu je lichý, v opačném případě je zde uložena nula. Popis její implementace lze najít v kapitole 3.2.1.

Bylo naměřeno 100 000 vzorků spotřeby, které byly rozděleny na jednotlivých 5000 měření a bylo na ně nahlíženo jako na 20 různých útoků. Průběh korelace mezi 750. - 850. pro první byte 10. rundovního klíče lze vidět na obrázku 5.4. Zeleně zvýrazněný vzorek je pro hodnotu 36_{16}, což je správná hodnota tohoto prvního bytu. Medián nejmenších počtů vzorků nutných pro prolomení 1. - 16. bytu klíče naleznete v tabulce 5.1, tedy z 20 útoků byl vždy vybrán údaj pro nejmenší nutný počet vzorků pro prolomení jednoho bytu klíče a z těchto 20 hodnot byl vypočítán medián, to se provedlo pro všech 16 bytů. Následně byla vybrána největší hodnota z každého měření, která tedy udává, kolik bylo nutných vzorků pro prolomení celého klíče. Z těchto 20 hodnot byl opět vypočítán medián a pro jednoduchou paritu vychází 2392,5 s mezikvartilovým rozpětím (vysvětlení v kapitole 5.4) 698. Medián vyšel největší ze všech měřených variant.



Obrázek 5.4: Průběh korelace pro variantu s jednoduchou paritou mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

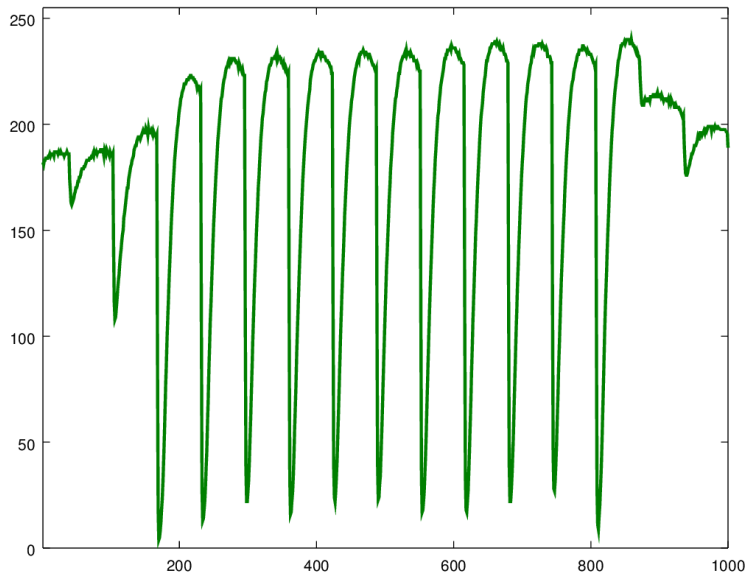
Tabulka 5.1: Tabulka mediánů vzorků pro každý byte ve variantě s jednoduchou paritou

1.	2.	3.	4.	5.	6.	7.	8.
1529	1548.5	1251.5	1137	1526.5	1115.5	539	1902.5
9.	10.	11.	12.	13.	14.	15.	16.
702.50	1127.5	881	650.50	537	807	1036	978.50

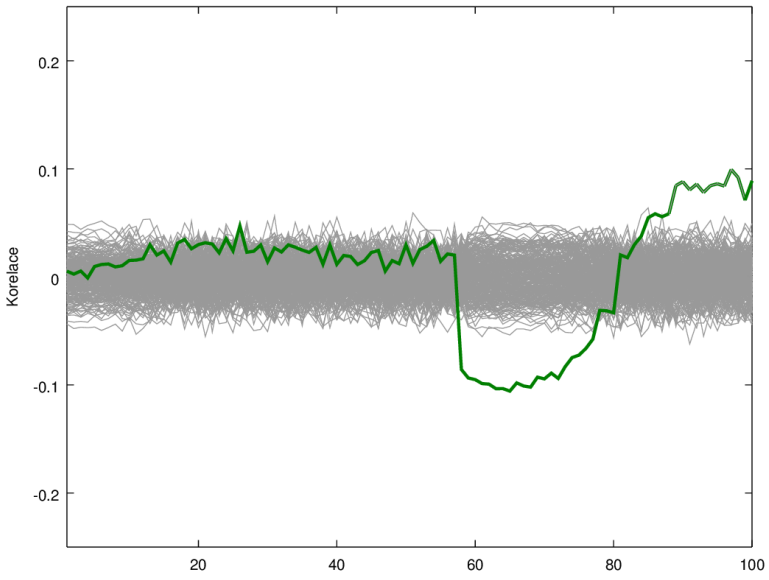
5.3.3 Varianta zabezpečená dvojitou paritou

Tato varianta obsahuje zabezpečení proti poruchám obvodu výpočtem parity u operace SubBytes. Parita je realizována jako bity dva - první bit pro informaci o jedničkových bitech na lichých indexech v bytu a druhý na indexech sudých. Pokud počet jedniček je sudý, bit je nastaven na jedna, v opačném případě je zde uložena nula. Popis její implementace lze najít v kapitole 3.2.2.

Bylo naměřeno 100 000 vzorků spotřeby rozdělených do 20 samostatných útoků. Průběh korelace mezi 750. - 850. pro první byte 10. rundovního klíče lze vidět na obrázku 5.6. Zeleně zvýrazněný vzorek je pro hodnotu 36_{16}, což je správná hodnota tohoto prvního bytu. Medián nejmenších počtů vzorků nutných pro prolomení 1. - 16. bytu klíče naleznete v tabulce 5.2. Následně byla vybrána největší hodnota z každého měření, která tedy udává, kolik bylo



Obrázek 5.5: Vzorek spotřeby pro variantu s dvojitou paritou



Obrázek 5.6: Průběh korelace pro variantu s dvojitou paritou mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

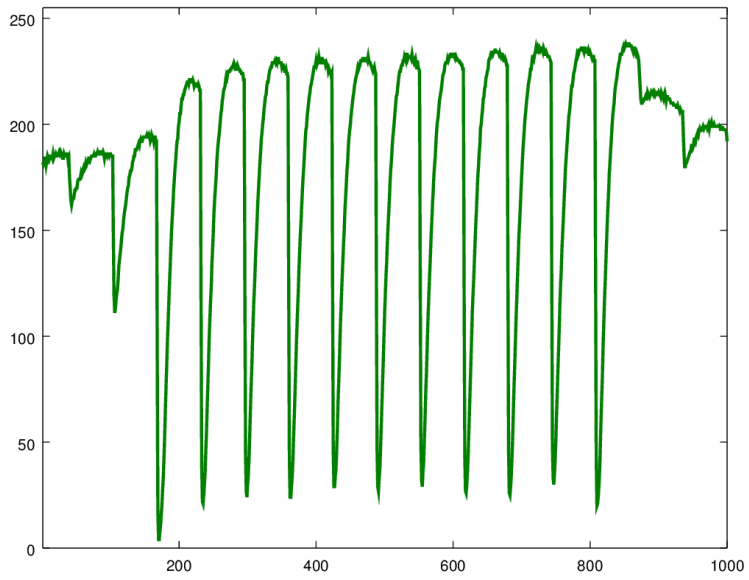
Tabulka 5.2: Tabulka mediánů vzorků pro každý byte ve variantě s dvojitou paritou

1.	2.	3.	4.	5.	6.	7.	8.
1623	1013	737.5	1244.5	1042	785.5	1082.5	929
9.	10.	11.	12.	13.	14.	15.	16.
641.5	842	670.5	759.5	712	1046.5	749.5	639.5

nutných vzorků pro prolomení celého klíče. Z těchto 20 hodnot byl opět vypočítán medián a pro dvojitou paritu vychází 2210 s mezikvartilovým rozpětím 993. Medián byl velmi podobný výsledku mediánu pro jednoduchou paritu.

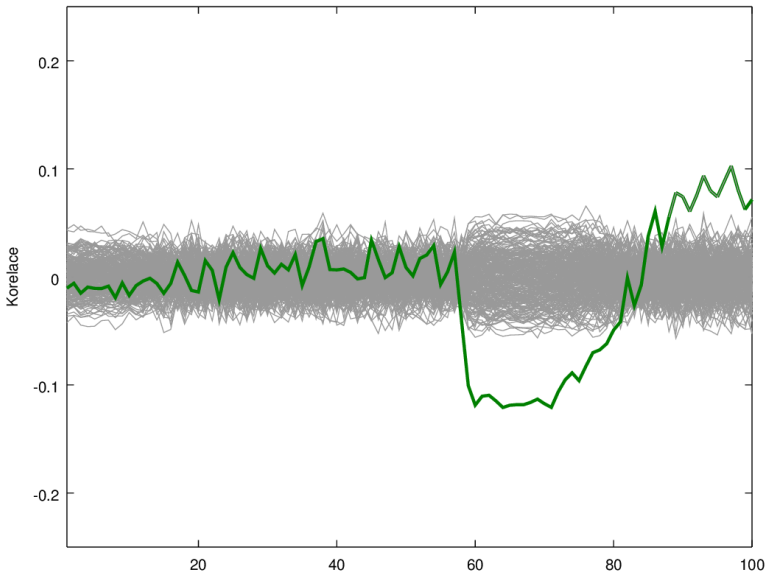
5.3.4 Varianta zabezpečená komplementární paritou

Tato varianta obsahuje zabezpečení proti poruchám obvodu výpočtem parity u operace SubBytes. Parita je opět realizována jako dva bity, ale v tomto případě je jeden bit pro paritu sudou a druhý pro paritu lichou. Obě dvě parity se počítají nad celým bytem. Popis implementace lze najít v kapitole 3.2.3.



Obrázek 5.7: Vzorek spotřeby pro variantu s komplementární paritou

Bylo naměřeno 100 000 vzorků spotřeby rozdělených do 20 samostatných útoků. Průběh korelace mezi 750. - 850. pro první byte 10. rundovního klíče lze vidět na obrázku 5.8. Zeleně zvýrazněný vzorek je pro hodnotu 36_{16}, což



Obrázek 5.8: Průběh korelace pro variantu s komplementární paritou mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

Tabulka 5.3: Tabulka mediánů vzorků pro každý byte ve variantě s komplementární paritou

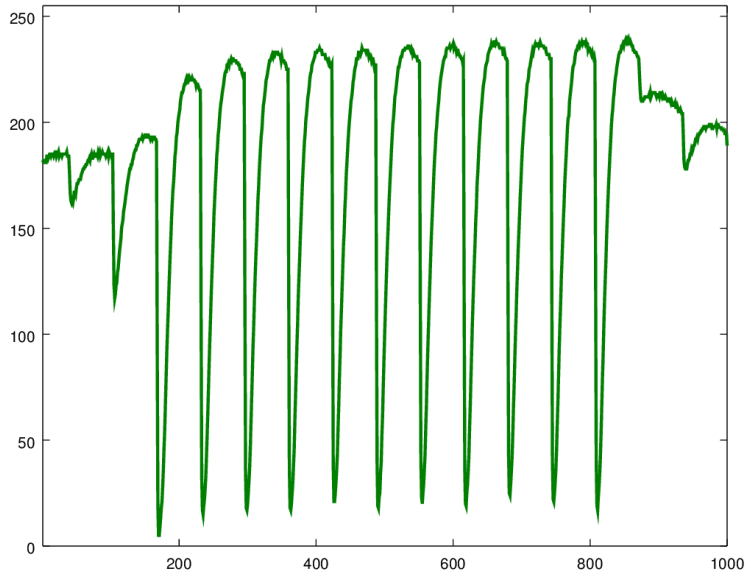
1.	2.	3.	4.	5.	6.	7.	8.
1006.5	1110.5	766	731	1098	568	964	1362.5
9.	10.	11.	12.	13.	14.	15.	16.
493	1090.5	977.5	705.5	523	776	1050	675

je správná hodnota tohoto prvního bytu. Medián nejmenších počtů vzorků nutných pro prolomení 1. - 16. bytu klíče naleznete v tabulce 5.3. Následně byla vybrána největší hodnota z každého měření, která tedy udává, kolik bylo nutných vzorků pro prolomení celého klíče. Z těchto 20 hodnot byl opět vypočítán medián a pro komplementární paritu vychází 1829,5 s mezikvartilovým rozpětím 324. Díky menší hodnotě rozpětí je přesnější hodnota mediánu.

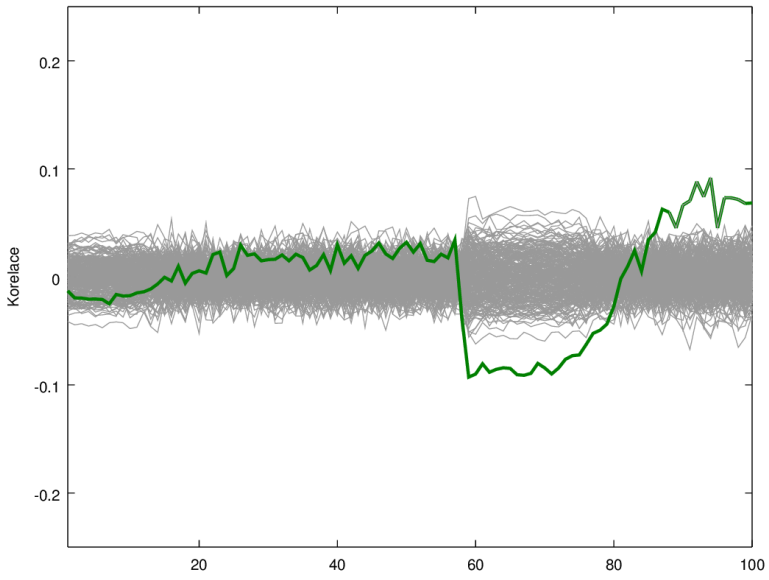
5.3.5 Varianta zabezpečená pomocí zbytků modulo 3

Tato varianta obsahuje zabezpečení proti poruchám obvodu výpočtem zbytku po operaci dělení 3, tedy operace modulo 3. Popis implementace lze najít v kapitole 3.2.4.

Bylo naměřeno 100 000 vzorků spotřeby rozdělených do 20 samostatných



Obrázek 5.9: Vzorek spotřeby pro variantu modulo 3



Obrázek 5.10: Průběh korelace pro variantu zabezpečenou pomocí zbytků modulo 3 mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

Tabulka 5.4: Tabulka mediánů vzorků pro každý byte ve variantě s modulo 3

1.	2.	3.	4.	5.	6.	7.	8.
1212	902.5	834.5	652	894	915.5	805	894.5
9.	10.	11.	12.	13.	14.	15.	16.
730.5	779	529	1052.5	561.5	875	1053	748.5

útoků. Průběh korelace mezi 750. - 850. pro první byte 10. rundovního klíče lze vidět na obrázku 5.10. Zeleně zvýrazněný vzorek je pro hodnotu 36_{16}, což je správná hodnota tohoto prvního bytu. Medián nejmenších počtů vzorků nutných pro prolomení 1. - 16. bytu klíče naleznete v tabulce 5.4. Následně byla vybrána největší hodnota z každého měření, která tedy udává, kolik bylo nutných vzorků pro prolomení celého klíče. Z těchto 20 hodnot byl opět vypočítán medián a pro komplementární paritu vychází 2059 s mezikvartilovým rozpětím 945. Tato hodnota mediánu byla podobná té ve variantě s jednoduchou a dvojitou paritou. Velké rozpětí ale udává velký rozdíl mezi nejmenšími a největšími hodnotami.

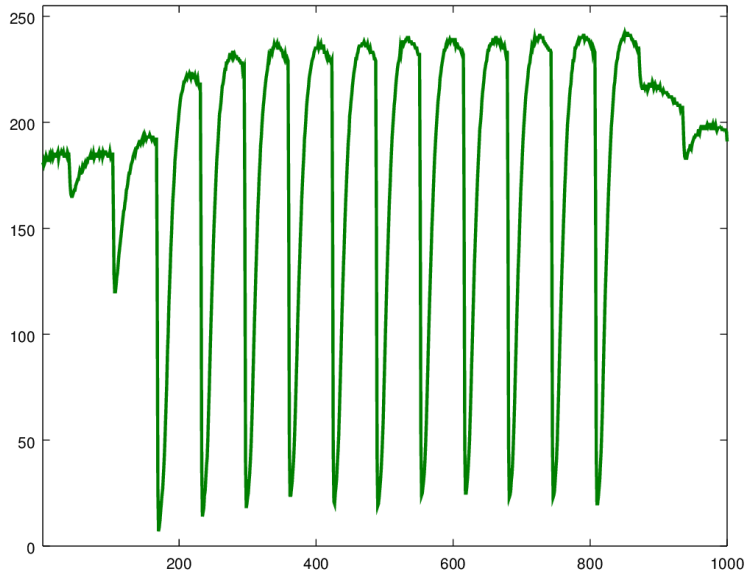
5.3.6 Varianta zabezpečená Hammingovým kódem

Tato varianta obsahuje zabezpečení proti poruchám obvodu výpočtem 4 paritních bitů za využití Hammingova kódu. Každý bit je složen z výpočtu paritní informace z jiných indexů bytu. Popis implementace lze nalést v kapitole 3.2.5.

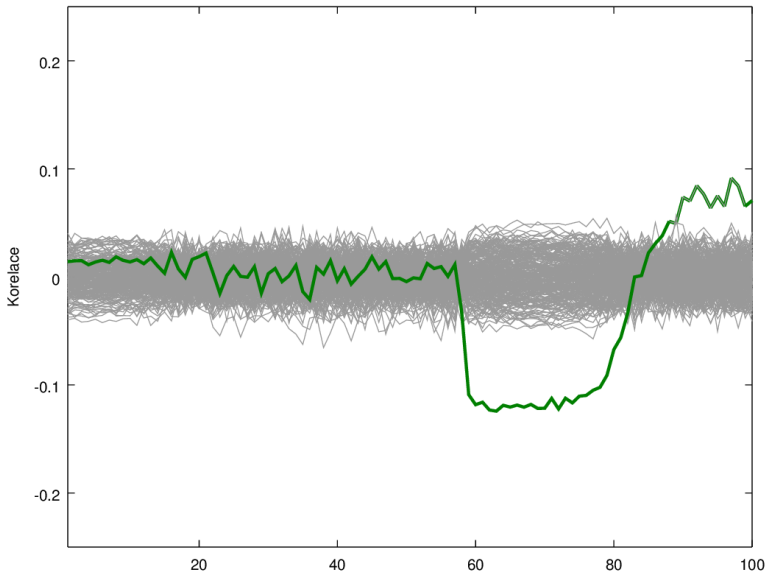
Bylo naměřeno 100 000 vzorků spotřeby rozdělených do 20 samostatných útoků. Průběh korelace mezi 750. - 850. pro první byte 10. rundovního klíče lze vidět na obrázku 5.12. Zeleně zvýrazněný vzorek je pro hodnotu 36_{16}, což je správná hodnota tohoto prvního bytu. Medián nejmenších počtů vzorků nutných pro prolomení 1. - 16. bytu klíče naleznete v tabulce 5.5. Následně byla vybrána největší hodnota z každého měření, která tedy udává, kolik bylo nutných vzorků pro prolomení celého klíče. Z těchto 20 hodnot byl opět vypočítán medián a pro komplementární paritu vychází 1594,5 s mezikvartilovým rozpětím 276. Hodnota výrazně nejmenší ze všech měření, včetně malého mezikvartilového rozpětí.

Tabulka 5.5: Tabulka mediánů vzorků pro každý byte ve variantě s Hammingovým kódem

1.	2.	3.	4.	5.	6.	7.	8.
1140.5	522.5	688.5	917	728.5	960.5	520.5	600.5
9.	10.	11.	12.	13.	14.	15.	16.
897	531	581	751.5	434.5	611.5	634.5	580.5



Obrázek 5.11: Vzorek spotřeby pro variantu s Hammingovým kódem



Obrázek 5.12: Průběh korelace pro variantu s Hammingovým kódem mezi 750. - 850. vzorkem pro první byte 10. rundovního klíče

5.4 Výsledek diferenciální proudové analýzy

Z důvodů neúspěchu prolomení šifry pro základní variantu se tedy výsledné hodnoty nedají s ní porovnat, lze však porovnat jednotlivé varianty mezi sebou a dojít k závěru, zda některá z nich dopomohla k úspěšnějšímu útoku DPA než jiná.

Pro každou variantu tedy proběhlo 20 útoků a mediány počtu nejmenších vzorků nutných pro prolomení celého 10. rundovního klíče lze najít v tabulce 5.6. Každý útok probíhal tak, že se pro každý byte zvlášť výpočítala hranice, kolik počtů vzorků ještě dává správný výstupní byte a která už ne. Z těchto 16 hodnot pro každý byte se vzala ta největší hodnota, která tedy udává počet vzorků pro celý šifrový klíč. Z 20 útoků tedy vzniklo 20 hodnot a z nich je zde uveden medián. Toto se opakovalo celkem pětkrát, tedy pro každou variantu.

Medián je nazývána prostřední hodnota nějakého souboru čísel. V tabulce jsou ještě uvedeny mezikvartilová rozpětí. Pokud medián rozděluje soubor na dvě poloviny, tři kvartily dělí na čtvrtiny. 25% prvků má hodnotu menší než dolní kvantil a 75% prvků hodnotu menší než horní kvantil. Pokud odečteme hodnotu horního a dolního kvantilu, získáme mezikvartilové rozpětí.

Tabulka 5.6: Tabulka mediánů vzorků pro všechny varianty

Varianta	Medián	Rozpětí
Jednoduchá parita	2392.5	698
Dvojitá parita	2210	993
Komplementární parita	1829.5	324
Modulo 3	2059	745
Hammingův kód	1594.5	276

Je tedy jisté, že žádná z variant nemá na průběh útoku tak velký vliv, aby nešel provést. Počet vzorků nutných pro prolomení tohoto designu AES se pohybuje mezi 1000 - 3000, kdy maximální hodnoty pro některá měření dosahovaly i ke 4000. Dále je z výsledků možné, že použití Hammingova kódu pro vypočítání parity, zlehčuje provední útoku DPA, ale nijak zásadní dopad to na výpočet nemá. Ostatní varianty dosahovaly velice podobných výsledků, takže jejich dopad na DPA není prokazatelný.

Kapitola 6

Budoucí práce

Dalším možným pokusem by byl přístup, který uplatňuje Regazzoni v [16]. V jejich práci sice neimplementují celý AES, ale simulují jen jeho část týkající se S-Box transformace. Jejich S-Box je také zabezpečen paritou, avšak ta se nevypočítává až těsně předtím, nýbrž je do celého obvodu přidán drát, ve kterém je signál pro paritu. Navíc jejich S-Box není implementován vyhledávající tabulkou, ale hodnota je počítána pomocí operace xor a příslušných posunů. Tedy navrhnout tento design i pro celý AES.

V jejich výzkumu poté zhodnocují, zda má nějaký vliv na útok pomocí DPA, pokud útočník ví informaci, že v implementaci byly použity detekce na odhalení chyb. Jestli se útok zaměří jen na 8 bitů nebo jestli lze využít dalších přidaných bitů pro úspěšné prolomení.

Dále pro každý správný statistický výzkum je nutné mít dostatečná data, pak je doporučeno naměřit více vzorků pro lepší porovnání výsledků.

Závěr

Tématem této práce bylo, zda má dopad na průběh útoku pomocí diferenciální proudové analýzy, pokud se do hardwarové implementace šifry AES přidá nějaký další obvod, který má za úkol detekovat chyby ve výpočtu.

Nejdříve jsme se seznámili s designem šifry blokové šifry AES, kterou jsme posléze navrhli hardwarovou implementaci na FPGA v jazyce VHDL. Zjistili jsme, že každý kryptografický systém vyměňuje s okolím informace prostřednictvím tzv. postranních kanálů a jedním z takových je kanál proudový. Tento kanál může možný útočník zneužít pro prolomení šifry a zjištění šifrového klíče. Jedním z možných útoků je s pomocí diferenciální proudové analýzy. Dozvěděli jsme se v čem taková analýza spočívá a jaký je jeden ze způsobů, jakým se vyhodnocuje, v našem případě to bylo použití korelačního koeficientu.

Jádrem tématu bylo přidání obvodů na detekci chyb. Vytvořili jsme tedy varianty šifry AES obsahující tyto mechaniky a ty jsme posléze podrobili zkoumání, zda jejich přítomnost ovlivňuje proveditelnost útoku DPA. Útok byl proveden na poslední rundu během operace šifrování a využilo se v něm výpočtu Hammingovy vzdálenosti mezi hodnotou výstupního šifrového textu a hodnotou v registru před operací SubBytes. Měření bylo provedno na desce Spartan 3E Starter Kit.

Obrana proti chybám je založena na přidání informační redundance do obvodu, což je v našem případě místo, kdy dochází k lineární transormaci S-Box. V této práci bylo detekce chyb docíleno tak, že ze vstupního bytu se vypočte vstupní parita a predikce, jakou paritu by měl mít byte výstupní. Po výměně bytu přes vyhledávací tabulku je z něj vypočtena výstupní parita a také predikce hodnoty vstupní. Příslušné hodnoty jsou posléze porovnány, a pokud by některé neseděly, systém hlásí chybu.

Testování správnosti návrhu všech jednotlivých částí šifry AES a všech odvozených variant proběhlo v programu iSim, který prokázal jejich správnou funkčnost.

Následovalo samotné měření spotřeby během šifrování. Proudová spotřeba se změřila pomocí osciloskopu a program k němu dodaný zajistil uložení všech

výsledků.

Následně proběhla diferenciální proudová analýza na změřená data. Výsledky měření byly vyhodnoceny pomocí skriptovacího programu Matlab a pro každý útok byl vypočten medián nejmenšího počtu vzorků nutných pro úspěšné prolomení celého šestnáctibytového šifrovacího klíče.

I přes opakované změny v návrhu AES se bohužel nepodařilo prolomit klíč v základní verzi, narozdíl tedy od variant obsahujících detekci chyb. Zkoumané varianty ale neprojevily výraznější dopad na proveditelnost útoku diferenciální proudovou analýzou, tedy nezhoršují ani nezlepšují její úspěšňost.

Literatura

[1] National Institute of Standards and Technology: *Announcing the Advanced Encryption Standard (AES)*. [cit. 2018-02-10]. Dostupné z: `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`

[2] Paar, C.; Pelzl, J.: *The Advanced Encryption Standard (AES)*. 2010, ISBN 978-3-642-04101-3, s. 87–121. Dostupné z: `https://doi.org/10.1007/978-3-642-04101-3_4`

[3] Daemen, J.; Rijmen, V.: *Rijndael/AES*. 2005, ISBN 978-0-387-23483-0, s. 520–524. Dostupné z: `https://doi.org/10.1007/0-387-23483-7_358`

[4] Daemen, J.; Rijmen, V.: *The Design of Rijndael*. 2002, ISBN 978-3-662-04722-4, s. 43–45.

[5] Kelsey, J.; Schneier, B.; Wagner, D.; aj.: Side channel cryptanalysis of product ciphers. In *Computer Security — ESORICS 98*, 1998, ISBN 978-3-540-49784-4, s. 97–110.

[6] Biryukov, A.; Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In *Advances in Cryptology – ASIACRYPT 2009*, 2009, ISBN 978-3-642-10366-7, s. 1–18.

[7] Kocher, P.; Jaffe, J.; Jun, B.: Differential Power Analysis. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, 1999, ISBN 978-3-540-48405-9, s. 388–397.

[8] Zhou, Y.; Feng, D.: Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. 2005.

[9] Ghosh, S.; Alam, M.; Chowdhury, D. R.; aj.: Effect of side channel attacks on RSA embedded devices. In *TENCON 2007 - 2007 IEEE Region 10 Conference*, 2007, ISBN 978-1-4244-1271-6.

[10] Kolařík, J.: *Postranní kanály.* Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2012, diplomová práce.

[11] Martinásek, Z.; Člupek, V.; Zeman, V.; aj.: Základní metody diferenciální proudové analýzy. Leden 2013: s. 611–624, ISSN 1573-0727, [cit. 2018-02-10]. Dostupné z: https://doi.org/10.1007/s10836-016-5598-9

[12] Brier, E.; Clavier, C.; Olivier, F.: Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ISBN 978-3-540-28632-5, s. 16–29.

[13] Barenghi, A.; Breveglieri, L.; Koren, I.; aj.: Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, 2012, ISSN 0018-9219. Dostupné z: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=6178001

[14] Bertoni, G.; Breveglieri, L.; Koren, I.; aj.: A parity code based fault detection for an implementation of the Advanced Encryption Standard. In *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings.*, 2002, ISSN 1550-5774, s. 51–59, doi:10.1109/DFTVS.2002.1173501.

[15] Dofe, J.; Pahlevanzadeh, H.; Yu, Q.: A Comprehensive FPGA-Based Assessment on Fault-Resistant AES against Correlation Power Analysis Attack. *Journal of Electronic Testing*, 2016: s. 611–624, ISSN 1573-0727. Dostupné z: https://doi.org/10.1007/s10836-016-5598-9

[16] Regazzoni, F.; Breveglieri, L.; Ienne, P.; aj.: *Interaction Between Fault Attack Countermeasures and the Resistance Against Power Analysis Attacks.* Springer Berlin Heidelberg, 2012, ISBN 978-3-642-29656-7, s. 257–272. Dostupné z: https://doi.org/10.1007/978-3-642-29656-7_15

[17] Error detection and correction. 2004, [cit. 2018-02-10]. Dostupné z: http://logos.cs.uic.edu/366/notes/ErrorCorrectionAndDetectionSupplement.pdf

[18] Semrád, O.: *Útok postranními kanály na implementaci algoritmu AES na platformě Xilinx.* Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, bakalářská práce.

[19] Mazur, L.: *Side channel analysis of cryptographic algorithms implementations.* Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, bakalářská práce.

[20] Severyn, J.: *Útoky postranními kanály na implementace kryptografických algoritmů*. Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016, bakalářská práce.

[21] Zimmerhakl, T.: *Implementace AES algoritmu pro FPGA*. Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2015, bakalářská práce.

[22] Natale, G. D.; Flottes, M. L.; Rouzeyre, B.: A Novel Parity Bit Scheme for SBox in AES Circuits. In *2007 IEEE Design and Diagnostics of Electronic Circuits and Systems*, 2007, doi:10.1109/DDECS.2007.4295295.

[23] Buček, J.; Novotný, M.: *Přednášky a cvičení předmětu Bezpečnost a technické prostředky (MI-BHW) [online]*. Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2011-2018, [cit. 2018-02-10]. Dostupné z: `https://edux.fit.cvut.cz/courses/MI-BHW/`

[24] Říha, J.: *Útok postranními kanály na implementaci algoritmu AES na platforme Altera*. Praha, České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, bakalářská práce.

[25] Advanced Serial Port Terminal. 2011, [cit. 2018-02-10]. Dostupné z: `http://www.eltima.com/products/serial-port-terminal`

Příloha **A**

Seznam použitých zkratek

DPA Differential power analysis

AES Advanced Encryption Standard

NIST National Institute of Standards and Technology

DES Data Encryption Standard

SA Simple Analysis

FPGA Field Programmable Gate Array

VHDL VHSIC Hardware Description Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `measurements` ..... adresář s naměřenými daty
  - `AES` ..... základní verze
  - `commplementary` ..... komplementární parita
  - `double` ..... dvojitá parita
  - `hamming` ..... Hammingův kód
  - `modulo` ..... modulo 3
  - `parity` ..... jednoduchá parita
- `src` ..... zdrojové kódy jednotlivých variant
  - `AES` ..... základní verze
  - `commplementary` ..... komplementární parita
  - `double` ..... dvojitá parita
  - `hamming` ..... Hammingův kód
  - `matlab` ..... skripty provádějící DPA v Matlab
  - `modulo` ..... modulo 3
  - `parity` ..... jednoduchá parita
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX