ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Útok rozdílovou odběrovou analýzou na implementaci algoritmu AES na platformě Xilinx
Student:	Ondřej Semrád
Vedoucí:	Dr.-Ing. Martin Novotný
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Seznamte se s metodami útoků na implementace kryptografických algoritmů prostřednictvím tzv. postranních kanálů. Zaměřte se zejména na rozdílovou odběrovou analýzu (differential power analysis, DPA).

Naučte se používat tuto metodu pro implementaci algoritmu AES na čipové kartě (Smart Card).

Algoritmus AES implementujte v hardwaru, konkrétně na programovatelném hradlovém poli (FPGA) a poté prozkoumejte možnosti aplikace metody DPA i na hardwarovou implementaci AES.

Po konzultaci s vedoucím práce zvolte některou variantu algoritmu AES zabezpečenou proti poruchám a prozkoumejte možnosti aplikace metody DPA i na tuto variantu.

Bude-li to možné, porovnejte odolnost základní a zabezpečené varianty AES vůči útoku prostřednictvím DPA, například porovnáním počtu průběhů spotřeby nezbytných pro správnou identifikaci klíče.

Ve své práci se zaměřte na technologii firmy Xilinx. Vhodnou platformou je přípravek Spartan 3E Starter Kit.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. ledna 2017

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

Bakalářská práce

Útok rozdílovou odběrovou analýzou na implementaci algoritmu AES na platformě Xilinx

Ondřej Semrád

Vedoucí práce: Dr.-Ing. Martin Novotný

16. května 2017

Poděkování

Děkuji Martinu Novotnému za lidský přístup a obrovskou trpělivost při vzniku této práce. Děkuji Jiřímu Bučkovi za cenné rady a pomoc při měření. Děkuji Janu Říhovi za ochotu mi pomoci vždy, když jsem si s něčím nevěděl rady. Děkuji rodině za podporu a lásku.

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 16. května 2017 .....

Odkaz na tuto práci

Semrád, Ondřej. *Útok rozdílovou odběrovou analýzou na implementaci algoritmu AES na platformě Xilinx*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá možnostmi aplikace metody rozdílové odběrové analýzy (DPA) na implementaci algoritmu AES na FPGA Spartan-3E od firmy Xilinx. V rámci práce byly vytvořeny dvě rozdílné hardwarové implementace šifry AES v jazyce VHDL a skript realizující DPA v programu Mathematica. Dále byla vytvořena obálka realizující komunikaci modulu AES s počítačem pomocí sériové linky. Do této obálky bylo následně vloženo osm různých variant šifry AES – tři základní a pět zaměřených na odolnost proti poruchám. U jedné základní varianty a všech zabezpečených byl určen minimální počet průběhů spotřeby, který ještě stačí k prolomení klíče. Na základě porovnání těchto výsledků bylo zjištěno, že časová, prostorová a ani informační redundance významně neovlivňuje odolnost implementace proti DPA.

Klíčová slova rozdílová odběrová analýza, DPA, bezpečnost, FPGA, AES, Rjindael, Xilinx

Abstract

We explored the possibilities of application of Differential Power Analysis (DPA) on the implementation of AES algorithm on the FPGA Spartan-3E by Xilinx. We created two different hardware implementations of the AES cipher in VHDL language, a script implementing the DPA method in the Mathematica software and a wrapper implementing the communication between an AES module and a computer using a serial line. We inserted eight different versions of AES cipher inside the wrapper – five versions with safety measures and three basic versions without any safety measures. We compared the resistance of basic variant with the fault tolerant ones by computing the minimal number of power traces needed for breaking the correct key for each variant. We discovered that the safety measures (hardware redundancy, time redundancy and information redundancy) had minimal influence on the resistance against DPA.

Keywords differential power analysis, DPA, security, safety, FPGA, AES, Rjindael, Xilinx

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Zařízení obsahující a využívající implementaci šifrovacího algoritmu jsou dnes všude kolem nás. Běžným příkladem jsou jsou takzvané čipové karty, které se vyskytují například ve formě kreditních a debetních karet, SIM karet, přístupových karet do satelitních přijímačů nebo karet pro přístup do školy či zaměstnání. Tato zařízení ale obvykle využívají softwarovou implementaci šifrovacího algoritmu. Hardwarová implementace se použije tam, kde je třeba vysoká rychlost či nízká spotřeba, případně kombinace obojího. Jako konkrétní příklad lze uvést hardwarovou implementaci šifry AES v dnešních procesorech architektury x86. Dalším příkladem jsou moduly pro šifrování různých úložišť jako USB disků či pevných disků v reálném čase.

„Když Kocher a kolektiv ukázali v roce 1998, že útoky odběrovou analýzou dokáží účinně odhalit tajemství čipových karet, víra v bezpečnost kryptografických zařízení byla otřesena“[3]. Je tedy nezbytné se zabývat bezpečným hardwarovým návrhem šifer ku zvýšení jejich odolnosti proti tomuto typu útoku.

Ve své práci se budu zabývat útoky odběrovou rozdílovou analýzou na hardwarové implementace šifry AES v programovatelném hradlovém poli. Vystřídám několik různých implementací AES a dle úspěšnosti útoků porovnám odolnost těchto variant.

Cílem této práce je prozkoumat odolnost variant hardwarové implementace šifry AES. V rámci práce budu implementovat šifru AES na programovatelném hradlovém poli a provedu útok na tuto implementaci prostřednictvím rozdílové odběrové analýzy. Následně provedu obdobná měření pro varianty AES upravené pro zvýšení odolnosti proti poruše a porovnám jejich odolnost se základní variantou AES. Ze zjištěných měření vyplyne, zdali má použití spolehlivostních architektur nějaký vliv na odolnost obvodu proti útoku rozdílovou odběrovou analýzou.

Kapitola 1

Úvod do problematiky

V této kapitole představím algoritmus AES, metodu útoku pomocí rozdílové odběrové analýzy a nakonec bakalářské práce, na které navazuji.

1.1 Advanced Encryption Standard

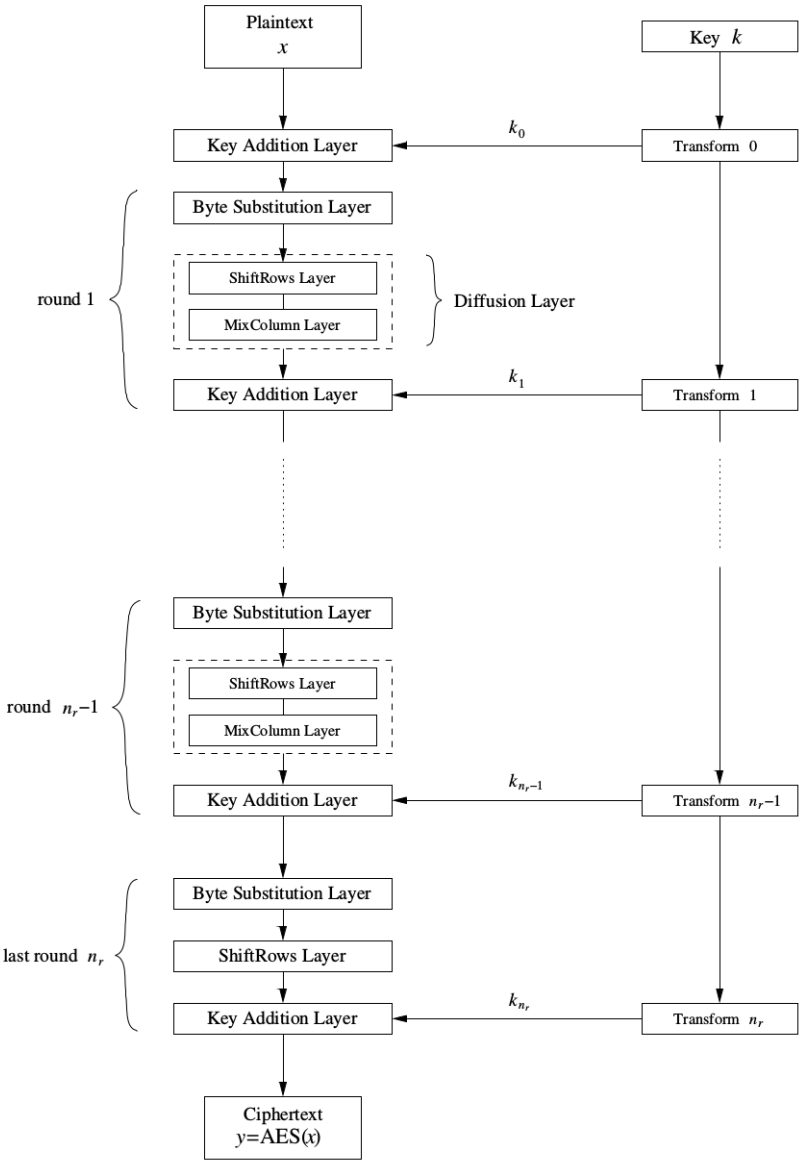
AES je šifrovací standard zavedený roku 2001 americkým Národním institutem standardů a technologie (NIST)[4], který používá blokovou šifru Rjindael s délkou bloku 128 bitů a délkou klíče 128, 192 nebo 256 bitů. V následujícím textu vás stručně seznámím s variantou šifry AES, kterou budu v celé své práci pro jednoduchost používat – jedná se o variantu, která používá klíč o délce 128 bitů. Podrobnější informace o šifre AES lze nalézt v [1], odkud čerpám i v následujících odstavcích.

1.1.1 AES 128

Tato verze používá klíč o velikosti 128 bitů a obsahuje deset iterací neboli rund, přičemž v každé rundě kromě poslední se manipuluje s blokem dat pomocí čtyř funkcí. Pro každou rundu existuje jiný rundovní klíč, kde každý je odvozený od předchozího rundovního klíče pomocí zvláštní funkce. Šifrování algoritmem AES je přehledně znázorněno v obrázku 1.1.

Byte substitution Každý z šestnácti bytů vstupu funkce Byte substitution je nahrazen jiným bytem dle tabulky 1.1. Byte substitution je v šifře AES jediná nelineární funkce, tedy platí vzorec $\mathrm{BS(A)} + \mathrm{BS(B)} \neq \mathrm{BS(A + B)}$, což šifře poskytuje odolnost proti známým analytickým útokům[1].

ShiftRows ShiftRows je prostá permutace bytů uvnitř 128 bitového vstupního bloku. Pro popis manipulace s daty uvnitř šifry AES je výhodné použít matici 4x4, znázorněnou v tabulce 1.2, která se nazývá *stav* nebo



Obrázek 1.1: Blokové schéma algoritmu AES – převzato z [1].

stavová matice[4]. ShiftRows posouvá data cyklicky v prvním řádku stavové matice o nula pozic, v druhém řádku o jednu pozici, v třetím o dvě a ve čtvrtém o tři místa doleva, čímž vznikne stavová matice výstupních dat funkce ShiftRows, znázorněná v tabulce 1.3.

	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	dvd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabulka 1.1: Tabulka hodnot SubBytes pro byte xy

B1	B5	B9	B13
B2	B6	B10	B14
B3	B7	B11	B15
B4	B8	B12	B16

Tabulka 1.2: Stavová matice

B1	B5	B9	B13
B6	B10	B14	B2
B7	B11	B15	B3
B8	B12	B16	B4

Tabulka 1.3: Stavová matice po operaci Shift rows

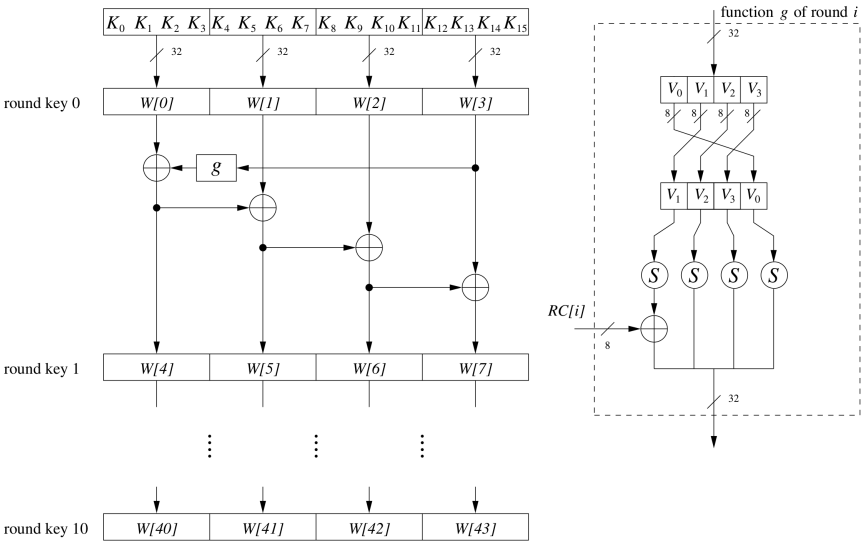
MixColumns Funkce MixColumns spolu s ShiftRows zajišťuje difúzi dat, tedy že se změna jednoho bitu vstupních dat projeví ve změně celého výstupu. Ta je v šifře AES natolik propracovaná, že již po třech rundách závisejí všechny byty stavové matice na všech bytech vstupu. Každý slou-

pec stavové matice je v MixColumns vynásoben maticí dle vzorce 1.1[1].

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{pmatrix} \tag{1.1}$$

Veškeré matematické operace v šifře AES probíhají v tělese $GF(2^8)$ s nerozložitelným polynomem $x^8 + x^4 + x^3 + x + 1$, vizte [1].

Key schedule Funkce Key schedule má na vstupu 128 bitový klíč a opakovaným aplikováním stejné operace z něj postupně vytvoří dalších deset rundovních klíčů. Každý rundovní klíč je jiný a je použit v odpovídající rundě ve funkci Key addition (vizte obrázek 1.1), která nedělá nic jiného, než prosté přixorování rundovního klíče k datům. Key schedule pracuje s bloky dat o délce 32 bitů, nazývané *slova*[1]. Celá funkce je přehledně znázorněna na obrázku 1.2.



Obrázek 1.2: Key schedule šifry AES 128 – převzato z [1] a upraveno.

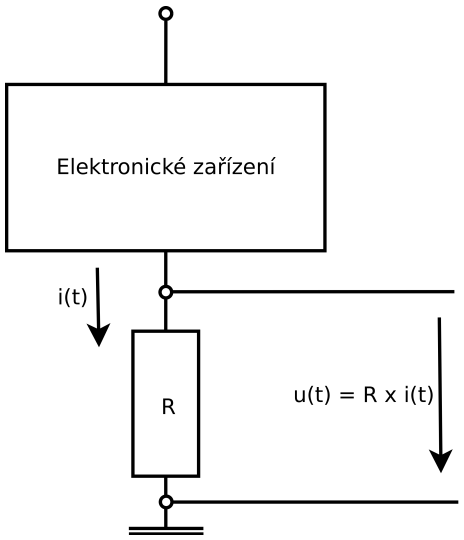
1.2 Rozdílová odběrová analýza

Rozdílová odběrová analýza, dále jen DPA (*Differential power analysis*), je metoda útoku na konkrétní zařízení implementující šifru pomocí postranních kanálů a byla poprvé uvedena v roce 1999 v [3]. Útok pomocí postranních kanálů nevyužívá chyby v obecném návrhu šifry jako takové, ale vlastností zařízení, na kterém je šifra implementovaná. *„Analýza postranními kanály může*

být použita k získání tajného klíče, například, měřením elektrického příkonu procesoru, který s tajným klíčem pracuje“ [1]. *„Cílem DPA útoku je odhalení tajného klíče kryptografických zařízení pomocí velkého množství vzorků spotřeby, které byly získány zatímco zařízení šifrovalo či dešifrovalo rozdílné datové bloky. [...] existuje obecná strategie, kterou používají všechny DPA útoky“* [3]. Ve své práci nebudu používat původní metodu DPA, tak jak je popsána v [3], ale zaměřím se na obecnější verzi, která využívá matematického výpočtu korelačních koeficientů, a proto se také nazývá *Correlation Power Analysis*, CPA. Tato verze byla představena v [5]. Zatímco původní verze DPA zjišťovala klíč v zařízení po jednotlivých bitech, tato jej určuje po bytech. Základní myšlenka je ale pořád stejná – okamžitý odběr zařízení závisí na právě zpracovávaných datech. V následujících odstavcích vám představím průběh metody DPA pro zjištění prvního bytu klíče, obdobný postup lze nalézt v [2]:

1. **Změříme průběh spotřeby během šifrování.**

   K měření spotřeby připojím pomocný rezistor na drát zajišťující napájení 0V zařízení, tedy zem, vizte obrázek 1.3. Osciloskopem pak měřím napětí na tomto rezistoru během probíhajícího šifrování zařízení. Proud protékající zařízením protéká i tímto rezistorem a je dle Ohmova zákona přímo úměrný naměřenému napětí. Šifrování provedeme mnohokrát, pokaždé pro jiný otevřený text. Získáme tak matici vzorků o velikosti $N \times K$, kde K je počet vzorků získaných v jednom šifrování a N je počet šifrování, ale také zároveň počet otevřených i šifrových textů, které mám nyní k dispozici.



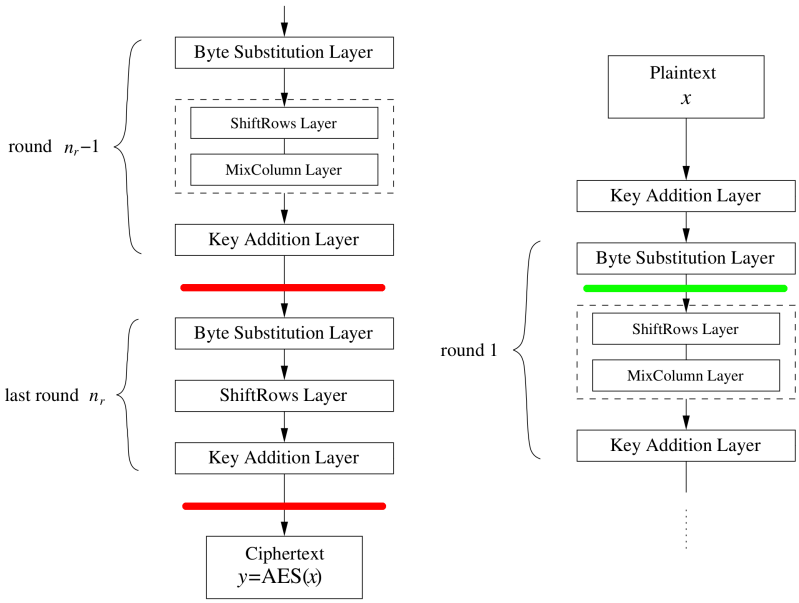
Obrázek 1.3: Schéma způsobu měření napětí na rezistoru připojeném sériově za zařízení.

$$\mathbb{M}_1 = \begin{array}{c} \\ OT_1 \\ OT_2 \\ \vdots \\ OT_N \end{array} \begin{array}{c} \begin{array}{cccc} S_1 & S_2 & \cdots & S_K \end{array} \\ \begin{pmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,K} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ V_{N,1} & V_{N,2} & \cdots & V_{N,K} \end{pmatrix} \end{array} \tag{1.2}$$

Obrázek 1.4: Matice vzorků. V jednom řádku se vždy nachází K jednotlivých vzorků spotřeby naležejících k patřičnému otevřenému textu.

2. **Zvolím hodnotu, která závisí na datech a klíči a následně sestavím matici hypotéz o hodnotě uvnitř šifry pro všechny průběhy a všechny možné první byty klíče.**

   Vhodné hodnoty lze vidět na obrázku 1.5. Zelená hodnota závisí na



Obrázek 1.5: Vhodná místa k DPA útoku – červeně místa pro útok s použitím Hammingových vzdáleností, zeleně pro útok s použitím Hammingovy váhy. Vlevo útok na poslední rundu, vpravo na první rundu. Převzato z [1] a upraveno.

   vstupních datech a klíči, pro přehlednost jí budu značit H. První byte vstupních dat znám, funkci Byte Substitution jsem schopen provést. To, co je neznámé, je první byte klíče – ten ale může nabývat pouze 256 různých hodnot, proto vytvořím matici hypotéz $256 \times N$, kde N je počet

průběhů, které jsem naměřil. Pro vytvoření této matice musím zjistit, jakých hodnot mohly nabývat jednotlivé byty H – byly určitě ovlivněny jedině prvním bytem tajného klíče. Jako příklad si představte situaci, kde první byte prvního otevřeného textu je 0x01. Postupně otestuji všechny klíče, začnu s 0x00. Nejprve provedu funkci Key Addition, tedy k 0x01 přixoruji 0x00 a tím získám hodnotu 0x01. Dále provedu operaci Byte Substitution, čímž získám hodnotu 0x7c, což je mé první možné H. Dále pokračuji s klíčem 0x01 až po hodnotu 0xFF. Tím získám všechny H, které mohly nastat při šifrování prvního bytu prvního otevřeného textu. Stejnou proceduru provedu pro všechny první byty otevřeného textu, čímž získám matici hypotéz o velikosti $N \times 256$.

$$\mathbb{M}_2 = \begin{array}{c} \\ B1_1 \\ B1_2 \\ \vdots \\ B1_N \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & \cdots & 255 \end{array} \\ \begin{pmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,256} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,256} \\ \vdots & \vdots & \ddots & \vdots \\ V_{N,1} & V_{N,2} & \cdots & V_{N,256} \end{pmatrix} \end{array} \tag{1.3}$$

Obrázek 1.6: Matice hypotéz. Řádky odpovídají příslušnému prvnímu bytu otevřeného textu B1, sloupce pak příslušné hodnotě prvního bytu klíče.

3. **Pomocí modelu spotřeby vypočteme matici hypotéz o spotřebě pro všechny průběhy a hodnoty, kterých mohl první byte klíče nabývat.**

   V tomto kroku se z informace o datech, se kterými se v zařízení v nějaký okamžik pracovalo, zjišťuje spotřeba, kterou zařízení v tom stejném okamžiku mělo. Matici hypotéz o spotřebě lze vytvořit pomocí Hammingovy váhy, tedy tak, že každou hodnotu v matici hypotéz nahradím číslem 0 - 8, dle počtu jedniček v binární reprezentaci původní hodnoty. Druhou možností je použití Hammingových vzdáleností. Ty získáme tak, že sledujeme, kolik jedniček se změnilo na nuly, a obráceně, někde uvnitř zařízení, typicky v registru či na sběrnici. Proto je na obrázku 1.5 zvýrazněno místo pro útok Hammingovými vzdálenostmi dvakrát – hardwarová implementace šifry AES s jedním registrem může ukládat právě tyto dvě hodnoty do jednoho registru, a proto je důležité je určit a vypočítat jejich Hammingovu vzdálenost. V první rundě by tento útok byl mnohem složitější, neboť jedna z hodnot se nachází až za funkcí MixColumns, která ale promíchá jednotlivé byty mezi sebou a já jsem schopen určit jen jeden byte na jejím vstupu – v opačném případě bych musel tipovat více bytů klíče, což by celý proces ohromně zkomplikovalo. Je ovšem možné použít hodnoty, které se nacházejí před a za operací Byte Substitution.

Užití Hammingových vah nemusí fungovat pro hardwarové implementace šifry, jak zjistil ve své bakalářské práci Jan Severyn[6]. Vhodnější je pak použití Hammingových vzdáleností, jak se píše například v [7]. Užití Hammingovy vzdálenosti by mělo být přesnější i pro softwarové implementace šifry, například na zařízení typu SmartCard, neboť popisuje i spotřebu těchto zařízení obvykle lépe [5]. Hammingova váha funguje poměrně dobře, neboť je to vlastně Hammingova vzdálenost dané hodnoty od nulové hodnoty[5], a proto v průměru předpoví správně polovinu hodnot Hammingovy vzdálenosti[7].

$$\mathbb{M}_3 = HW(\mathbb{M}_2) \tag{1.4}$$

$$\mathbb{M}_3 = HD(\mathbb{M}_2, \mathbb{M}'_2) \tag{1.5}$$

Obrázek 1.7: Matice hypotéz o spotřebě se určí prostým nahrazením hodnot v matici hypotéz jejich Hammingovými váhami. V případě Hammingových vzdáleností bych musel mít matice hypotéz dvě.

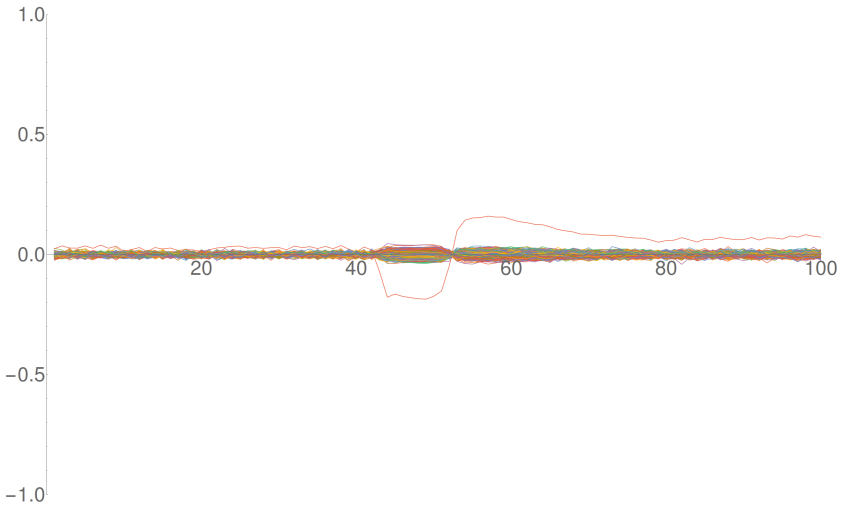
4. **Statisticky vyhodnotíme, která hypotéza (klíč) nejlépe odpovídá skutečně naměřené spotřebě pro každý okamžik měření.**

   Nyní mám matici vzorků, jejíž jednotlivé sloupce odpovídají vzorkům získaným postupně v čase. Bohužel ale neznám přesný čas, kdy obvod pracoval s H. Mám ale matici hypotéz o spotřebě, jejíž každý sloupec odpovídá jednomu možnému prvnímu bytu klíče. Někde v matici vzorků by se měl nacházet sloupec, jehož průběh hodnot odpovídá průběhu hodnot některého sloupce v matici hypotéz o spotřebě. Tyto dva sloupce tedy musejí mít vysokou korelaci. Vypočítám proto korelaci pro všechny dvojice sloupců, které lze v těchto maticích vytvořit, a získám výslednou matici $256 \times K$, kde K je počet vzorků získaných během jednoho šifrování. V této matici najdu maximální hodnotu X. Řádek, v němž se X nachází je s největší pravděpodobností výsledným prvním bytem klíče.

Pro prolomení zbylých patnácti bytů klíče stačí nastíněný postup zopakovat od kroku dva pro každý byte zvlášť.

1.3 Současný stav řešení problematiky

Má práce navazuje na bakalářskou práci Jana Severyna *„Útoky postranními kanály na implementace kryptografických algoritmů“*[6], Lukáše Mazura *„Side Channel Analysis of Cryptographic Algorithms Implementations.“*[8] a Tomáše Zimmerhakla *„Implementace AES algoritmu pro FPGA“*[9].



Obrázek 1.8: Příklad průběhů korelačních koeficientů v čase. Každý řádek výsledné matice je zde vyobrazen jako jedna spojitá linka. Vítěz je v tomto případě jasně identifikovatelný.

1.3.1 Jan Severyn

Jan Severyn nejprve prolomil AES na čipové kartě útokem pomocí Hammingových vah. Následně stejným způsobem útočil na hardwarovou implementaci šifry AES na FPGA, ale nebyl úspěšný. Pokusil se proto zlepšit podmínky útoku snížením hodinové frekvence FPGA, odstraněním blokujících kondenzátorů z desky a změnou zdroje napětí ze síťového adaptéru, který do desky vysílal rušení, na olověný akumulátor, ale ani přesto nebyl v útoku úspěšný.

1.3.2 Lukáš Mazur

Lukáš Mazur nejprve postupoval stejně, jako Jan Severyn, později pak pokračoval v prozkoumávání dalších možností – upravil design své implementace šifry AES, zaútočil na poslední rundu pomocí Hammingových vah, ale nebyl úspěšný. Po konzultaci s Priv.-Doz. Dr. Amir Moradi místo diferenciální sondy použil AC předzesilovač PA 303 BNC od firmy Langer EMV-Technik[10] (se zesílením 30 dB) a také zapnul na osciloskopu omezení šířky pásma. Díky těmto vylepšením byl útok na FPGA popré úspěšný. Následně bylo ještě provedeno několik dalších měření, která odhalila, že odstranění kondenzátorů a použití olověných akumulátorů skutečně snižuje počet vzorků, které je třeba naměřit, aby byl útok úspěšný.

1.3.3 Tomáš Zimmerhakl

Tomáš Zimmerhakl se ve své práci zabýval návrhem různých variant šifry AES 128. Kromě základní verze na deset taktů a speciální kombinační varianty na jediný takt se ve své práci zaměřil na vývoj spolehlivostních variant šifry AES, které jsou odolné proti selhání v důsledku nepříznivého prostředí či stárnutí zařízení. Takovou odolnost získávají buď s využitím prostorové rendundance, kdy se některá část šifrovacího algoritmu provádí uvnitř zařízení paralelně vícekrát a následně se porovnají výsledky. Další možností je časová redundance, kdy se část algoritmu opakuje sekvenčně vícekrát, výsledky se ukládají a opět se porovnají. Poslední možnost je informační redundance. Tomáš Zimmerhakl přidal před a za operaci Byte Substitution prediktory parity a generátory parity. Porovná se predikce před SubBytes s výslednou paritou za SubBytes, stejně tak zpětná predikce za SubBytes s určenou paritou před SubBytes. Ve své práci zaútočím na tyto varianty šifry a porovnám jejich odolnost proti DPA podle počtu vzorků nezbytných k úspěšnému získání tajného klíče v zařízení v porovnání se základní variantou, která posloužila jako základ všech spolehlivostních variant.

Kapitola 2

Realizace

V rámci své práce jsem vytvořil různé varianty šifry AES a další programové vybavení. V následující kapitole je podrobně popíši. Jedná se o skripty, které implementují metodu DPA. Dále jde o tři varianty šifry AES – jedna v jazyce C, určená pro čipovou kartu, a dvě v jazyce VHDL, určené pro FPGA. Nakonec popíši obě varianty obálky, které sloužily jako prostředník mezi modulem AES a modulem realizujícím komunikaci přes sériovou linku.

Hardwarové varianty šifry AES a obálka, realizující prostředníka mezi šifrou AES a modulem sériové linky, byly plánovány pro desku Spartan-3E Starter Kit Board[11] osazenou FPGA Xilinx Spartan-3E 500, kterou jsem se rozhodl po konzultaci s vedoucím používat.

2.1 Skript implementující DPA v programu Mathematica

Pro realizaci útoků pomocí DPA jsem používal program Mathematica. Vytvořil jsem několik variant skriptu lišících se zejména použitým modelem spotřeby. Jedná se o skripty implementující útok pomocí Hammingových vah na první a poslední rundu, dále útok pomocí Hammingových vzdáleností na poslendí rundu a nakonec skript, který jsem použil pro útok na varianty šifry Tomáše Zimmerhakla. U všech variant používám k nahrávání dat lehce upravený kód, který lze nalézt na stránkách kurzu Bezpečnost a technické prostředky[2]. K němu jsem přidal načtení tabulky sbox, inverzní sbox a rc, které slouží k výpočtům a nahrávají se ze souborů *sbox.txt*, *isbox.txt* a *rc.txt* ve složce se skriptem. Proměnná workingDir určuje složku, v níž se nachází výstup programu SC Power Measurements, tedy *traces.bin*, *plaintext.txt*, *ciphertext.txt* a *traceLength.txt*. V této složce se též vytvoří podsložky grafy a korelacni_matice, do kterých se ukládá výstup výpočtů.

Samotný výpočet korelací probíhá postupným vytvořením matice hypotéz o spotřebě a následným voláním funkce Correlation[namerena_spotreba, hy-

poteza]. Výsledná korelační matice je uložena do souboru matX.dat do složky $workingDir/korelacni_matice, kde X je číslo bytu, který zrovna prolamuji. Tuto funkcionalitu jsem do skriptu přidal z toho důvodu, že výpočet korelací pro vyšší počet, například 100000, vzorků spotřeby trval přibližně dvě hodiny a bylo proto nezbytné si výsledek výpočtu uchovat pro možnost jeho rychlého znovunačtení. Do složky $workingDir/korelacni_matice se též ukládá soubor info.txt, ve kterém je uložen počátek čtení vzorku, délka a celkový počet vzorků.

Do složky $workingDir/grafy jsou dále ukládány grafy korelací pro všech 256 hypotéz bytu klíče. Při úspěšném útoku by v každém grafu měl být jasně patrný správný byte klíče, tak jako na obrázku 1.8.

Výsledný klíč je vždy uložen do souboru *key.txt* ve složce se skriptem v hexadecimální podobě bez mezer.

Při porovnávání rychlosti mého skriptu a toho, který používá Jan Říha, který pracuje na stejném zadání práce, ale na platformě Altera[12], jsme zjistili, že jeho skript v programu Matlab prolomí 5000 průběhů, každý o délce 1000 vzorků, za necelou sekundu, zatímco tento můj v Mathematice stejnou úlohu počítá přibližně dvacet minut. Nejvíce času zabere vestavěná funkce Correlation[], proto s tímto faktem nelze nic udělat. Nemohu proto Mathematicu pro tento typ úlohy do budoucna doporučit.

2.1.1 Útok pomocí Hammingových vah na první rundu

Tento skript se nachází na dvd ve složce /src/mathematica/h_vahy_prvni_runda.nb a implementuje útok pomocí Hammingových vah na první rundu, který jsem použil při útoku na čipovou kartu. Model spotřeby se vytváří k datům, která se nacházejí za první operací Byte Substitution.

2.1.2 Útok pomocí Hammingových vah na poslední rundu

Tento skript se nachází na dvd ve složce /src/mathematica/h_vahy_posledni_runda.nb a implementuje útok pomocí Hammingových vah na poslední rundu, který jsem použil při útoku na čipovou kartu. Model spotřeby se vytváří k datům, která se nacházejí před poslední operací Byte Substitution.

2.1.3 Útok pomocí Hammingových vzdáleností na poslední rundu

Tento skript se nachází na dvd ve složce /src/mathematica/h_vzdalenosti_posledni_runda.nb a implementuje útok pomocí Hammingových vzdáleností na poslední rundu, který jsem použil při útoku na čipovou kartu i při útoku na FPGA. Model spotřeby se vytváří k dvojici dat – první odpovídá výslednému šifrovému textu, druhé se nachází před poslední operací Byte Substitution. V tomto skriptu je navíc část, která přepočte nalezený desátý rundovní klíč na originální první klíč, který se snažím prolomit.

2.1.4 Útok pomocí Hammingových vzdáleností na poslední rundu – automatický skript pro 50 měření u variant šifry Tomáše Zimmerhakla

Tento skript se nachází na dvd ve složce /src/mathematica/h_vzdalenosti-_posledni_runda_50.nb. Pro porovnání spolehlivostních variant šifry AES Tomáše Zimmerhakla jsem původní skript implementující útok pomocí Hammingových vzdáleností na poslední rundu dále upravoval. U každé jeho šifry jsem změřil 100000 průběhů spotřeby, a jelikož nikde nebylo třeba více než 2000 průběhů k úspěšnému prolomení, tak jsem naměřená data rozdělil na 50 nezávislých segmentů. Útok na jeden segment probíhá následovně: vypočítám korelační koeficienty za použití 2000 průběhů. Buďto provedu výpočet nad kompletním průběhem spotřeby, nebo pouze nad časovým úsekem, ve kterém se pracovalo s hodnotou, pro kterou vytvářím model spotřeby. Dále již útočím v každém průběhu pouze na jediný vzorek – ten, který měl nejvyšší korelaci při prvním útoku. Počet průběhů snižuji po jednom z 2000 až na hodnotu, při které nezvítězí správný byte klíče, a předchozí hodnotu si zapamatuji jako výsledný počet průběhů, při jehož použití ještě úspěšně prolomím správný byte klíče. V implementaci používám funkci SearchForMin, která provádí hledání minima a je předkompilovaná, čímž jsem dosáhl zrychlení přibližně o 50%.

2.2 AES pro čipovou kartu

Pro seznámení se s šifrou AES a metodou DPA jsem implementoval tuto šifru v jazyce C. Implementaci jsem pak vložil do jednoduchého operačního systému BHW-SOSSE, který se používá v kurzu MI-BHW[2], a je zjednodušenou verzí operačního systému pro chytré karty SOSSE. Implementace se na dvd nachází ve složce /src/AES_c/main.c, nebo již vložená do BHW-SOSSE v /src/AES-_c/BHW_SOSSE.

2.3 AES pro FPGA

Vytvořil jsem dvě varianty šifry AES. První verze, AES01, neodpovídala implementaci, na kterou je útok pomocí DPA s využitím Hammingových vzdáleností na poslední rundu myšlený, proto jsem jí později přepracoval tak, aby se její rundovní registr nacházel mezi operacemi Key Addition a Byte Substitution, což mělo vést k jejímu snazšímu prolomení – tím vznikl AES02. S oběmi verzemi vás v následujících sekcích seznámím.

2.3.1 AES01

V rámci své práce a zároveň v rámci kurzu Praktika v návrhu číslicových obvodů jsem implementoval šifru AES na FPGA v jazyce VHDL. Implementace se na dvd nachází ve složce /src/AES_vhdl/AES01. Nejprve jsem vytvořil

dle zadání verzi, která přijímá otevřený text postupně po osmi bitech, které jsou zadány na přepínačích přípravku Digilent Basys2, a výsledek zobrazuje na displayi po dvou bytech. Tuto verzi jsem pak upravil do obecnější podoby, která přijímá celý otevřený text najednou a celý šifrový text najednou i vrací – je ovšem nezbytné jí spojit s jinou částí, která řeší příjem a odesílání dat například pomocí sériové linky. Dále jsem po konzultaci s vedoucím práce přidal možnost nahrát jiný klíč.

Samotný návrh jsem rozdělil standardně na datovou cestu a řadič.

2.3.1.1 Rozhraní

Kód 2.1: Rozhraní AES01

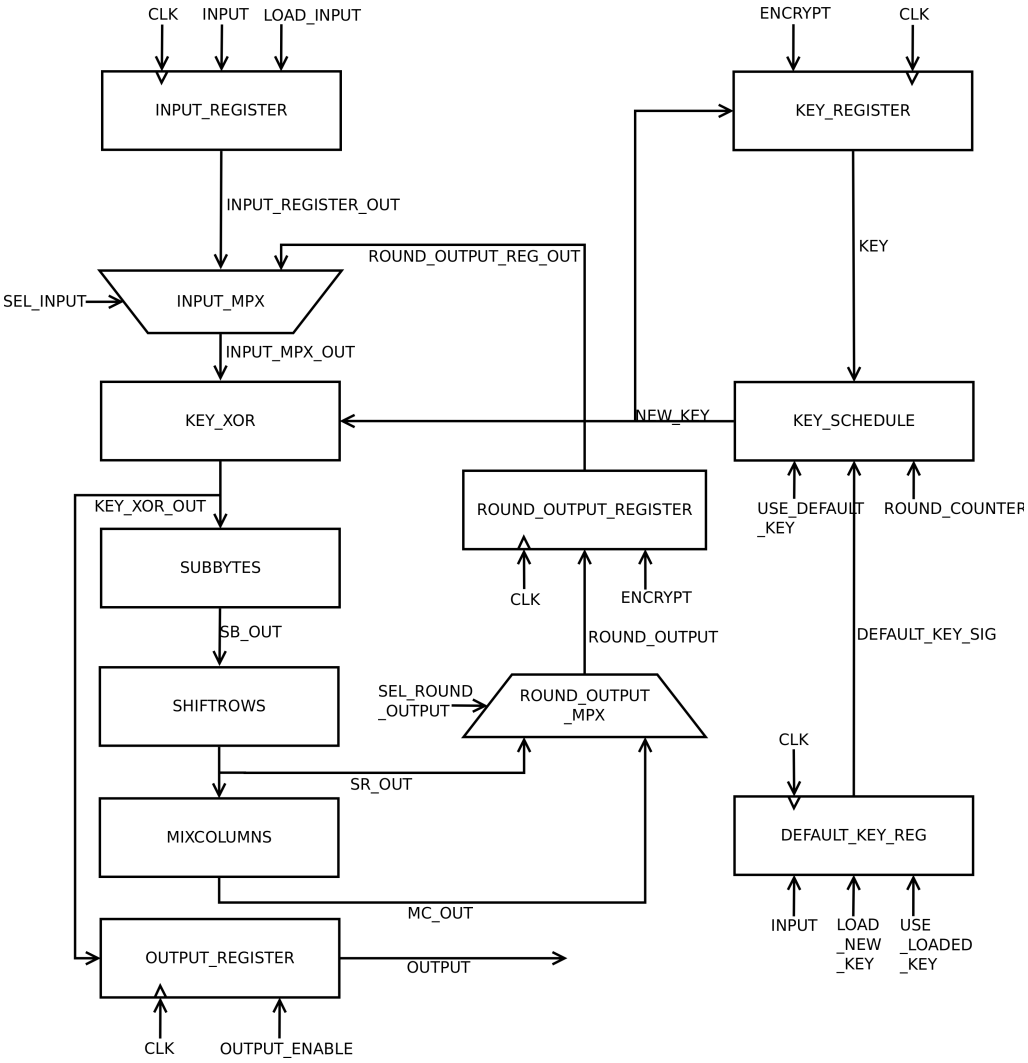
```
entity AES01 is
  port (
    CLK               : in  std_logic;
    RESET             : in  std_logic;
    START_ENCRYPTION  : in  std_logic;
    START_KEY_LOAD    : in  std_logic;
    INPUT             : in  std_logic_vector(127 downto 0);
    OUTPUT            : out std_logic_vector(127 downto 0);
    CT_READY          : out std_logic
    );
end AES01;
```

Na signál INPUT je přiveden otevřený text, na signálu OUTPUT bude po dokončení šifrování šifrový text. Signál START_ENCRYPTION dává najevo, že se má začít šifrovat. Signál START_KEY_LOAD říká, že je na signálu INPUT místo otevřeného textu nový klíč, a modul AES si jej má uložit a ponechat. Signál CT_READY je aktivní po dobu jednoho hodinového taktu, když je šifrování dokončeno a na signálu OUTPUT je šifrový text. Šifrový text na signálu OUTPUT zůstává, dokud není nahrazen dalším.

2.3.1.2 Datová cesta

Datová cesta je zobrazena na obrázku 2.1.

- **INPUT_REGISTER** – Registr, který načte vstupní data.
- **INPUT_MPX** – Multiplexor, který před první rundou nastaví na výstup data ze vstupního registru a v dalších rundách z rundovního registru.
- **KEY_XOR** – Prosté přixorování signálu NEW_KEY k signálu INPUT_MPX_OUT.



Obrázek 2.1: AES01 – datová cesta

- **SUBBYTES** – Funkce SubBytes je řešena tabulkou S, která je uložena v souboru CONSTANTS.vhd.
- **SHIFTROWS** – Funkce ShiftRows je v hardwarové implementaci šifry pouhá permutace vodičů.
- **MIXCOLUMNS** – Ve funkci MixColumns probíhá sčítání, které je ale v tělese GF(2^8) ekvivalentní operaci XOR. Dále zde probíhá násobení dvěmi. V případě, že výsledek přeteče přes hodnotu 2^8, je nutné provést modulární redukci polynomem $x^8+x^4+x^3+x+1$, což je ekvivaltní s přixorováním hodnoty 0x1b. K přetečení dojde, pokud byl nejvyšší

bit jedničkový. Řešení jsem maximálně zjednodušil tak, že k prvnímu, druhému, čtvrtému a pátemu bitu (tedy bity, které mají hodnotu jedna v bytu 0x1b), přixoruji jeho nejvyšší bit – pokud měl hodnotu jedna, provede se operace XOR dle očekávání, pokud ne, nestane se vůbec nic. Poslední operace je násobení třemi, které lze jednoduše vyřešit násobením dvěma a přixorováním původní hodnoty dle vztahu 3a = 2a + a. Implemetace obsahuje for cyklus, který používá čtyři proměnné, z nichž každá symbolizuje jeden byte stavové matice šifry. Cyklus vždy do proměnných vybere čtyři byty, které tvoří jeden sloupec stavové matice, neboť všechny tyto sloupce se násobí stejnou hodnotou.

- **ROUND_OUTPUT_MPX** – Multiplexor, který prvních devět rund nastaví na svůj výstup signál z entity MIXCOLUMNS, pouze v poslední rundě použije výstupní signál entity SHIFTROWS, neboť MixColumns se v poslední rundě neprovádí.

- **ROUND_OUTPUT_REG** – Registr, do kterého se ukládá výstup každé „rundy“. Ve skutečnosti rundy šifry AES začínají funkcí SubBytes a končí Key Addition – má implemetace nestandardně každý takt začíná s funkcí Key Addition, přičemž na konci šifrování proběhně ještě poslední Key Addition a pak jsou data uložena do výstupního registru. Šifrování proto trvá dvanáct taktů – deset rund + nahrání vstupních dat + poslední přixorování klíče a nahrání výstupních dat do registru.

- **OUTPUT_REGISTER** – Registr, do kterého jsou nahrána zašifrovaná data, a který je následně drží na výstupu.

- **DEFAULT_KEY_REG** – Entita, která defaultně nahrává klíč, uložený v souboru CONSTANTS.vhd, a vystavuje ho na signál DEFAULT_KEY_SIG. Při platnosti signálu LOAD_NEW_KEY načte hodnotu ze signálu INPUT a uchová ji jako registr na signálu DEFAULT_KEY_SIG po dobu platnosti signálu USE_LOADED_KEY.

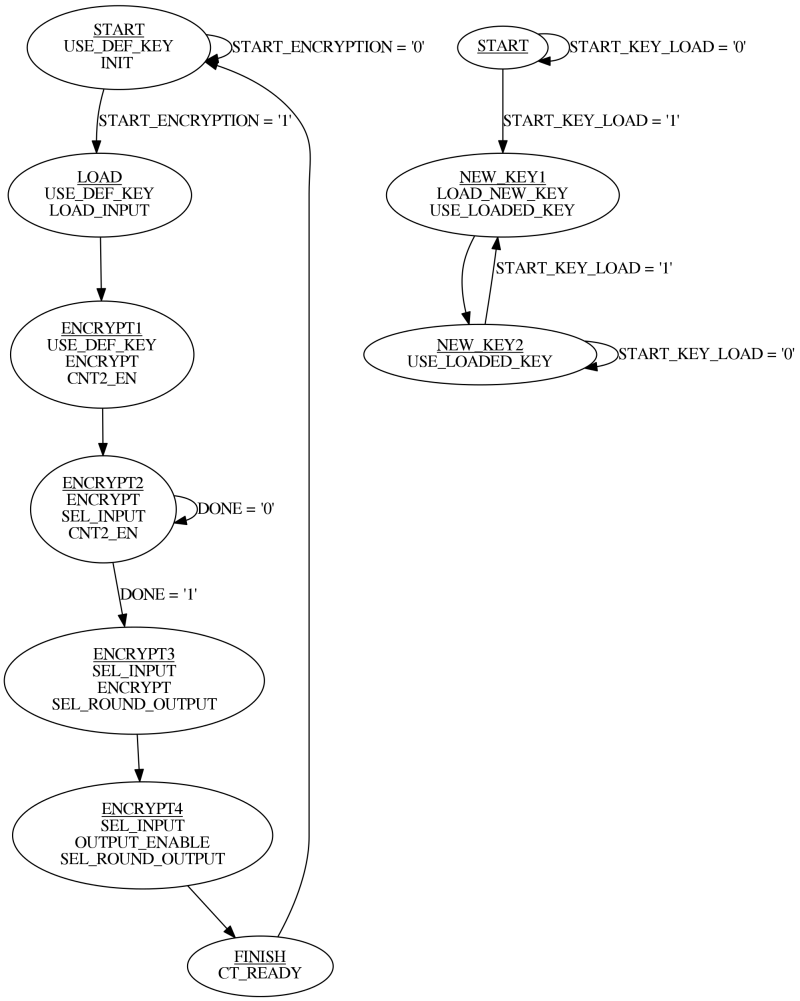
- **KEY_SCHEDULE** – Při platnosti signálu USE_DEF_KEY tato entita vystaví na signál NEW_KEY vstup DEFAULT_KEY_SIG. V opačném případě použije vstup KEY jako n-tý rundovní klíč, vypočte z něj (n+1)-ní rundovní klíč a vystaví jej na signál NEW_KEY. K výpočtu nového rundovního klíče se používá tabulka RCON uložená v CONSTANTS.vhd, a jako index se používá signál ROUND_COUNTER, který přichází z čítače v řadiči a značí číslo rundy, která právě probíhá.

- **KEY_REG** – Registr, který uchovává rundovní klíče, pokud je platný signál ENCRYPT.

2.3.1.3 Řadič

Řadič je zobrazen na obrázku 2.2. Řadič se skládá ze dvou nezávislých konečných automatů typu Moore. První slouží k samotnému šifrování a druhý k nahrání nového klíče. Je na implementaci obálky pro příjem dat, aby nedošlo k přechodu ze stavu START u obou automatů – ovšem taková situace by nikdy neměla nastat. Součástí řadiče je čítač, který počítá od nuly do devíti. Čítač se resetuje na hodnotu nula signálem INIT a počítá, pokud je platný signál CNT2_EN. Jeho hodnota se se spožděním 1 hodinového taktu zapisuje do signálu ROUND_COUNTER, který využívá proces KEY_SCHEDULE pro získání správné hodnoty z tabulky RCON. Před poslední rundou nastaví čítač signál DONE na jedna. Oba automaty jsou ve VHDL implementovány standardně jako tři procesy – stavový registr, kombinační obvod pro určení dalšího stavu a kombinační obvod pro určení výstupu.

- **Řadič šifrování**
  - **START** – Defaultní stav. Čeká se na signál START_ENCRYPTION.
  - **LOAD** – Zde dojde k náhrání vstupních dat na signálu INPUT do registru INPUT_REGISTER.
  - **ENCRYPT1** – Data projdou první „rundou“ a jsou uložena do registru ROUND_OUTPUT_REG.
  - **ENCRYPT2** – V tomto stavu proběhnou všechny rundy až do předposlední. Signál USE_DEF_KEY již není aktivní a tudíž entita KEY_SCHEDULE generuje další rundovní klíče. Do dalšího stavu se přechází po aktivaci signálu DONE.
  - **ENCRYPT3** – Zde proběhne poslední „runda“. Signál SEL_ROUND_OUTPUT změní výstup ROUND_OUTPUT_MPX tak, aby neproběhla operace MixColumns.
  - **ENCRYPT4** – K výstupním datům z ROUND_OUTPUT_REG je přixorován poslední rundovní klíč a tato data jsou následně uložena do registru OUTPUT_REGISTER.
  - **FINISH** – V posledním stavu je signálem CT_READY obálce, která zajišťuje příjem a odesílání dat, dáno najevo, že šifrování skončilo a šifrový text je připraven.
- **Řadič přijímání nového klíče**
  - **START** – Defaultní stav, ve kterém entita DEFAULT_KEY_REG nahrává defaultní klíč uložený v souboru CONSTANTS.vhd. Čeká se na signál START_KEY_LOAD.



Obrázek 2.2: AES01 – konečné automaty řadiče – vlevo řadič šifrování, vpravo přijímání nového klíče

- **NEW_KEY1** – Zde dojde k nahrání nového klíče ze signálu INPUT do registru v entitě DEFAULT_KEY_REG. Signál USE_LOADED_KEY, který zůstane již navždy aktivní, dává najevo, že se má použít tento klíč, a nikoli ten defaultní.
- **NEW_KEY2** – Tento stav je ekvivalentní se stavem START, jediným rozdílem je platnost signálu USE_LOADED_KEY.

2.3.2 AES02

Po neúspěšné snaze prolomit AES01 jsem jej upravil přesně do podoby, na kterou je útok pomocí Hammingových vah na poslední rundu myšlený. V této

verzi se nachází registr umístěný mezi operace Key Addition a Byte Substitution a ponechán byl výstupní registr. Implementace se na dvd nachází ve složce /src/AES_vhdl/AES02.

2.3.2.1 Rozhraní

Rozhraní AES02 je zcela totožné s rozhraním AES01, vizte 2.3.1.1.

2.3.2.2 Datová cesta

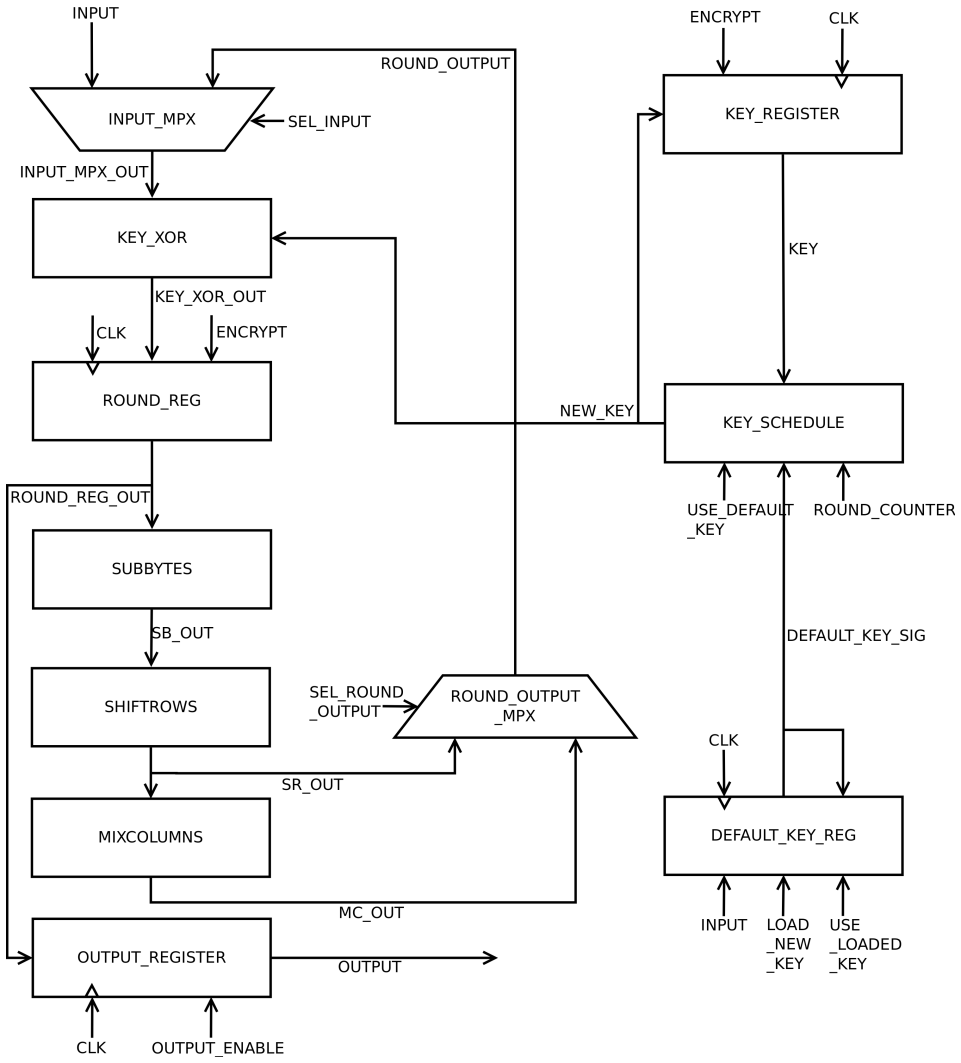
V datové cestě se již nenachází INPUT_REGISTER a ROUND_OUTPUT-_REG, jejich funkci převzal jediný registr ROUND_REG. Tento registr je umístěn tak, aby útok pomocí Hammingových vah na poslední rundu měl co největší šanci na úspěch, neboť dvě hodnoty, jejichž Hammingova váha se počítá, přesně odpovídají hodnotám, které se nacházely v rundovním registru na konci rundy číslo devět a deset.

Na začátku šifrování je k datům přixorován originální klíč a jsou uložena do rundovního registru. Poté proběhne všech deset rund a v posledním taktu jsou data opět uložena do registru OUTPUT_REGISTER. Šifrování proto celkem trvá dvanáct taktů. Datová cesta je zobrazena na obrázku 2.3.

2.3.2.3 Řadič

Řadič AES02 je velice podobný verzi AES01. Opět je tvořen dvěma konečnými automaty, jeden je určen pro samotné šifrování, druhý pro nahrání nového klíče. Součástí řadiče je čítač, který počítá od nuly do devíti. Čítač se resetuje na hodnotu nula signálem INIT a počítá, pokud je platný signál CNT2-_EN. Jeho hodnota se se spožděním 1 hodinového taktu zapisuje do signálu ROUND_COUNTER, který využívá proces KEY_SCHEDULE pro získání správné hodnoty z tabulky RCON. Před poslední rundou nastaví čítač signál DONE na jedna. Oba automaty jsou ve VHDL implementovány standardně jako tři procesy – stavový registr, kombinační obvod pro určení dalšího stavu a kombinační obvod pro určení výstupu. Řadič je vyobrazen na obrázku 2.4.

- **START** – Defaultní stav. Čeká se na signál START_ENCRYPTION.
- **LOAD** – Zde dojde k přixorování originálního klíče ke vstupnímu signálu INPUT a data jsou následně uložena do registru ROUND_REG.
- **ENCRYPT1** – V tomto stavu proběhnout všechny rundy až na poslední – zde runda skutečně odpovídá definici rundy dle definice AES, začíná operací Byte Substitution a končí Key Addition. Při platnosti signálu DONE se přejde do dalšího stavu.
- **ENCRYPT2** – Zde proběhne poslední „runda“. Signál SEL_ROUND-_OUTPUT změní výstup ROUND_OUTPUT_MPX tak, aby neproběhla operace MixColumns.

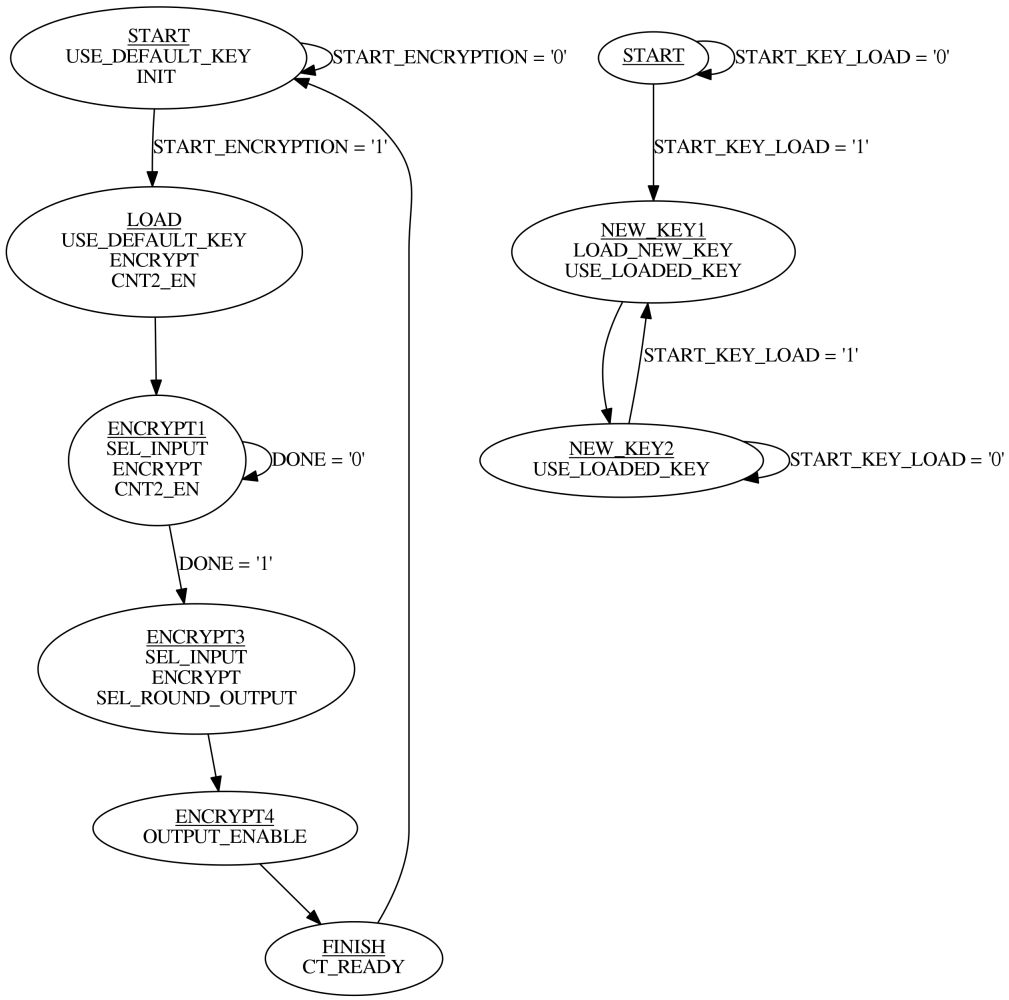


Obrázek 2.3: AES02 – datová cesta

- **ENCRYPT3** – Výstupní data z ROUND_REG jsou uložena do výstupního registru OUTPUT_REG.
- **FINISH** – V posledním stavu je signálem CT_READY obálce, která zajišťuje příjem a odesílání dat, dáno najevo, že šifrování skončilo a šifrový text je připraven.

2.4 Obálka pro komunikaci přes sériovou linku

Pro zajištění komunikace jednotky AES s počítačem jsem se rozhodl po konzultaci s vedoucím práce využít sériovou linku. Vytvořil jsem jednotku v jazyce



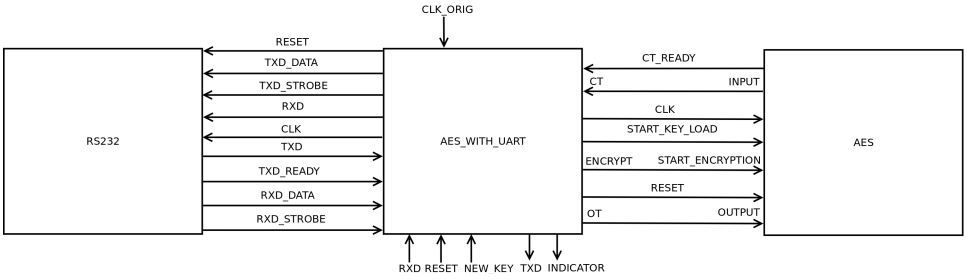
Obrázek 2.4: AES02 – konečné automaty řadiče – vlevo řadič šifrování, vpravo přijímání nového klíče

VHDL nazvanou AES_WITH_UART, která slouží jako prostředník mezi jednotkou AES a jednotkou zajišťující komunikaci přes sériovou linku. Modul pro komunikaci přes sérivou linku RS232.vhd lze nalézt na stránkách předmětu Bezpečnost a technické prostředky[2].

2.4.1 AES_WITH_UART01

První varianta obálky pro modul šifry a sériové linky byl použit u mých verzí šifry AES01 a AES02. Implementace se na dvd nachází ve složce /src/AES_WITH_UART_vhdl/01. Tato verze narozdíl od té druhé nemá žádný registr pro defaultní a nový klíč a předpokládá, že se tato funkcionalita nachází přímo

v šifře. Obálka též obsahuje předděličku hodin, která sníží frekvenci 32x – v mém případě z 50 MHz na 1,5625 MHz. Blokové schéma obálky, modulu pro komunikaci a šifry lze vidět na obrázku 2.5.



Obrázek 2.5: Sériová linka, prostředník AES_WITH_UART01 a šifra – blokové schéma

2.4.1.1 Rozhraní

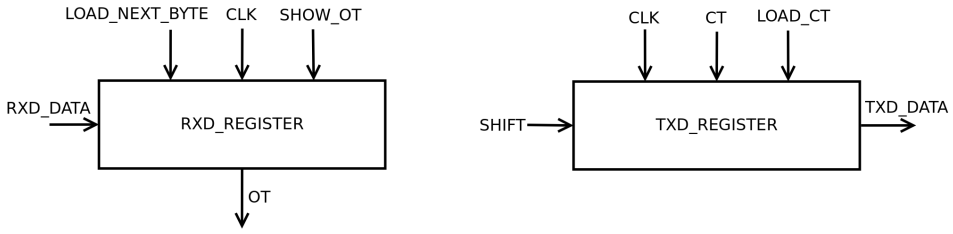
Kód 2.2: Rozhraní AES_WITH_UART01

```
entity AES_WITH_UART01
  port(
      RXD          : in  std_logic;
      TXD          : out std_logic;
      CLK_ORIG     : in  std_logic;
      RESET        : in  std_logic;
      NEW_KEY      : in  std_logic;
      INDICATOR    : out std_logic
       );
end AES_WITH_UART01;
```

RXD a TXD jsou vyvedeny na odpovídající piny sériové linky. Hodiny jsou nazvány CLK_ORIG, neboť v obálce projdou předděličkou a signál CLK je dále brán z nejvyššího bitu pětibitového čítače. Signál NEW_KEY dává najevo, že příští přijatá data nejsou otevřený text, ale nový klíč, a je napojen na jeden z přepínačů. Signál INDICATOR je po dobu jednoho hodinového taktu nastaven na hodnotu logické 1, a to přesně 32 taktů před začátkem šifrování. Tento signál se používá jako synchronizační impuls (trigger) pro osciloskop. Je vyveden na jeden z volných pinů.

2.4.1.2 Datová cesta

Datová cesta obálky se skládá ze dvou posuvných registrů. Jeden posuvný registr slouží pro příjem otevřeného textu (vstupu šifry) a druhý pro vysílání šifrového textu (výstupu šifry). Zobrazena je na obrázku 2.6.



Obrázek 2.6: Datová cesta obálky

- **RXD_REGISTER** – 16-bytový posuvný registr, který nahraje jeden byte na signálu RXD_DATA, pokud je platný signál LOAD_NEXT-_BYTE. Při platnosti signálu SHOW_OT zobrazí svůj obsah na signál OT.

- **TXD_REGISTER** – 16-bytový posuvný registr, který vystavuje nejvyšší byte na signál TXD_DATA. Při platnosti signálu LOAD_CT nahraje obsah signálu CT. Při platnosti signálu SHIFT dojde k posunutí o jeden byte směrem k vyšším bytům.

2.4.1.3 Řadič

Řadič je opět navržen jako automat typu Moore. Ve VHDL je navržen jako tři procesy – stavový registr, kombinační obvod pro určení dalšího stavu a kombinační obvod pro určení výstupu. Součástí řadiče jsou dva čítače – CNT1 a CNT2. CNT1 počítá od šestnácti do nuly a při nulové hodnotě aktivuje signál DONE – tento čítač se používá při příjmu a odesílání šestnácti bytů přes sériovou linku. Čítač CNT2 počítá od třiceti jedné do nuly a používá se v zdržovacím stavu DELAY2. Oba čítače se nastavují signálem INIT. Automat je zobrazen na obrázku 2.7.

- **START** – Defaultní stav. Čeká se signál RXD_STROBE, který dává najevo, že modul sériové linky RS232 přijal jeden byte.

- **LOAD1** – Zde se nahrává přijatý byte do registr RXD_REGISTER. Zároveň je platný signál CNT1_EN, dojde tak k jedné dekrementaci v registru CNT1.

- **LOAD2** – Pokud neplatí signál DONE, v tomto stavu se čeká na příjetí dalšího bytu indikovaného signálem RXD_STROBE. V případě, že je signál DONE jedna, pak pokud je i signál NEW_KEY jedničkový, je dalším stavem LOAD_KEY1, a bude se nahrávat nový klíč. V opačném případě je dalším stavem DELAY1 a bude se šifrovat.

- **DELAY1** – Zde je platný výstupní signál INDICATOR, který je na přípravku vyveden na jeden z pinů a je nezbytný pro nalezení místa ve kterém probíhalo šifrování na osciloskopu.

- **DELAY2** – Signál INDICATOR je nastaven na nulu a aby tím vzniklé rozdíly v odběru zařízení nemohly nijak ovlivnit měření samotného šifrování, čeká se po dobu 32 taktů v tomto zdržovacím stavu.

- **ENCRYPT1** – Registr RXD_REGISTER vystaví obsah na signál OT a obsah čítačů je zresetován.

- **ENCRYPT2** – Signál ENCRYPT dává modulu AES najevo, že může šifrovat. Čeká se na signál CT_READY.

- **ENCRYPT3** – Data na signálu CT jsou nahrána do registru TXD_REGISTER.

- **SEND1** – Pokud je signál DONE nulový, čeká se na možnost odeslat jeden byte přes sériovou linku, tedy na signál TXD_READY. Jinak je dalším stavem START.

- **SEND2** – Zde je modulu RS232 dáno najevo, že má odeslat byte. Zároveň dojde k jedné dekrementaci v čítači CNT1.

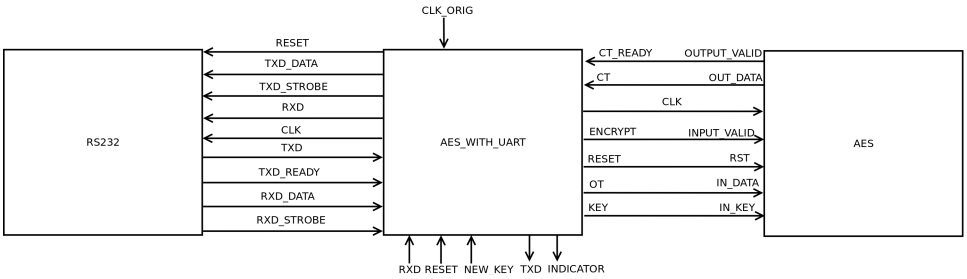
- **SEND3** – V tomto stavu dojde k posunu v posuvném registru TXD_REGISTER.

- **LOAD_KEY1** – Obsah registru RXD_REG, což je v tomto případě nový klíč, je vystaven na signál OT.

- **LOAD_KEY2** – Signálem START_KEY_LOAD je modulu AES dáno najevo, že má načíst nový klíč, který má na vstupu.

2.4.2 AES_WITH_UART02

Tato varianta obálky se od té první odlišuje v tom, že obsahuje entitu, která se stará o poskytnutí defaultního klíče modulu AES a případné nahrání nového klíče. Byla vytvořena proto, že žádná z verzí AES od Tomáše Zimmerhakla neobsahovala takový registr pro klíč, pouze defaultní hodnotu, a po konzultaci s vedoucím práce jsem měl tuto funkcionalitu umožnit, neboť v dřívějších pracích, nebo také v práci Jana Říhy, který pracuje na stejném tématu, ale na jiném přípravku[12], nebylo možné šifru prolomit, pokud obsahovala pouze konstantní klíč. Implementace se na dvd nachází ve složce /src/AES_WITH_UART_vhdl/02. Blokové schéma obálky, modulu pro komunikaci a šifry lze vidět na obrázku 2.8.



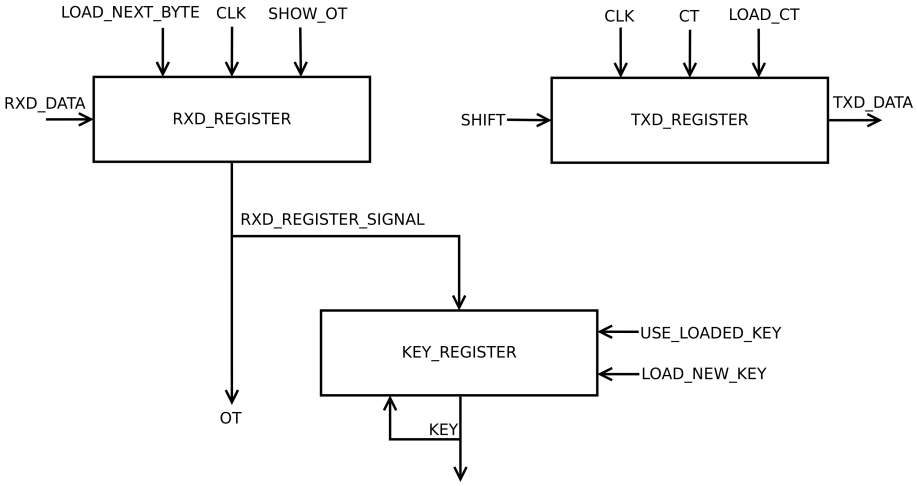
Obrázek 2.8: Sériová linka, prostředník AES_WITH_UART02 a šifra – blokové schéma

2.4.2.1 Rozhraní

Rozhraní je zcela totožné s AES_WITH_UART01, vizte 2.4.1.1.

2.4.2.2 Datová cesta

Datová cesta nově obsahuje entitu KEY_REGISTER, která defaultně nahrává konstantní klíč. Při platnosti signálu LOAD_NEW_KEY nahraje novou hodnotu klíče, kterou si při platnosti signálu USE_LOADED_KEY pamatuje. Datová cesta je zobrazena na obrázku 2.9.



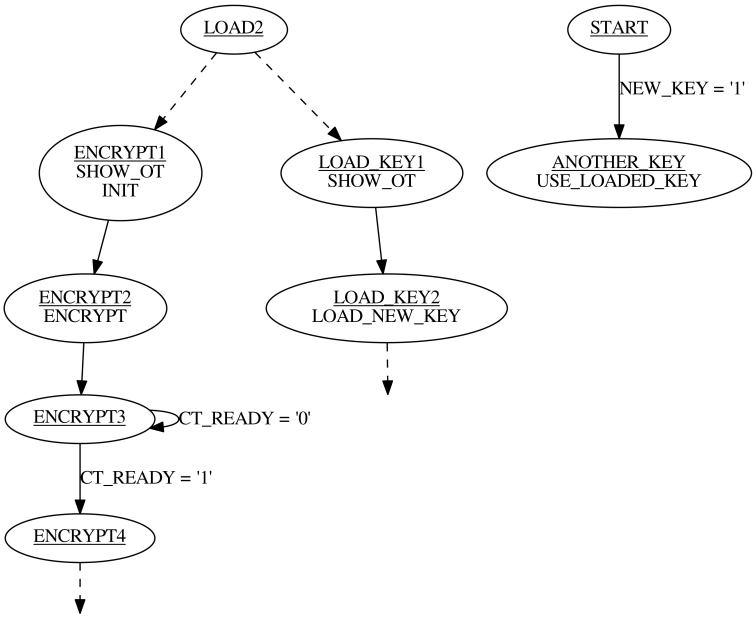
Obrázek 2.9: Datová cesta druhé verze obálky

2.4.2.3 Řadič

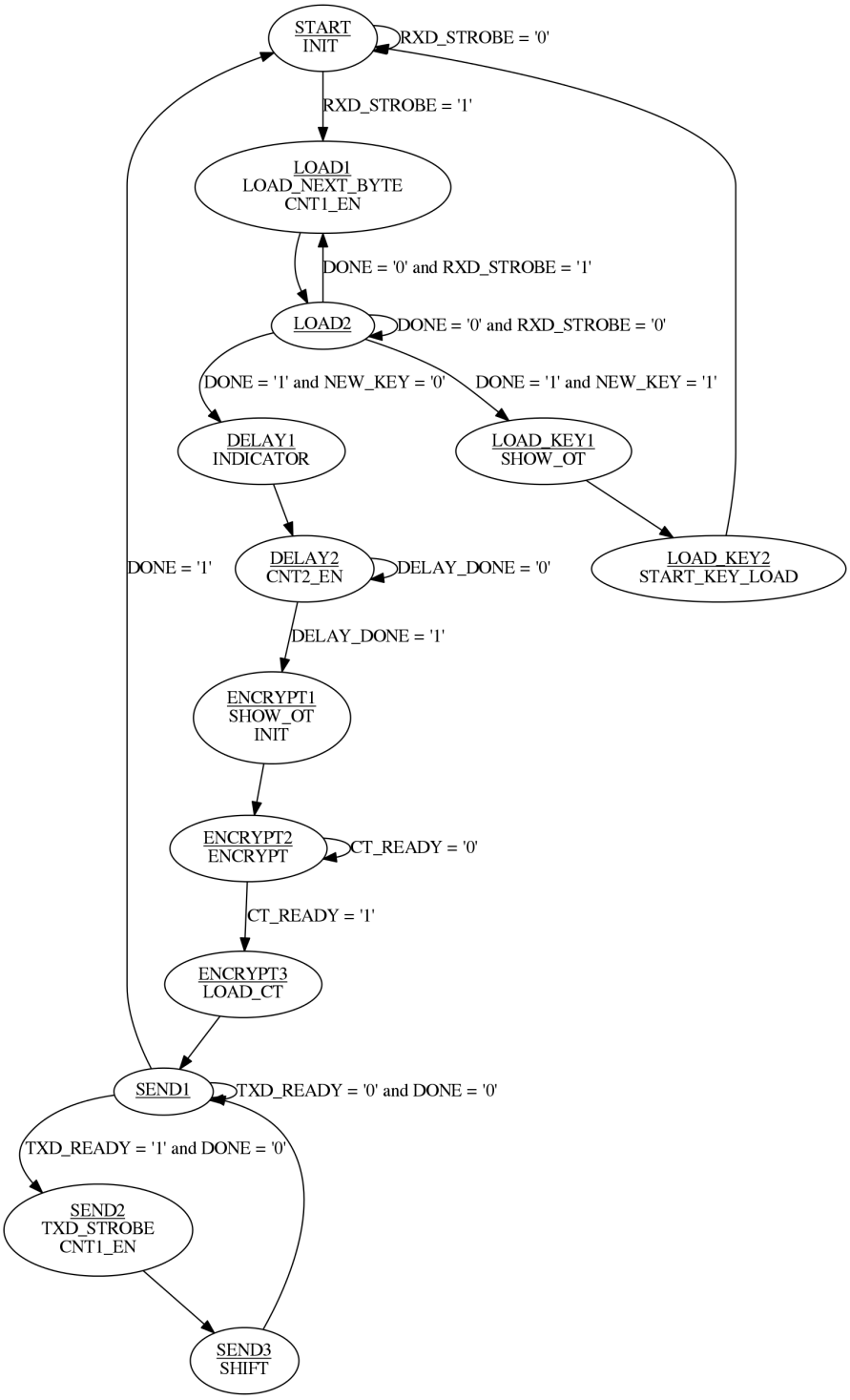
Řadič prošel oproti verzi na obrázku 2.7 několika úpravami, které jsou naznačeny na obrázku 2.10. Signál ENCRYPT je aktivní pouze v jediném taktu, jelikož to tak moduly Tomáše Zimmerhakla vyžadují. Dále se již nepoužívá signál LOAD_CT, který dával registru TXD_REGISTER najevo, že má nahrát

šifrový text – místo něho je na registr napojen přímo signál OUTPUT_VALID modulu šifry AES, který dává najevo, že šifrování skončilo a data jsou platná.

Největší úpravou je přítomnost druhého konečného automatu, který se stará o příjem nového klíče a jeho používání. Tento automat je velice jednoduchý – pokud přejde signál NEW_KEY do hodnoty jedna, přejde tento automat ze stavu START do stavu ANOTHER_KEY a setrvá tam do resetu. V tomto stavu je signál USE_LOADED_KEY jedničkový, a proto KEY_REGISTR nebude používat defaultní hodnotu. Proto je po přepnutí přepínače, který je zdrojem signálu NEW_KEY, nezbytné, aby byl do přípravku odeslán nový klíč, jinak bude otevřený text šifrován náhodnou hodnotou.



Obrázek 2.10: Řadič obálky AES_WITH_UART02 – vlevo naznačeny změny v řadiči pro šifrování oproti AES_WITH_UART01, vpravo část pro příjem nového klíče.



Obrázek 2.7: Řadič obálky AES_WITH_UART01

Kapitola 3

Testování

V této kapitole popíši jakým způsobem jsem testoval správnost skriptů implementujících metodu DPA, implementací šifry AES a obálky pro komunikaci přes sériovou linku.

3.1 Skript implementující DPA v programu Mathematica

Svůj skript na prolamování hesel pomocí DPA jsem otestoval na ukázkových datech dostupných na Wiki webu školy FIT ČVUT[13]. Při pozdějších úpravách skriptu, jako je útok na poslední rundu, či použití Hammingových vah, jsem použil naměřená data Lukáše Mazura[8], u kterých jsem znal klíč, a ověřil, že tento klíč útokem na jeho data získám.

3.2 AES pro čipovou kartu

Verzi AES v jazyce C jsem zakomponoval do operačního systému BHW-SOSSE a nahrál jej do čipové karty. Následně jsem do karty odeslal několik otevřených textů k zašifrování a přijal šifrové texty. Otevřené texty jsem zašifroval nezávisle na počítači, a šifrové texty poté manuálně porovnal. Nastíněný postup jsem několikrát zopakoval.

3.3 AES pro FPGA a obálka AES_WITH_UART

Testování hardwarových variant šifry AES a obálky pro šifru a modul sériové linky jsem rozdělil na verifikaci, při které jsem otestoval implementaci v jazyce VHDL v simulátoru ModelSim PE Student Edition 10.4. Dále jsem provedl validaci, tedy kontrolu již nahrané implementace na přípravku Spartan-3E.

3.3.1 Verifikace

Pro otestování návrhu ještě před jeho nahráním do přípravku jsem vytvořil jednoduchý testbench v jazyce VHDL nazvaný TEST.vhd, který do modulu AES pošle dva různé otevřené texty a pokaždé ověří správnost vrácených šifrových textů. Poté nahraje do jednotky AES nový klíč a provede ještě jeden test správnosti šifrování.

Po verifikaci funkčnosti samotné šifry jsem dále testoval funkčnost obálky sloužící ke komunikaci přes sériovou linku. Napsal jsem testbench AES_UART_TB, který pošle na signál RXD otevřený text a následně přečte příchozí šifrový text na TXD a ověří jeho správnost.

3.3.2 Validace

Validaci jsem provedl na desce Spartan-3E Starter Kit Board[11]. Otestování finální hardwarové implementace mých šifer jsem provedl jednak opět porovnáním několika otevřených a šifrových textů získaných přes sériovou linku z přípravku s jinými, nezávisle zašifrovanými na počítači. Pro testovací komunikace s FPGA jsem používal Advanced Serial Port Terminal[14]. Dále jsem pro validaci implementace napsal jednoduchý program v jazyce C, který ověří správnost šifrování ze souborů plaintext.txt a ciphertext.txt, což jsou výstupy programu SC Power Measurements[15]. Program vždy načte otevřený text ze souboru plaintext.txt, zašifruje jej pomocí implementace AES, kterou jsem použil i na čipové kartě, a výsledný šifrový text porovná s odpovídajícím v ciphertext.txt.

Kapitola 4

Rozdílová odběrová analýza

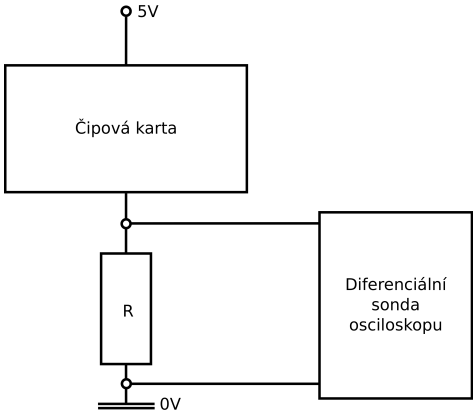
Tato kapitola je jádro celé mé bakalářské práce. Během své práce jsem provedl mnoho různých útoků pomocí rozdílové odběrové analýzy. Nejprve jsem útočil pomocí DPA na čipovou kartu. Později jsem zaútočil na své implementace šifry AES na FPGA. V případě AES01 jsem zprvu nebyl vůbec úspěšný, a proto vzniknul AES02, u kterého jsem předpokládal větší úspěch, což se ovšem nakonec úkazalo jako nesprávná domněnka. Po úspěšném prolomení AES01 jsem se přesunul k implementacím šifry AES Tomáše Zimmerhakla, které se mi prolomit dařilo, a proto jsem následně provedl jejich porovnání na základě minimální počtu vzorků, které ještě stačí k úspěšnému prolomení konkrétní implementace.

Během célé své práce jsem používal osciloskop Agilent Technologies InfiniiVision 7000A[16].

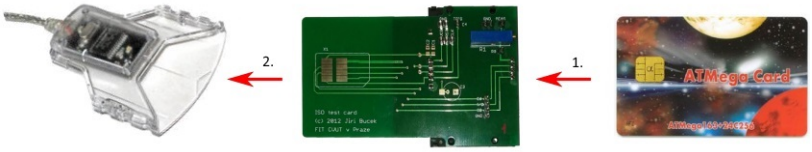
4.1 Aplikace DPA na čipovou kartu

Jako první jsem pro seznámení se s metodou DPA a také s prací s osciloskopem útočil na čipovou kartu AVR Smart Card. Nejprve jsem zaútočil na ukázková naměřená data (výstup programu SC Power Measurements[15]) na čipové kartě. Útok jsem provedl pomocí Hammingových vah na první rundu a tuto implementaci jsem úspěšně prolomil. Následně jsem ještě provedl stejný útok na ukázková naměřená data s neznámým klíčem a i tuto implementaci jsem úspěšně prolomil (správnost klíče stačí otestovat na přiložených otevřených a šifrových textech).

Pokračoval jsem útoky na vlastní implementaci šifry AES v jazyce C, kterou jsem vložil do operačního systému BHW-SOSSE a nahrál do čipové karty. Změřil jsem její spotřebu během 200 šifrování, což by mělo dle zkušeností vedoucího práce z předmětu MI-BHW[2] stačit k úspěšnému prolomení. Zapojení měřící diferenciální sondy osciloskopu je na obrázku 4.1. K měření spotřeby karty během šifrování jsem použil adaptér vytvořený pro předmět MI-BHW[2], který je na obrázku 4.2. Provedl jsem útok pomocí Hammingových vah na



Obrázek 4.1: Zapojení měřící proby osciloskopu při měření spotřeby čipové karty během šifrování.

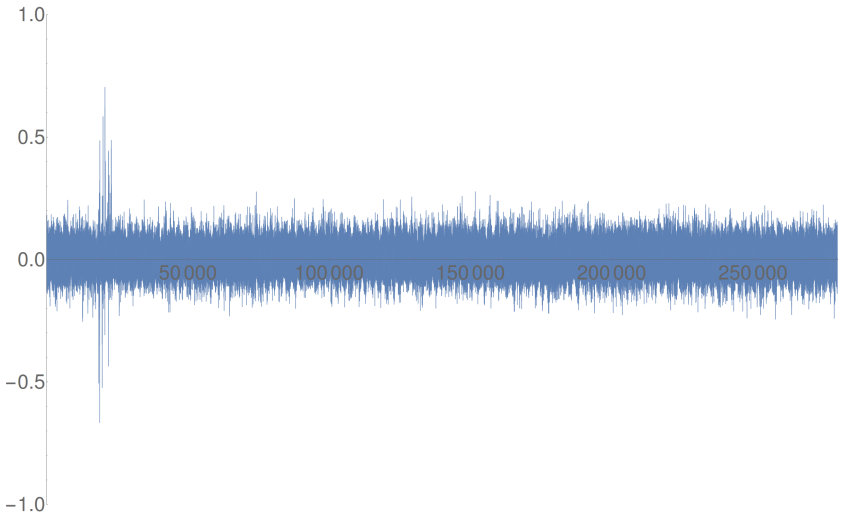


Obrázek 4.2: Zleva USB čtečka karet, adaptér pro snadné měření spotřeby, AVR Smart Card. Převzato z [2].

první a poslední rundu. Následně jsem ještě provedl útok pomocí Hammingových vzdáleností na poslední rundu, jelikož tento typ útoku budu používat při útoku na implementace šifry na FPGA. Na základě zjištění Lukáše Mazura[8] vím, že útok Hammingovými vzdálenostmi na první rundu nebudu používat, proto jsem jej netestoval ani zde. Pro komunikaci s čipovou kartou jsem použil software Java Smart Card Explorer[17].

4.1.1 DPA na čipovou kartu – Hammingovy váhy na první rundu

Při tomto útoku vypočítám hodnotu bytu v místě, které lze vidět na obrázku 1.5 vpravo. Pro vypočítanou hodnotu bytu následně určím Hammingovu váhu. Útok byl úspěšný, výsledný průběh korelačních koeficientů v čase pro šestnáctý byte klíče lze vidět na obrázku 4.3.



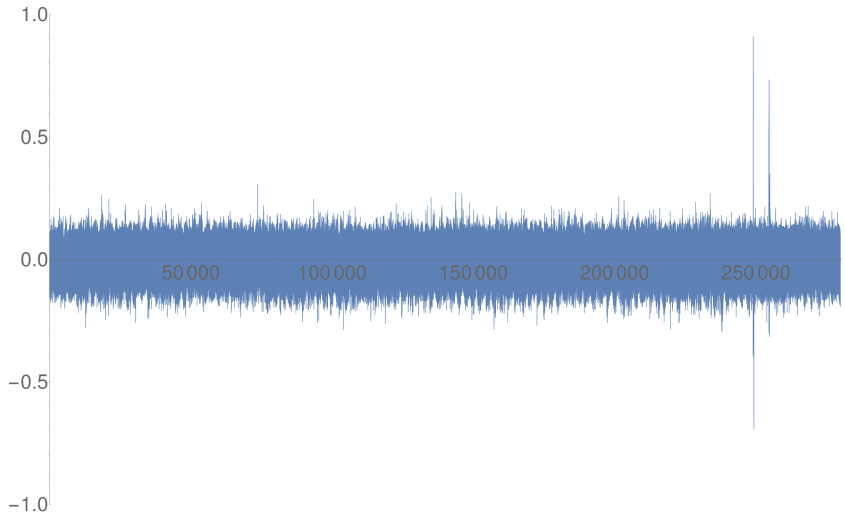
Obrázek 4.3: Průběh korelačních koeficientů v čase pro šestnáctý byte klíče při útoku na první rundu pomocí Hammingových vah. Zde je korelace vypočtená nad celou délkou naměřeného průběhu spotřeby, proto místo, ve kterém se pracovalo s hodnotou, na kterou útočím, je správně na začátku celého grafu.

4.1.2 DPA na čipovou kartu – Hammingovy váhy na poslední rundu

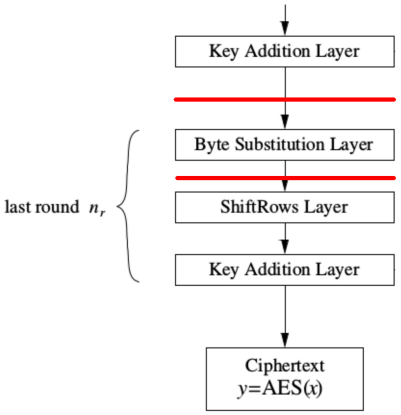
Při tomto útoku pracuji s hodnotou bytu před poslední operací Byte Substitution, jejíž umístění lze vidět na obrázku 1.5 červeně vlevo nahoře. Útok prolamuje desátý rundovní klíč, proto je nezbytné jej pak převést na originální klíč. Útok byl opět úspěšný a výsledný průběh korelačních koeficientů v čase pro prolomený šestnáctý byte desátého rundovního klíče je na obrázku 4.4.

4.1.3 DPA na čipovou kartu – Hammingovy vzdálenosti na poslední rundu

Poslední útok, který jsem provedl na čipovou kartu, byl pomocí Hammingových vzdáleností. Po konzultaci s vedoucím jsem použil vzdálenost dvou hodnot, které se nacházejí před a za poslední operací Byte Substitution. Lze je vidět na obrázku 4.5. Podařilo se mi prolomit všechny byty klíče až na jediný – čtvrtý. Důvod jsem neodhalil. Graf korelací pro šestnáctý byte klíče, který jsem prolomil, je na obrázku 4.6. Ostatní prolomené byty klíče mají grafy korelací velice podobné. Neprolomený čtvrtý byte klíče má průběhy korelačních koeficientů v čase vyobrazeny na obrázku 4.7.



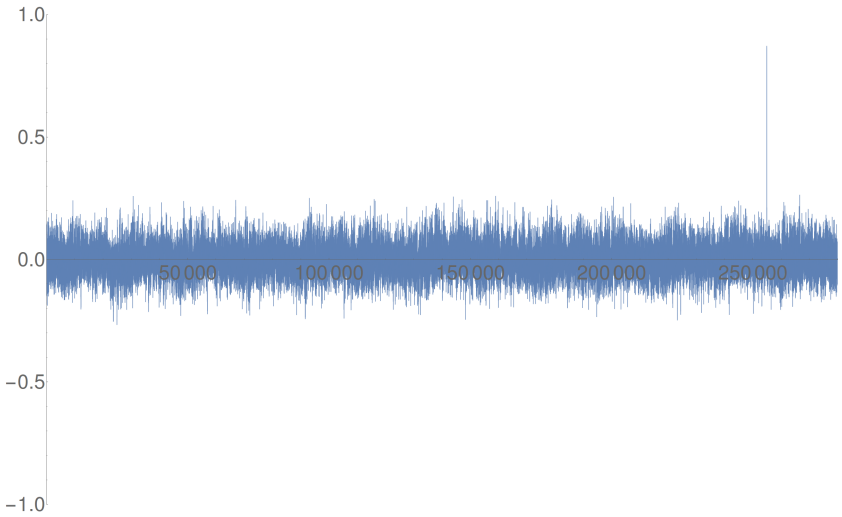
Obrázek 4.4: Průběh korelačních koeficientů v čase pro šestnáctý byte desátého rundovního klíče při útoku na poslední rundu pomocí Hammingových vah. Zde je korelace vypočtená nad celou délkou naměřeného průběhu spotřeby, proto místo, ve kterém se pracovalo s hodnotou, na kterou útočím, je správně na konci celého grafu.



Obrázek 4.5: Hodnoty, jejichž Hammingovu vzdálenost počítám při útoku na poslední rundu na čipovou kartu. Převzato z [1] a upraveno.

4.2 Aplikace DPA na FPGA

V následující sekci popíši útoky na hardwarovou implementaci šifry AES. Nejprve jsem útočil na svůj AES01. Kvůli neúspěchu jsem vytvořil a zaútočil na AES02, který se mi ovšem nepovedlo kompletně prolomit. Po hlubším zamyšlení nad vnitřní implementací AES01 jsem jej nakonec dokázal prolomit.

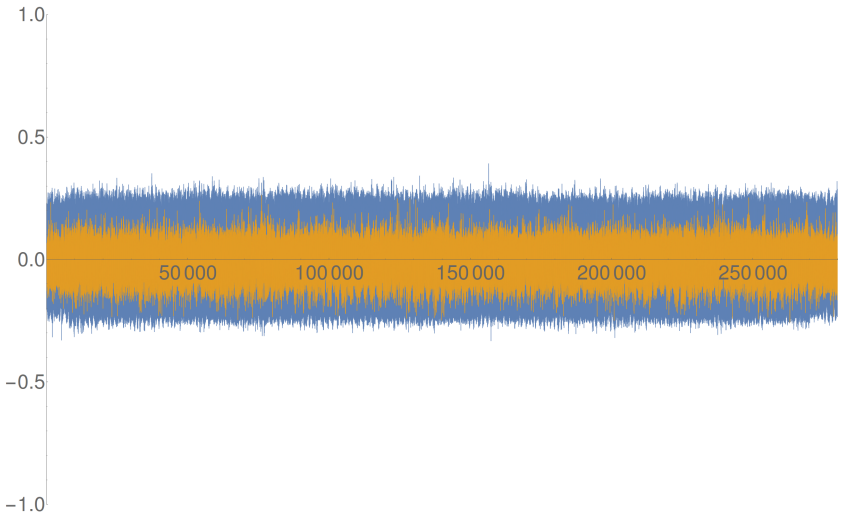


Obrázek 4.6: Průběh korelačních koeficientů v čase pro šestnáctý byte klíče při útoku na poslední rundu pomocí Hammingových vzdáleností. Zde je korelace vypočtená nad celou délkou naměřeného průběhu spotřeby, proto místo, ve kterém se pracovalo s hodnotou, na kterou útočím, je správně na konci celého grafu.

Následně jsem zaútočil na varianty AES Tomáše Zimmerhakla. Jelikož jsem byl úspěšný, porovnal jsem odolnost verzí zabezpečených proti poruše oproti základní verzi.

Po konzultaci s vedoucím práce jsem se rozhodl používat přípravek Spartan-3E Starter Kit Board[11], neboť pro něj po mých předchůdcích již byl připraven obvod s rezistorem, na němž se měří napětí, vizte obrázek 4.9. Při útocích pomocí DPA na FPGA jsem využil informací z bakalářských prací Jana Severyna[6] a Lukáše Mazura[8]. Od začátku jsem používal předděličku hodin 1:32, čímž jsem snížil frekvenci FPGA z 50 MHz na 1,5625 MHz. Dále jsem použil desku Spartan-3E s odstraněnými blokujícími kondenzátory C158 - C175 – za jejich odstranění děkuji vedoucímu práce. Použil jsem AC předzesilovač PA 303 BNC se zesílením 30dB[10]. Zapnul jsem na osciloskopu bandwidth limit – omezení šířky pásma. Útočil jsem výhradně za použití Hammingových vah na poslední rundu, jelikož to byl jediný úspěšný typ útoku mých předchůdců.

Kvůli absenci regulátoru napětí a olověných akumulátorů jsem alespoň zpočátku používal spínaný síťový zdroj, který není tak vhodný a jeho použití způsobuje v naměřených datech rušení o frekvenci 16 kHz. Později jsem koupil olověné akumulátory a pan Jiří Buček upravil regulátor napětí tak, aby bylo možné ho použít s jediným akumulátorem o napětí 6V, za což mu děkuji. Regulátor napětí vyrábí ze vstupu, kterým je napětí 6V z olověného akumulátoru, tři různá napětí - 1,2V, 2,5V a 3,3V. Napájení 1,2V je připo-



Obrázek 4.7: Průběh korelačních koeficientů v čase pro čtvrtý byte klíče při útoku na poslední rundu pomocí Hammingových vzdáleností. Modře je vyobrazena korelace pro špatně určený byte s hodnotou 0x07. Žlutě je zobrazena korelace pro správnou hodnotu 0x46.

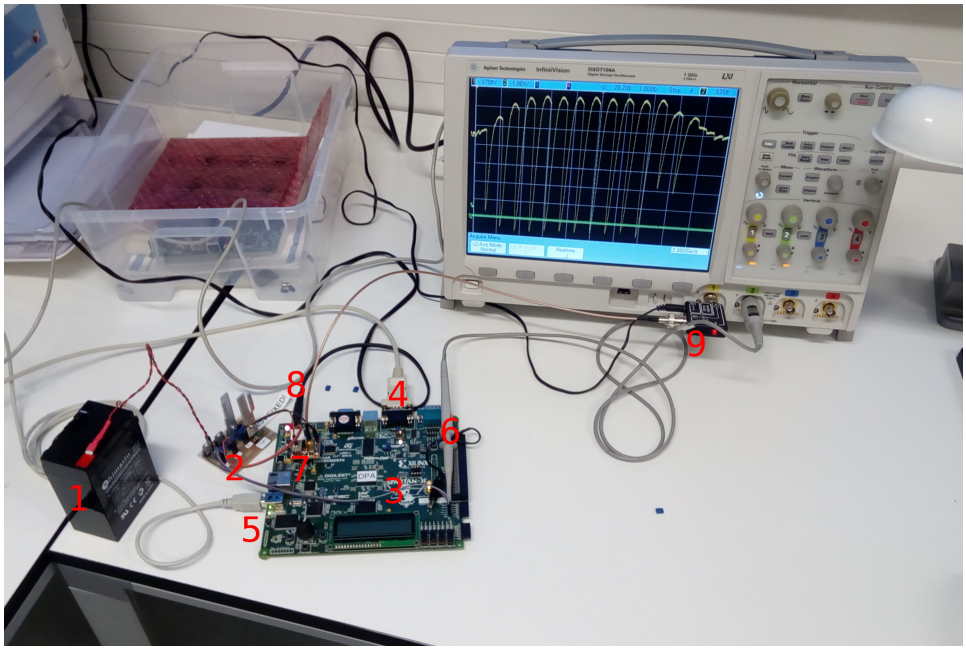
jeno na konektor JP7, napájení 2,5V na konektor JP6 – obě s pomocí obvodu s rezistorem o odporu 1Ω, který je vidět na obrázku 4.9. Napájení 3,3V lze připojit na kterýkoliv periferní Vcc a GND pin na desce. Způsob zapojení předzesilovače na FPGA je na obrázku 4.10. Narozdíl od čipové karty zde měřím úbytek napětí na FPGA dle vzorce

$$u(FPGA) = 1,2V - R \times i(t), \tag{4.1}$$

tedy v momentech většího odběru FPGA naměřím na osciloskopu propady napětí. K automatizaci získávání průběhů spotřeby z osciloskopu a komunikace s FPGA používám software SC Power Measurements[15]. Jeden průběh spotřeby se skládá z 20000 vzorků. Nastavení osciloskopu při všech měřeních je v tabulce v příloze C, vizte C.1.

4.2.1 DPA na FPGA – AES01

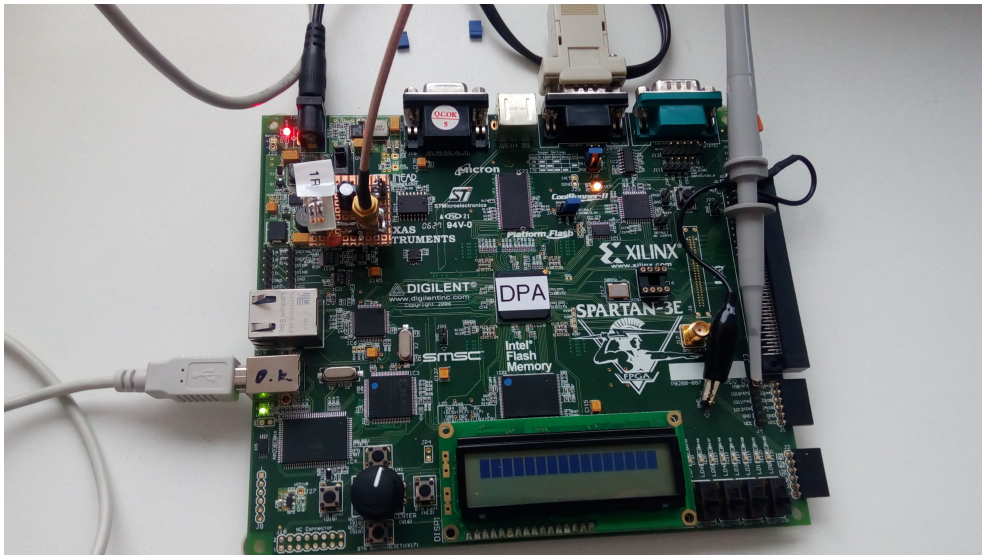
První varianta hardwarové implementace šifry, na kterou jsem útočil, byl můj AES01. Nedařilo se mi jej kompletně prolomit, kvůli čemuž vzniknul AES02. Později jsem odhalil příčinu neúspěchu a nakonec tuto verzi prolomit dokázal. V následujících sekcích popíši útok na data naměřená při použití spínaného zdroje a poté se krátce zmíním o útoku na data naměřená při použití olověného akumulátoru.



Obrázek 4.8: Probíhající měření. Vysvětlivky: 1: olověný akumulátor, 2: regulátor napětí, 3: Spartan 3E, 4: konektor sériové linky z PC zapojený do FPGA, 5: konektor USB z PC k programování desky, 6: sonda druhého kanálu osciloskopu měřící trigger, 7: místo měření úbytku napětí na rezistoru, 8: místo připojení konektoru síťového zdroje k desce (v tuto chvíli se nepoužívá, na desce je vypínač), 9: předzesilovač 30dB.

4.2.1.1 Deska napájená spínaným zdrojem

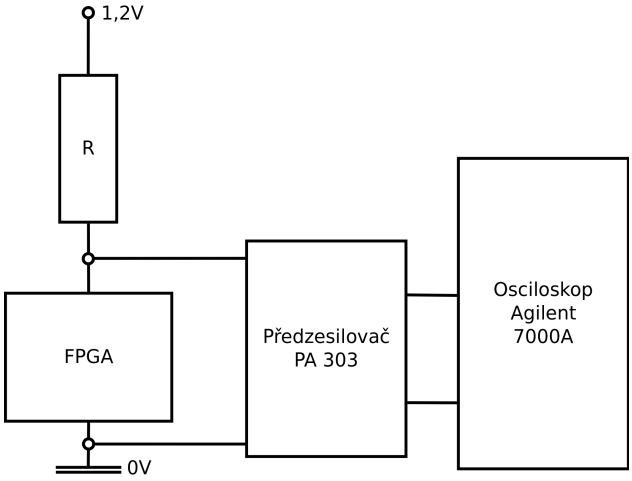
Nejprve jsem naměřil 100000 vzorků spotřeby za použití síťového zdroje. Průběh prvních 10 vzorků spotřeby lze vidět na obrázku 4.11. Poslední runda, na kterou budu útočit, by se dle znalosti návrhu této implementace šifry měla odehrát někde mezi 14000. - 15000. vzorkem. Útok na kompletní data zde není možný z důvodu vyčerpání paměti PC (8GB). Útok úspěšně prolomil 3., 6., 7., 8., 9., 11., 12., 13., 14., 15. a 16. byte 10. rundovního klíče. Graf korelací pro všech 256 hypotéz 3. bytu 10. rundovního klíče je na obrázku 4.12. Graf korelací neprolomeného 1. bytu klíče je na obrázku 4.13. Při snaze zjistit, proč je útok na AES01 neúspěšný (nelze nalézt všechny byty klíče) jsem si uvědomil, že samotný návrh hardwaru této implementace šifry je nevhodný pro útok, jak ho provádím. Během útoku Hammingovými vzdálenostmi na poslední rundu předpokládám existenci registru tak, jak je umístěn u AES02, vizte obrázek 2.3. Vypočítám proto hodnotu, která se v registru někdy nacházela, budu jí značit X, a hodnotu, která jí v nějakém okamžiku nahradila, budu jí značit Y, přičemž jejich Hammingova vzdálenost by měla dobře odhadnout spotřebu během této změny dat na stupnici 0 - 8. Hodnotu X vypočtu tak, že



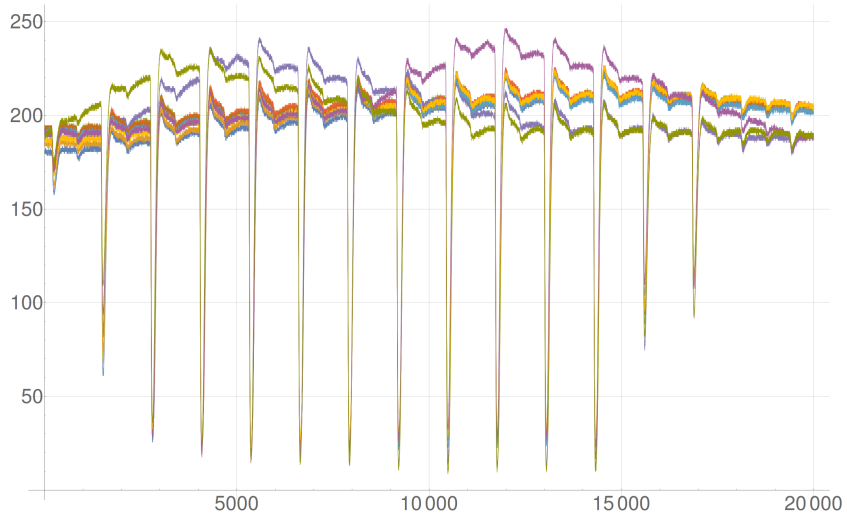
Obrázek 4.9: Detail desky Spartan-3E Starter Kit Board s nasazeným obvodem s rezistorem sloužícím k měření na konektorech JP6 a JP7 (vlevo nahoře). Vpravo diferenciální sonda osciloskopu měřící synchronizační impuls (trigger) na pinu D7.

vezmu některý byte šifrového textu, provedu nad ním inverzní Key Addition pro všechny možnosti hodnot bytu klíče a inverzní Byte Substitution. Hodnota X se ale nachází na jiném místě, než původní byte šifrového textu, který jsem použil, a to kvůli operaci ShiftRows. Vypočtu proto umístění hodnoty X, a jako Y použiji odpovídající byte šifrového textu. Popsaný způsob ale vůbec neodpovídá implementaci AES01, vizte obrázek 4.14. Abych zde získal dvě hodnoty, které se po sobě nacházely v rundovním registru (X a Y), musel bych pro i-tý byte provést inverzní Key Addition, inverzní ShiftRows, inverzní Byte Substitution a následně další inverzní Key Addition, ovšem kvůli operace ShiftRows na jiném místě – musel bych hádat další, tentokrát k-tý byte klíče, a to navíc 9. místo 10. rundovního. Tímto způsobem bych získal hodnotu X. Hodnotu Y bych pak získal provedením inverzního Key Addition nad k-tým bytem šifrového textu, ovšem to vede k hádání dalšího bytu 10. rundovního klíče – kromě bytů 1, 5, 9 a 13, které operace ShiftRows nepřesouvá. Pokud by původní útok pomocí DPA neodhalil žádný byte klíče, bylo by vytvoření modelu spotřeby pomocí přesných hodnot X a Y nemožné už jen pro obrovskou paměťovou náročnost kvůli všem možným hodnotám, kterých mohlo vícero bytů nabývat, a jejich kombinacím.

V případě AES01 mi zbývalo pět neprolomených bytů 10. rundovního klíče. Zjistil jsem, že pro zlepšení výsledných korelací pomáhá ke správnému bytu šifrového textu přixorovat odpovídající byte 10. rundovního klíče. Tím získám

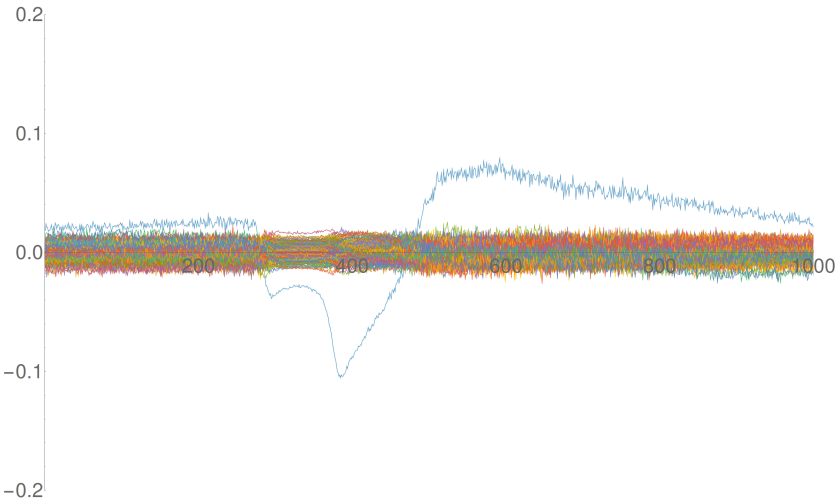


Obrázek 4.10: Schéma zapojení předzesilovače na FPGA.



Obrázek 4.11: Prvních 10 vzorků spotřeby pro AES01 s deskou napájenou síťovým zdrojem. Vzorky na sebe nelícují kvůli rušení, které do obvodu vysílá síťový zdroj.

hodnotu, která se v této implementaci šifry AES skutečně nacházela v rundovním registru po desáté rundě – hodnotu Y. Hodnota X je pořád špatně – bylo by potřeba k ní přixorovat odpovídající byte 9. rundovního klíče, ovšem ten neznám a jeho hádaní by vedlo k již zmíněným problémům. I pouhé zpřesnění jedné ze dvou hodnot mi ovšem umožnilo čtyři ze zbývajících pěti bytů 10. rundovního klíče odhalit. Například u prvního bytu vždy přixoruji k prvnímu bytu šifrového textu hodnotu 1. bytu 10. rundovního klíče, kterou zrovna hádám, neboť operace ShiftRows první byte nikam nepřesouvá – výsledné ko-



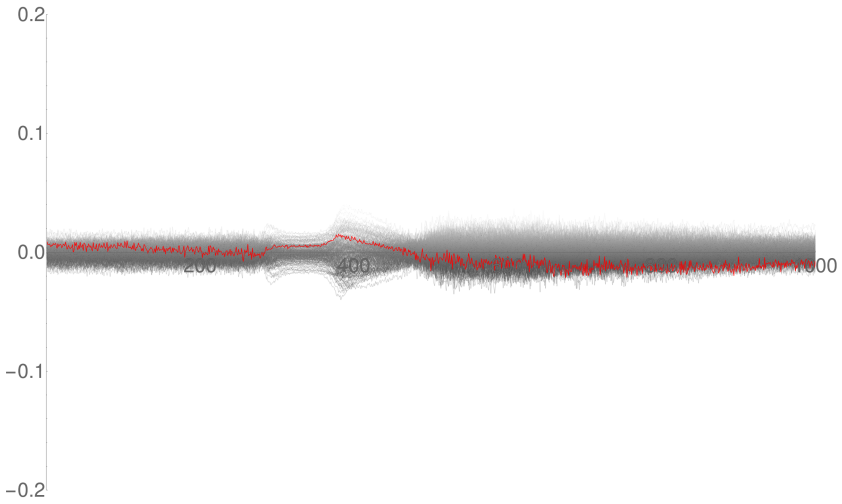
Obrázek 4.12: Průběh korelačních koeficientů v čase pro všech 256 hypotéz 3. bytu 10. rundovního klíče AES01. Správná hodnota (0x24) je zde dobře patrná. Ostatní byty, které jsem úspěšné prolomil, měly velice podobné grafy korelací.

relace jsou na obrázku 4.15. Dále například u 2. bytu dojde při inverzním ShiftRows k přesunu na šestou pozici. 6. byte klíče jsem úspěšně prolomil, proto jej přixoruji k 6. bytu otevřeného textu a tím zase získám přesnou hodnotu Y k výpočtu Hammingovy vzdálenosti.

Jediný byte, který se mi tímto postupem nepodařilo prolomit, byl byte 10.. Proto jsem se u něj rozhodl i k zpřesnění hodnoty X. K tomu jsem potřeboval zjistit 10. byte 9. rundovního klíče. 10. byte 10. rundovního klíče se snažím uhodnout, ovšem hodnotu, kterou zrovna hádám, mohu použít jakožto známou. Ze znalosti Key schedule šifry AES (vizte obrázek 1.2) jsem odvodil, že je třeba k 10. bytu 10. rundovního klíče přixorovat 6. byte 10. rundovního klíče, který znám. Tím jsem získal 10. byte 9. rundovního klíče. Ten jsem poté přixoroval k nepřesné hodnotě X, tak jak jsem jí doteď používal, a získal přesnou hodnotu X, tedy hodnotu, která se nacházela v této implementaci po 9. rundě v rundovním registru. Model spotřeby vytvořený pomocí Hammingových vzdáleností X a Y byl již dostatečně přesný a i 10. byte se mi podařilo prolomit. Tím jsem prolomil celý tajný klíč.

4.2.1.2 Deska napájená olověným akumulátorem

Po úpravě regulátoru napětí a nákupu olověných akumulátorů jsem začal používat k napájení desky tuto kombinaci, neboť Lukáš Mazur[8] ve své práci zjistil, že použití tohoto zdroje napětí snižuje počet průběhů nutný k prolomení. Graf prvních 10 průběhů spotřeby při napájení desky olověným akumulátorem lze vidět na obrázku 4.16. Porovnal jsem minimální nutný počet průběhů



Obrázek 4.13: Průběh korelačních koeficientů v čase pro všech 256 hypotéz 1. bytu 10. rundovního klíče AES01. Správná hodnota (0x36 – červeně) není nijak výrazná. Ostatní neprolomené byty měly podobné grafy korelací.

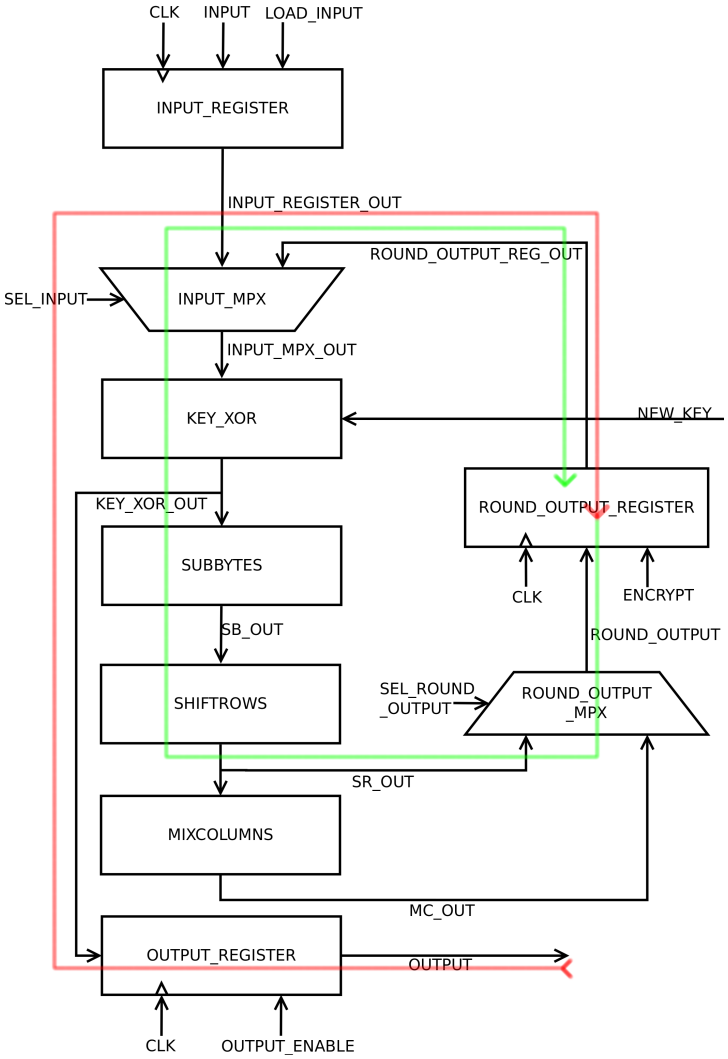
k úspěšnému prolomení několika bytů a zjistil, že verze napájená spínaným zdrojem vyžaduje méně průběhů, což je v rozporu s tím, co ve své práci zjistil Lukáš Mazur. Je ale možné, že došlo k nějaké chybě měření, pro relevantnější závěr by bylo nutné obě varianty změřit a prolomit vícekrát. Dále jsem ve své práci používal jakožto zdroj napětí výhradně olověný akumulátor.

	spínaný zdroj	baterie
3. byte	1200	1500
9. byte	1800	2250
13. byte	1450	1500
16. byte	14500	22250

Tabulka 4.1: Přibližný minimální počet průběhů nutný k úspěšnému prolomení bytu klíče – útoky na verzi napájenou spínaným zdrojem jsou překvapivě úspěšnější.

4.2.2 DPA na FPGA – AES02

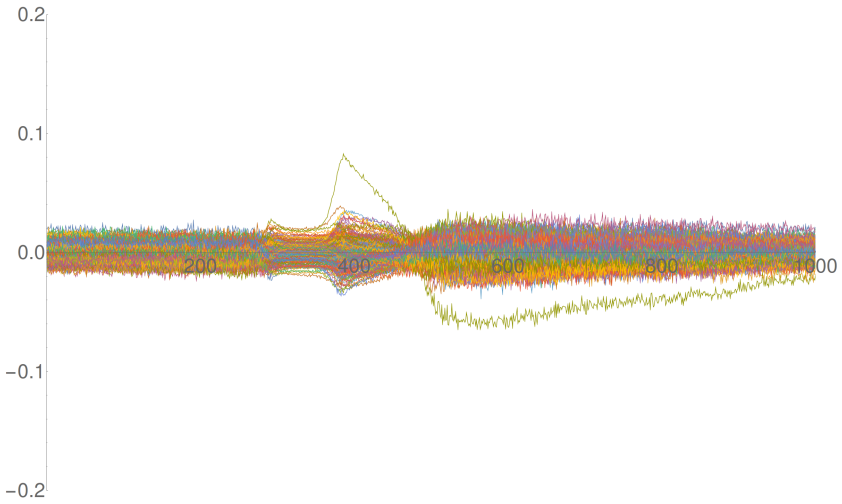
Po problémech s prolamováním AES01 jsem obvod upravil přesně do podoby, na kterou je útok pomocí Hammingových vzdáleností na poslední rundu myšlený. Rundovní registr se v ní nachází mezi operacemi Key Addition a Byte Substitution, vizte obrázek 2.3, a tudíž dvě hodnoty, mezi kterými se počítá Hammingova vzdálenost, přesně odpovídají hodnotám, které se v této imple-



Obrázek 4.14: Blokové schéma AES01 – červená cesta naznačuje úpravy dat pro získání hodnoty, která se v rundovním registru nacházela po 10. rundě, zelená další úpravy pro získání předcházející hodnoty, tedy té, která se v rundovním registru nacházela po 9. rundě.

mentaci nacházely v rundovním registru po 9. a následně po 10. rundě za sebou.

Naměřil jsem 100000 průběhů spotřeby. Graf prvních 10 průběhů spotřeby lze vidět na obrázku 4.17. Ze znalosti implementace této šifry jsem odhadl, že poslední runda probíhá někde mezi 15500. a 15900. vzorkem. Útok na kompletní data zde opět není možný z důvodu vyčerpání operační paměti. Zaútočil jsem s použitím všech 100000 vzorků spotřeby a úspěšně prolomil 2., 3., 4., 6., 7., 8., 9., 10., 11., 12., 13., 14., 15. a 16. byte 10. rundovního klíče. Graf kore-



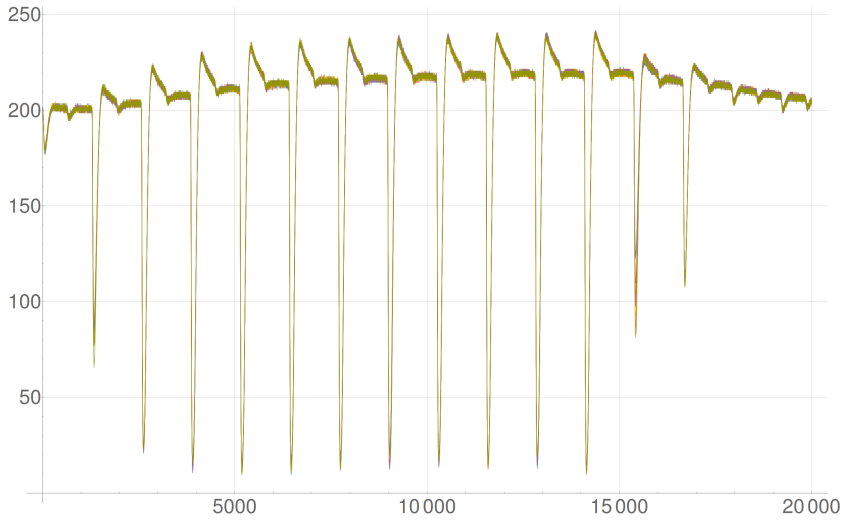
Obrázek 4.15: Průběh korelačních koeficientů v čase pro všech 256 hypotéz 1. bytu 10. rundovního klíče AES01 za použití správné hodnoty X i Y. Nyní je již korelace správného klíče jasně patrná.

lací pro všech 256 hypotéz 2. bytu 10. rundovního klíče je na obrázku 4.18. Na příčinu neúspěchu u bytů 1 a 5 jsem nepřišel. Nezdá se, že by zvýšení počtu vzorků spotřeby vedlo k lepšímu zvýraznění správné hodnoty. Graf korelací pro všech 256 hypotéz 5. bytu 10. rundovního klíče se zvýrazněnou správnou hodnotou je na obrázku 4.19.

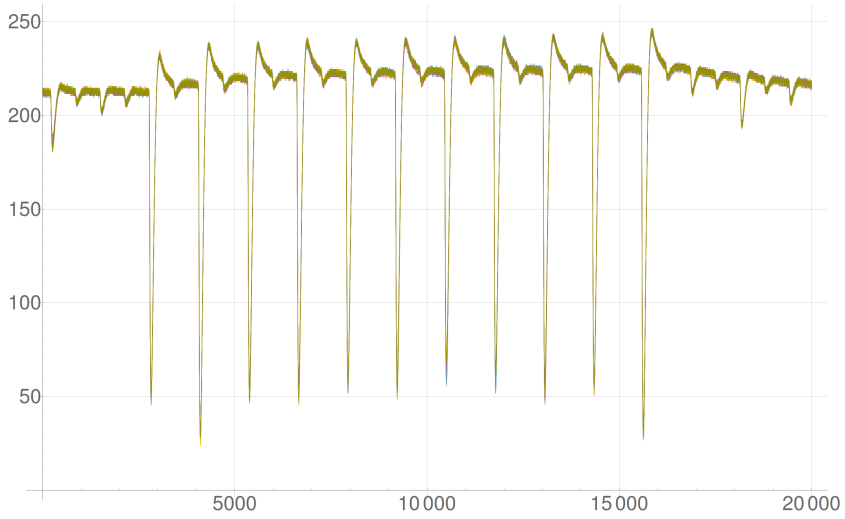
4.2.3 DPA na FPGA – varianty šifry AES Tomáše Zimmerhakla

Jak jsem již řekl v úvodu, Tomáš Zimmerhakl navrhl základní variantu šifry AES a následně na jejím základu dalších několik různých odolnostních variant. Jeho varianty nejsou myšleny jakožto odolné proti útoku pomocí DPA, místo toho jsou navržené tak, aby odolaly nějaké vnitřní poruše a byly schopné pokračovat i takovém případě. V následujícím textu popíši útok na základní variantu šifry AES Tomáše Zimmerhakla. Dále popíši útok na odolnostní varianty. Jedná se o varianty využívající prostorovou rendundanci na úrovni celého algoritmu a jedné rundy, poté časovou rendundanci na úrovni algoritmu a jedné rundy a nakonec informační rendundaci.

Pro porovnání odolnosti jednotlivých variant AES Tomáše Zimmerhakla je třeba zjistit minimální počet vzorků spotřeby, který ještě stačí na úspěšné prolomení klíče. Při prvních pokusech zaútočit na Tomášova data jsem zjistil, že nikde není třeba více než 2000 průběhů k prolomení, proto jsem u každé varianty, základní i dalších odolnostních, vždy změřil 100000 průběhů spotřeby a naměřené průběhy jsem pak rozdělil na úseky dlouhé 2000 průběhů, díky



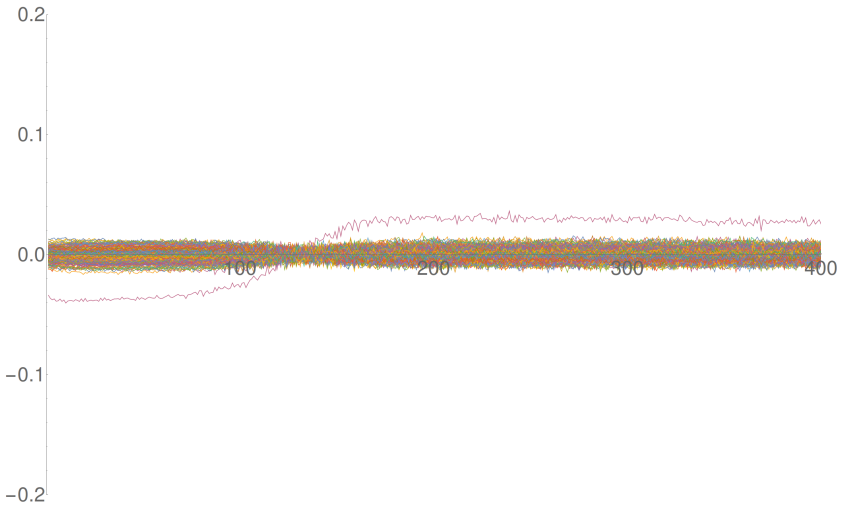
Obrázek 4.16: Prvních 10 vzorků spotřeby pro AES01 s deskou napájenou olověným akumulátorem. Vzorky na sebe narozdíl od verze napájené spínaným zdrojem lícují perfektně.



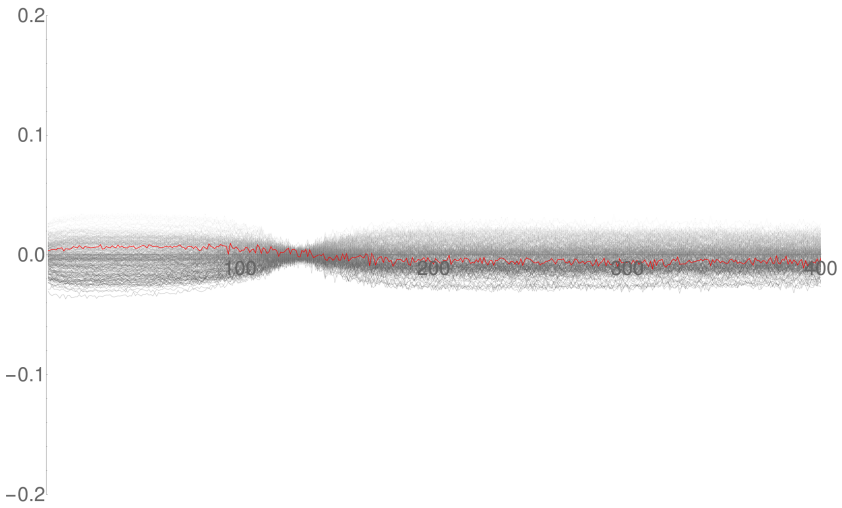
Obrázek 4.17: Prvních 10 vzorků spotřeby pro AES02.

čemuž jsem mohl provést 50 nezávislých útoků na různé průběhy s různými odpovídajícími šifrovými texty a výsledky zprůměrovat v jednu průměrnou minimální hodnotu, kterou už lze považovat za relevatní výsledek.

Útok na úsek o délce 2000 průběhů provádím po konzultaci s vedoucím práce následovně: nejprve zaútočím na celý průběh, případně pouze na časový úsek, v němž se pracovalo s hodnotou, ke které vytvářím model spotřeby.



Obrázek 4.18: Průběh korelačních koeficientů v čase pro všech 256 hypotéz 2. bytu 10. rundovního klíče AES02. Ostatní úspěšně prolomené byty měly velice podobné grafy korelací.



Obrázek 4.19: Průběh korelačních koeficientů v čase pro všech 256 hypotéz 5. bytu 10. rundovního klíče AES02. 1. byte měl podobný výsledek.

Dále již útočím pouze na místo v čase, které mělo největší korelaci. Postupně snižuji počet průběhů z původních 2000 po jednom a pokaždé ověřím, že útok na toto jediné místo v čase stále vyhrála správná hodnota. V momentně, kdy tato hodnota již není nejvyšší, si zapamatuji předchozí počet vzorků jakožto minimální hodnotu, která ještě stačí k úspěšnému prolomení.

Ve výsledku získám 50 minimálních hodnot pro každý byte 10. rundovního

klíče. Tyto hodnoty zprůměruji v jeden relevantní výsledek.

Při útocích na všechny varianty AES Tomáše Zimmerhakla používám jednotně klíč 0x00, 0xFF, 0x11, 0xEE, 0x22, 0xDD, 0x33, 0xCC, 0x44, 0xBB, 0x55, 0xAA, 0x66, 0x99, 0x77, 0x88, což vede na prolamování 10. rundovního klíče o hodnotě 0x98, 0xA0, 0x76, 0x3D, 0x64, 0xAB, 0xEB, 0x23, 0x43, 0xA4, 0x80, 0x89, 0x11, 0x5C, 0x18, 0x66.

Došlo ke snížení počtu vzorků spotřeby na jeden průběh z 20000 na 1000, díky čemuž je možné útok provádět nad celým průběhem s mnohem menším využitím operační paměti. I přes důkladné prostudování manuálu se mi nepodařilo zjistit, jak počet vzorků na jeden průběh ovlivnit. Pouze jsem této změny, která nastala pravděpodobně zapříčiněním někoho jiného, využil, jelikož díky ní zabírají naměřená data dvacetinu místa a není problém provést útok na kompletní průběh spotřeby. Později jsem od Jana Říhy[12] zjistil, že po vybrání „Save configuration“ a „Load configuration“ v programu SC Power Measurements[15] dojde k nastavení počtu vzorků na jeden průběh na hodnotu 1000.

Pro propojení modulu AES a modulu sériové linky používám všude obálku AES_WITH_UART02, která obsahuje registr pro klíč a možnost nahrát nový klíč. Všechny varianty AES Tomáše Zimmerhakla původně obsahovaly natvrdo zadrátovaný klíč na signálu KEY nejvyššího modulu. Tento signál jsem změnil na vstupní signál a přivedl na něj výstup registru pro klíč mé obálky. Tuto úpravu jsem udělal po konzultaci s vedoucím, neboť mí předchůdci měli problém implementaci AES na FPGA prolomit, pokud obsahovala konstantní klíč.

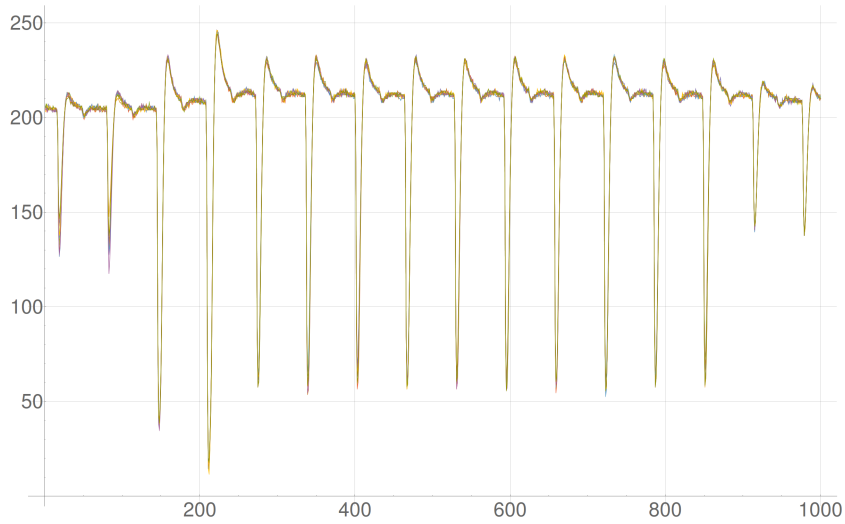
Kód 4.1: Rozhraní variant AES Tomáše Zimmerhakla po úpravě signálu KEY

```
entity AES_TOP_... is
 port (
  CLK               : in std_logic;
  RST               : in std_logic;
  INPUT_VALID       : in std_logic;
  IN_DATA           : in std_logic_vector(127 downto 0);
  IN_KEY            : in std_logic_vector (127 downto 0);

  DATA_REQUEST      : out std_logic;
  BUSY              : out std_logic;
  OUTPUT_VALID      : out std_logic;
  OUT_DATA          : out std_logic_vector(127 downto 0)
 );
end AES_TOP_...;
```

4.2.3.1 AES01

Varianta AES01 není odolnostní variantou, je to základní verze, která provede jednu rundu na jeden hodinový takt. Její rundovní registr je umístěn mezi operace Key Addition a Byte Substitution, tedy tak, jak ho útok pomocí Hammingových vzdáleností předpokládá. Průběh spotřeby prvních deseti měření je na obrázku 4.20. Útok na první byte 10. rundovního klíče pomocí 2000



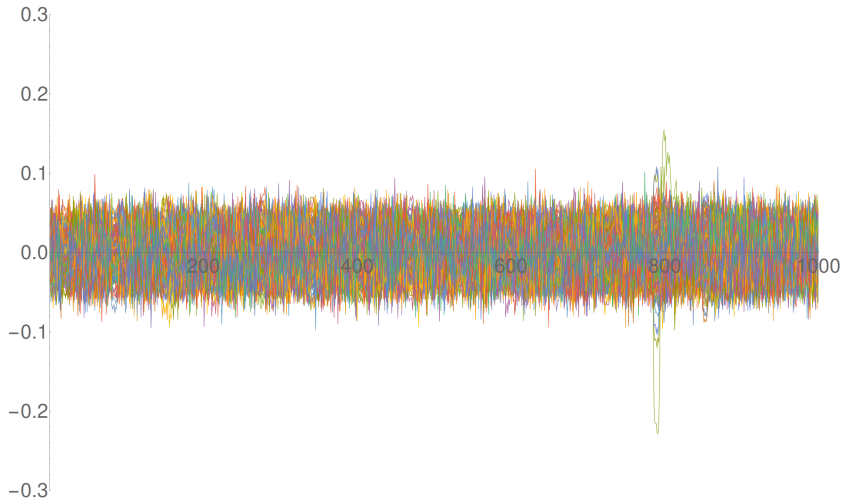
Obrázek 4.20: Průběh spotřeby během prvních deseti měření u AES01 Tomáše Zimmerhakla.

průběhů je na obrázku 4.21. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 750. - 850. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpovídajícího bytu klíče, jsou v tabulce 4.2.

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	192	297	325	166	266	319	271	250
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	263	267	253	355	264	405	345	362

Tabulka 4.2: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče. Průměr byl spočítán z 50 měření.

Zajímavostí, kterou jsem objevil, bylo, že jsem AES01 úspěšně prolomil i při použití obálky AES_WITH_UART bez registru pro klíč. Jan Říha, který pracuje na stejném zadání, ale na platformě Altera[12], podobnou implementaci bez registru pro klíč neprolomil. Nejspíše je to způsobeno rozdílnými

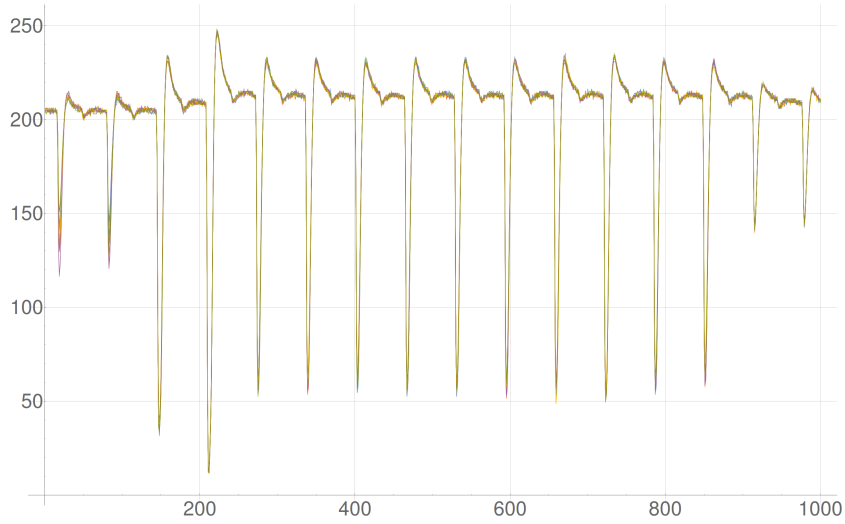


Obrázek 4.21: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. Vítěz je jasně patrný.

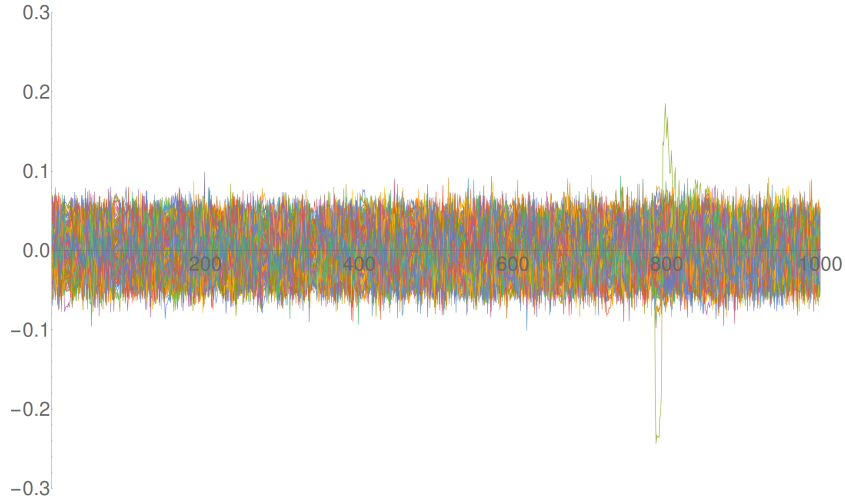
syntézními nástroji – ten, který používá Jan Říha, dost možná optimalizuje výsledný obvod až natolik pokročile, že výsledná podoba obvodu vůbec neodpovídá té, na kterou je myšlený útok, který provádíme. Naopak syntézní nástroje, které jsou součástí prostředí Xilinx ISE, takto pokročilou optimalizaci neprovádějí, a proto se konstantní klíč ve výsledném obvodu příliš neprojeví.

4.2.3.2 AES_1Aa

AES_1Aa je odolnostní varianta, která využívá prostorovou redundanci na celý algoritmus. Celé šifrování probíhá třikrát paralelně vždy se stejnými vstupními daty, ale každé se odehrává v jiné fyzické jednotce na čipu. Změna dat, ke kterým vytvářím model spotřeby, se děje ve třech registrech ve stejném okamžiku, a tak by dle mého názoru měl útok pomocí Hammingových vzdáleností, tak jak jej provádím, tuto variantu prolomit stejně spolehlivě jako nezabezpečenou variantu AES01. Průběh spotřeby prvních deseti měření je na obrázku 4.22. Útok na první byte 10. rundovního klíče pomocí 2000 průběhů je na obrázku 4.23. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 750. - 850. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpovídajícího bytu klíče, jsou v tabulce 4.3.



Obrázek 4.22: Průběh spotřeby během prvních deseti měření u AES_1Aa Tomáše Zimmerhakla je takřka identický s průběhem spotřeby u AES01.



Obrázek 4.23: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. Vítěz je jasně patrný.

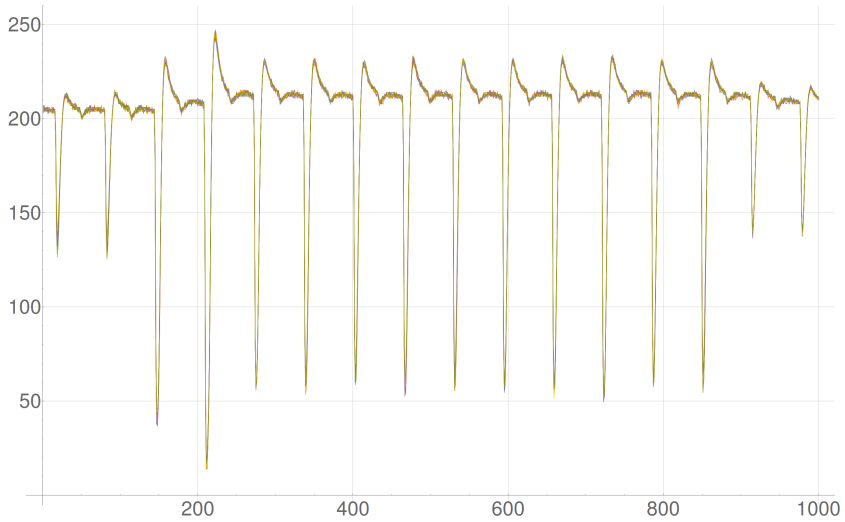
4.2.3.3 AES_1Ar

AES_1Ar je spolehlivostní varianta, která využívá prostorovou redundanci na úrovni jedné rundy. Každá runda probíhá třikrát paralelně vždy se stejnými vstupními daty, ale každá se odehrává v jiné fyzické jednotce na čipu. Výstup každé jednotky je pak porovnán s ostatními a za platný výstup se bere ten,

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	293	294	217	173	329	314	362	257
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	333	390	274	264	254	382	442	301

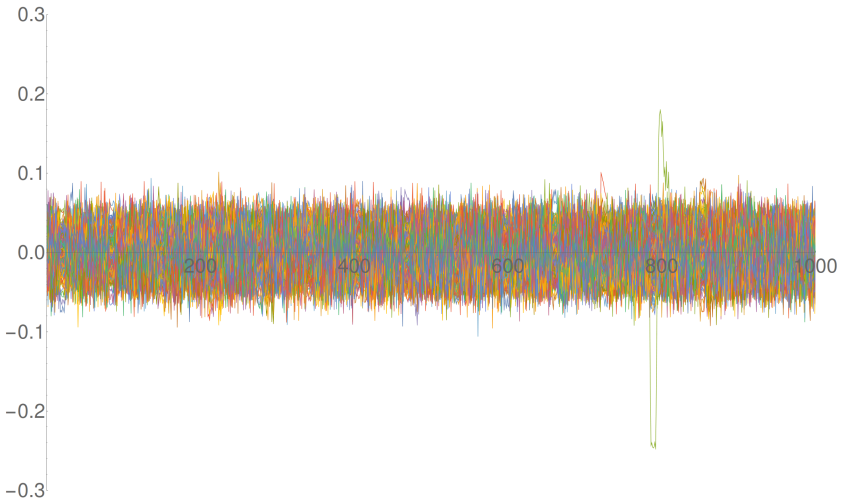
Tabulka 4.3: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče AES_1Aa. Průměr byl spočítán z 50 měření.

na kterém se shodly alespoň dvě ze tří jednotek. Výstup je následně uložen do rundovního registru, který je zde opět třikrát – pro každou jednotku zvlášť. Změna dat, ke kterým vytvářím model spotřeby, se děje ve třech registrech najednou, proto by i tato varianta měla jít prolomit, tak jako lze prolomit základní AES01 Tomáše Zimmerhakla. Průběh spotřeby prvních deseti měření je na obrázku 4.24. Útok na první byte 10. rundovního klíče pomocí 2000



Obrázek 4.24: Průběh spotřeby během prvních deseti měření u AES_1Ar Tomáše Zimmerhakla je takřka identický s průběhem spotřeby u AES01 a AES_1Aa.

průběhů je na obrázku 4.25. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 750. - 850. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpovídajícího bytu klíče, jsou v tabulce 4.4.



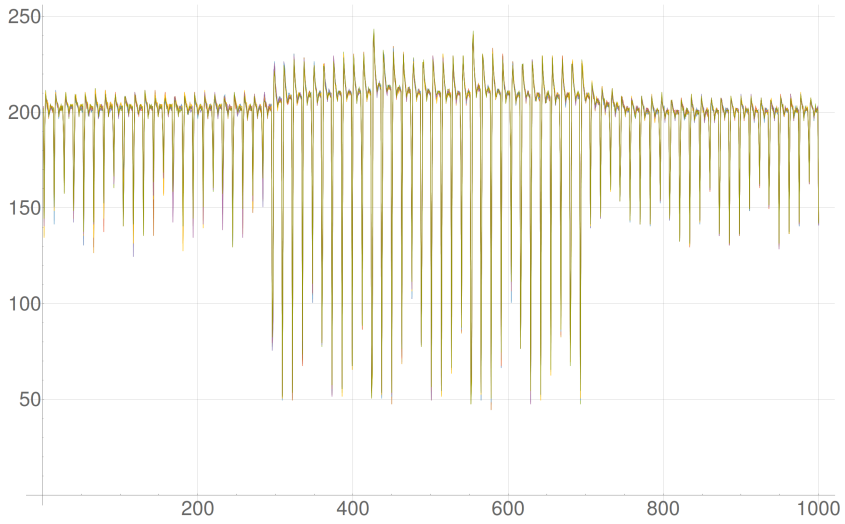
Obrázek 4.25: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. Vítěz je jasně patrný.

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	264	225	344	259	343	312	221	250
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	277	267	301	318	274	222	270	300

Tabulka 4.4: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče AES_1Ar. Průměr byl spočítán z 50 měření.

4.2.3.4 AES_1Ta

AES_1Ta je spolehlivostní varianta, která využívá časové redundance na úrovni celého šifrování. Vstupní data jsou tedy zašifrována třikrát za sebou, každý výsledek je uložen do jiného registru a po dokončení třetího šifrování jsou výstupy porovnány, přičemž za správný se považuje ta hodnota, která se na výstupu objevila alespoň dvakrát. Změna hodnot v registru, ke které vytvářím model spotřeby, se proto v této verzi uskuteční třikrát, ale pokaždé v jiném časovém okamžiku, čemuž odpovídá obrázek 4.27. Průběh spotřeby prvních deseti měření je na obrázku 4.26. Útok na první byte 10. rundovního klíče pomocí 2000 průběhů je na obrázku 4.27. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 350. - 750. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpovídajícího bytu klíče, jsou v tabulce 4.5.



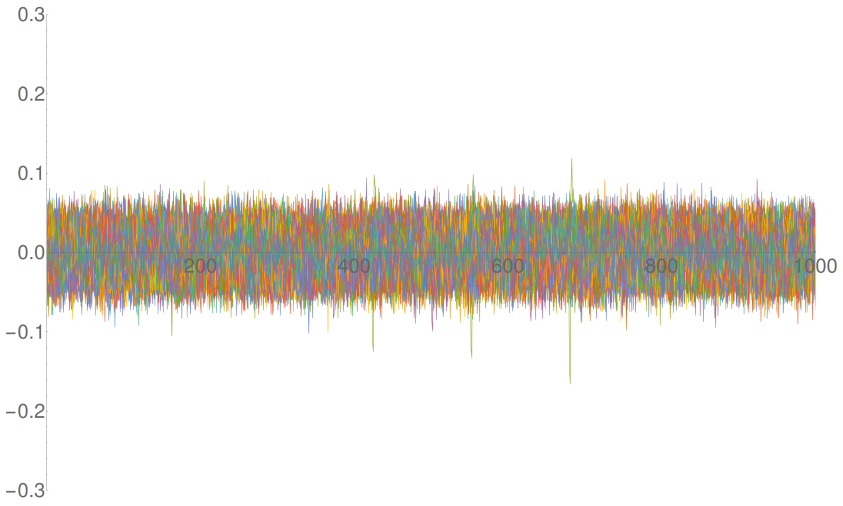
Obrázek 4.26: Průběh spotřeby během prvních deseti měření u AES_1Ta Tomáše Zimmerhakla. Průběh odpovídá implementaci – šifrování se celé třikrát identicky opakuje. Pro měření jsem na osciloskopu použil menší rozlišení, jelikož šifrování zde trvá třikrát déle.

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	379	229	383	273	343	315	258	261
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	384	270	315	352	296	255	312	236

Tabulka 4.5: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče AES_1Ta. Průměr byl spočítán z 50 měření.

4.2.3.5 AES_1Tr

AES_1Tr je spolehlivostní varianta, která využívá časové redundance na úrovni jedné rundy. Každá runda se proto provádí třikrát za sebou a výsledek je pokaždé uložen do jiného registru. Po dokončení třetího opakování rundy jsou výsledky v registrech porovnány, a za správný se považuje ten, který byl na výstupu alespoň dvakrát. Tato hodnota je poté zapsána do rundovního registru, čímž v něm dojde ke změně dat, ke které vytvářím model spotřeby. Naměřený průběh spotřeby prvních deseti měření je na obrázku 4.28. Útok na první byte 10. rundovního klíče pomocí 2000 průběhů je na obrázku 4.29. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 750. - 850. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpo-



Obrázek 4.27: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. S hodnotou, ke které vytvářím model spotřeby, se zde pracovalo třikrát, a to pokaždé v jinou dobu, což odpovídá třem nalezeným zvýšeným korelacím u jediného – správného bytu 10. rundovního klíče na obrázku.

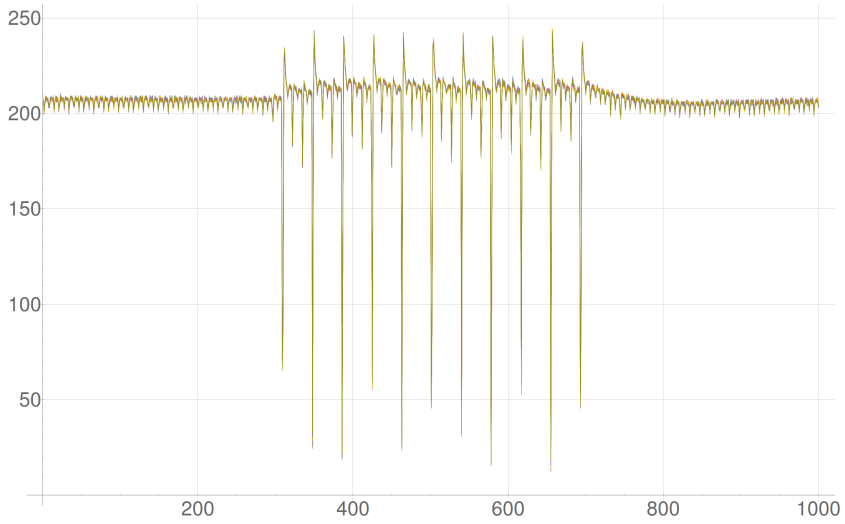
vídajícího bytu klíče, jsou v tabulce 4.6.

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	207	385	435	334	387	367	227	322
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	376	208	295	323	158	343	363	331

Tabulka 4.6: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče AES_1Tr. Průměr byl spočítán z 50 měření.

4.2.3.6 AES_1p

AES_1p je spolehlivostní varianta, která využívá paritního bytu u operace Byte Substitution. Ke vstupu operace SubBytes je vypočítána parita, vstup jde zároveň do asociativní paměti, která předpoví výsledný paritní bit. K výstupu operace SubBytes je též vypočtena parita a je srovnána s předpovězenou. Výstup jde zároveň do druhé asociativní paměti, která určí, jaká měla být parita na vstupu, a tato hodnota je porovnána s původní vypočtenou. Pokud v některém komparátoru nejsou hodnoty stejné, došlo k chybě, což je signalizováno na signálu WARNING.

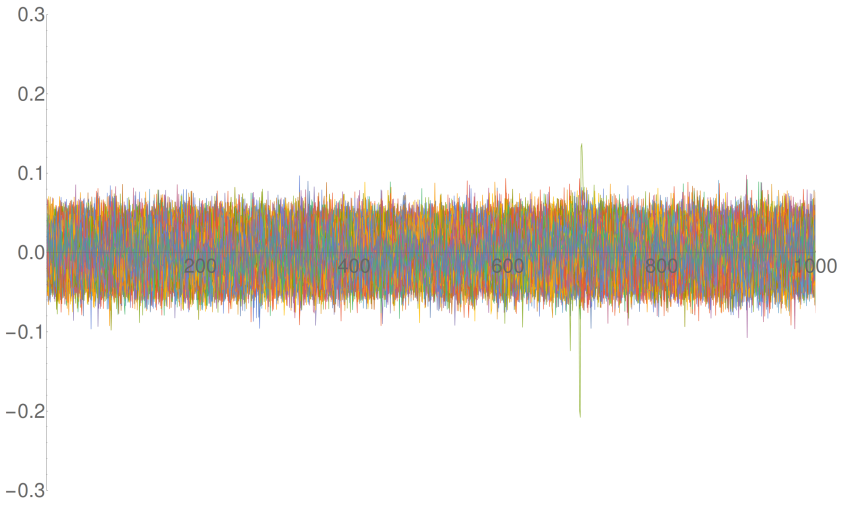


Obrázek 4.28: Průběh spotřeby během prvních deseti měření u AES_1Tr Tomáše Zimmerhakla. Průběh odpovídá implementaci – každá runda se opakuje třikrát, k největší změně spotřeby dochází při zapsání do rundovního registru, čímž se změní i stav celé kombinační logiky AES za ním.

Rundovní registr se zde nachází mezi operacemi Key Addition a Byte Substitution, stejně jako u základní varianty AES01. Ani zde by proto neměl být problém s prolamováním. Naměřený průběh spotřeby prvních deseti měření je na obrázku 4.30. Útok na první byte 10. rundovního klíče pomocí 2000 vzorků je na obrázku 4.31. Pro urychlení výpočtů provádím první útok nad úsekem dat mezi 750. - 850. vzorkem. Dále již vyberu vzorek s největší hodnotou korelace a útočím pouze na něj, přičemž snižuji pokaždé počet průběhů o jedna. Výsledné průměry minimálních hodnot z 50 měření, které ještě stačí ke správnému určení odpovídajícího bytu klíče, jsou v tabulce 4.7.

číslo bytu	1	2	3	4	5	6	7	8
minimální hodnota	197	249	225	256	350	249	416	411
číslo bytu	9	10	11	12	13	14	15	16
minimální hodnota	350	258	288	418	416	384	257	197

Tabulka 4.7: Průměrný minimální počet průběhů nutný k prolomení odpovídajícího bytu klíče AES_1p. Průměr byl spočítán z 50 měření.



Obrázek 4.29: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. Vítěz je jasně patrný.

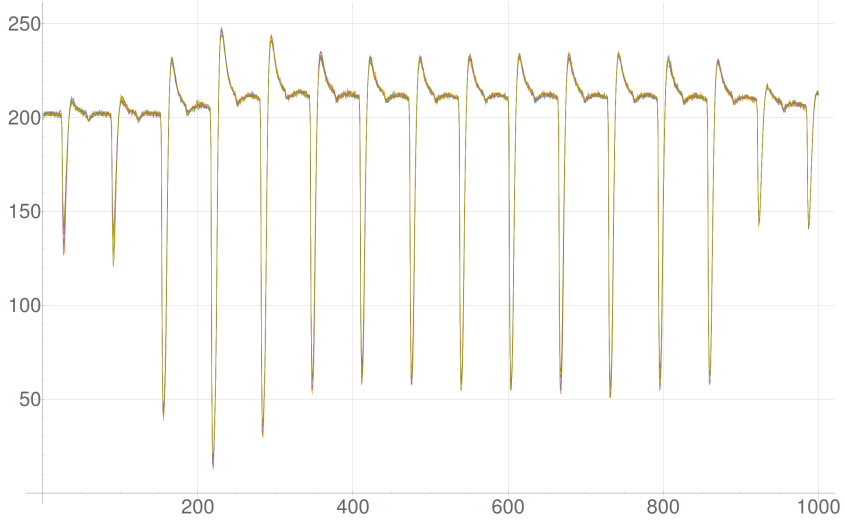
4.2.3.7 Porovnání odolnosti jednotlivých variant AES Tomáše Zimmerhakla

Celkový průměr průběhů nutný k úspěšnému prolomení každého bytu 10. rundovního klíče je zanesen v tabulce 4.8. Pro porovnání mých výsledků s výsledky

Varianta	Průměrné minimum	% základního AES01
AES01	287,5	100%
AES_1Aa	304,9	106,07%
AES_1Ar	284,2	98,85%
AES_1Ta	303,8	106,67%
AES_1Tr	316,3	110,02%
AES_1p	307,6	106,98%

Tabulka 4.8: Průměrný minimální počet průběhů v porovnání se základní variantou AES01.

Jana Říhy, který používal FPGA od Altery[12], úvádím ještě výsledky za použití mediánu a aritmetického průměru, kdy u každého měření vždy vyberu maximální hodnotu z minim všech šestnácti bytů a tu považuji za minimum pro celé jedno z padesáti měření. Jedná se tedy o hodnotu, která je minimální pro prolomení kompletního klíče. Medián a průměr je pak určen ze všech padesáti minim.

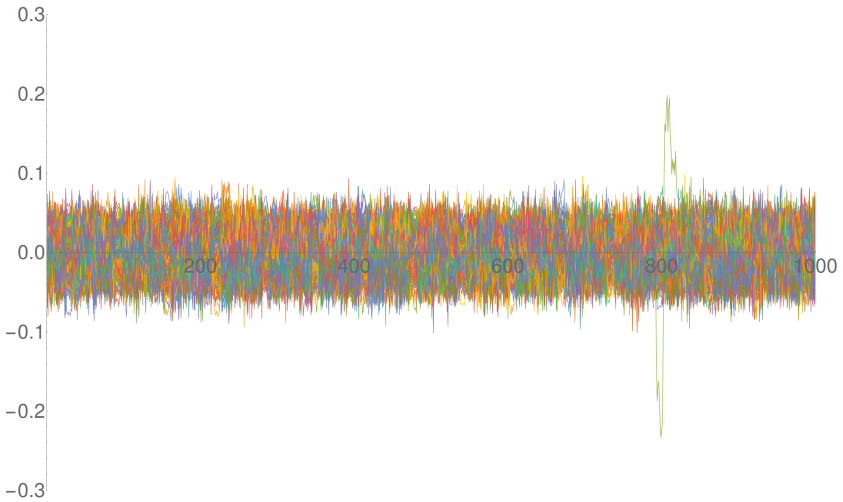


Obrázek 4.30: Průběh spotřeby během prvních deseti měření u AES_1p Tomáše Zimmerhakla je takřka identický s průběhem spotřeby u AES01, AES_1Aa a AES_1Ar.

Varianta	Medián	% AES01	průměr	% AES01
AES01	623,5	100%	659,70	100
AES_1Aa	681,5	109,30%	709,72	107,58%
AES_1Ar	575,0	92,22%	600,60	91,04%
AES_1Ta	638,0	102,33%	653,40	99,05%
AES_1Tr	745,0	119,49%	740,00	112,17%
AES_1p	687,5	110,27%	714,60	108,32%

Tabulka 4.9: Medián minimálního počtu vzorků k prolomení celého desátého rundovního klíče v porovnání se základní variantou AES01.

Jak je z obou tabulek vidět, nelze říct, že by některá spolehlivostní varianta byla vůči útoku pomocí DPA nějak významně více nebo naopak méně odolná, než základní varianta AES01.



Obrázek 4.31: Průběh korelačních koeficientů v čase pro první byte 10. rundovního klíče při korelaci celého průběhu spotřeby o 1000 vzorcích. Naměřeno bylo 2000 průběhů spotřeby. Vítěz je jasně patrný.

Kapitola 5

Budoucí práce

Během své práce jsem zjistil, že Jan Říha, který pracuje na stejném zadání, ale na platformě Altera[12], nedokázal prolomit šifru AES01 Tomáše Zimmerhakla, pokud v ní byl ponechán původní konstatní klíč. Naopak mně se tuto implementaci prolomit podařilo. Důvodem by mohl být vliv syntézních nástrojů. V případě Jana Říhy dost možná provádějí syntézní nástroje pro přípravek Altera natolik pokročilou optimalizaci výsledného obvodu, že se až příliš liší od uvažované podoby, na kterou je útok pomocí Hammingových vah na poslední rundu myšlený, a proto nebyl jeho útok úspěšný. Naopak syntézní nástroje od firmy Xilinx takovou optimalizaci provádět nemusejí, a proto nepředstavuje konstantní klíč v obvodu problém.

Má proto význam porovnat vliv různých syntézních nástrojů na odolnost proti útoku pomocí DPA.

Závěr

Ve své práci jsem se zabýval možnostmi aplikace rozdílové odběrové analýzy zejména na hardwarové implementace šifry AES na programovatelném hradlovém poli.

Nejprve jsem pro seznámení se s šifrou AES a metodou rozdílové odběrové analýzy implementoval šifru AES v jazyce C a následně svou implementaci prolomil na čipové kartě. Pro realizaci metody rozdílové odběrové analýzy jsem vytvořil několik variant skriptů v programu Mathematica.

Pro realizaci komunikace modulu AES s počítačem přes sériovou linku jsem vytvořil obálku, která sloužila jako prostředník pro komunikaci mezi modulem šifry a modulem sériové linky.

Dále jsem vytvořil dvě implementace šifry AES. První implementaci se mi zpočátku nedařilo prolomit, proto jsem vytvořil druhou, která měla být svým návrhem vhodnější k útoku pomocí Hammingových vzdáleností na poslední rundu tak, jak jsem jej prováděl. Tu se mi prolomit nepodařilo, přestože útok přesně odpovídal její vnitřní podobě. Naopak první variantu se mi nakonec prolomit podařilo, jelikož jsem útok upravil tak, aby více odpovídal jejímu vnitřnímu návrhu.

Nakonec jsem zkoumal odolnost spolehlivostních variant šifry AES Tomáše Zimmerhakla oproti jeho základní verzi. Prolomil jsem základní variantu a poté i další verze zabezpečené různými způsoby proti poruše. Zjistil jsem, že nikde není třeba více než 2000 průběhů spotřeby k úspěšnému prolomení. Změřil jsem proto u každé varianty 100000 průběhů spotřeby. Naměřené průběhy jsem rozdělil na úseky dlouhé 2000 průběhů, díky čemuž jsem mohl provést 50 nezávislých útoků na různé průběhy s různými odpovídajícími šifrovými texty. Každý úsek lze považovat za rozdílná data získaná při šifrování rozdílných vstupů. Vypočítal jsem korelační koeficienty pro časový úsek, ve kterém se pracovalo s hodnotou, ke které vytvářím model spotřeby. Dále jsem počítal korelační koeficient již pouze pro vzorek, který měl při prvním výpočtu nejvyšší korelační koeficient. Snižoval jsem počet průběhů pokaždé o jedna, až dokud jsem nenašel hodnotu, při které zvítězil nesprávný byte klíče. Předchozí

hodnotu jsem uložil jako nalezené minimum. Nalezená minima jsem zprůměroval ve výsledné hodnoty náležející ke každé variantě šifry. K minimálním hodnotám nezbytným k prolomení kompletního klíče jsem určil medián.

Podle průměrů bylo největší zvýšení odolnosti oproti základní variantě šifry AES o 10,02%, a to u varianty využívající časovou rendundanci na úrovni jedné rundy. Naopak verze využívající prostorovou rendundanci na úrovni rundy byla méně odolná, a to o 2,15%. Z výsledných dat vyplývá, že varianty zabezpečené proti poruše pomocí prostorové, časové a informační redundance nelze považovat za významně více či méně odolné proti útoku pomocí rozdílové odběrové analýzy než základní variantu.

Literatura

[1] PAAR, C.; PELZL, J.: *Understanding Cryptography.* Springer-Verlag, 2010, ISBN 978-3-642-04100-6.

[2] BUČEK, J.; NOVOTNÝ, M.: *Přednášky a cvičení předmětu Bezpečnost a technické prostředky (MI-BHW)* [online]. [cit. 2017-03-09]. Dostupné z: `https://edux.fit.cvut.cz/courses/MI-BHW/`

[3] KOCHER, P.; JAFFE, J.; JUN, J.: *Introduction to differential power analysis - Crypto 99 Proceedings, Lecture Notes in Computer Science Vol. 1666.* Springer-Verlag, 1999, ISBN 978-0-387-30857-9.

[4] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY: *Announcing the ADVANCED ENCRYPTION STANDARD (AES)* [online]. 2001, [cit. 2017-03-10]. Dostupné z: `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`

[5] BRIER, E.; CLAVIER, C.; OLIVIER, F.: *Correlation Power Analysis with a Leakage Model.* Volume 3156 of the book series Lecture Notes in Computer Science (LNCS), Springer-Verlag, 2004, doi:10.1007/978-3-540-28632-5_2.

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

[7] PAAR, C.: *Implementation of Cryptographic Schemes 1* [online]. 2015, [cit. 2017-03-15]. Dostupné z: `https://www.emsec.rub.de/media/attachments/files/2015/09/IKV-1_2015-04-28.pdf`

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

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

[10] *Langer EMV-Technik PA 303 BNC preamplifier* [online]. [cit. 2017-05-10]. Dostupné z: `https://www.langer-emv.de/en/product/preamplifier/37/pa-303-bnc-set-preamplifier-100-khz-up-to-3-ghz/519`

[11] *Spartan-3E Starter Kit Board User Guide* [online]. 2009, [cit. 2017-05-10]. Dostupné z: `https://reference.digilentinc.com/_media/s3e:s3estarter_ug.pdf`

[12] ŘÍHA, J.: *Útok rozdílovou odběrovou analýzou na implementaci algoritmu AES na platformě Altera. Bakalářská práce.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[13] BUČEK, J.; NOVOTNÝ, M.; ŠTĚPÁNEK, F.: *Practical Session: Differential Power Analysis for Beginners* [online]. 2014, [cit. 2016-05-22]. Dostupné z: `http://rozvoj.cvut.cz/main/Lisbon`

[14] *Advanced Serial Port Terminal* [online]. 2011, [cit. 2017-05-10]. Dostupné z: `https://www.eltima.com/products/serial-port-terminal/`

[15] BUČEK, J.; VÝLETA, P.: *Power consumption measurement* [online]. [cit. 2017-05-14]. Dostupné z: `https://edux.fit.cvut.cz/courses/MI-BHW/tutorials/dpa_measurement`

[16] *Agilent Technologies InfiniiVision 7000A Series Oscilloscopes* [online]. 2012, [cit. 2017-05-10]. Dostupné z: `http://literature.cdn.keysight.com/litweb/pdf/5989-7736EN.pdf?id=1373609`

[17] TUCCI, P.: *Java Smart Card Explorer* [online]. [cit. 2017-05-14]. Dostupné z: `https://sourceforge.net/projects/jsmartcard/`

Příloha **A**

Seznam použitých zkratek

AES Advanced Encryption Standard

FPGA Field Programmable Gate Array

DPA Differential Power Analysis

CPA Correlation Power Analysis

VHDL VSIC Hardware Description Language

UART Universal Asynchronous Receiver/Transmitter

Příloha **B**

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `measurements`
  - `AES_TZ_data.zip` ..... naměřená data pro AES Tomáše Zimmerhakla
    - `AES01` ..... naměřená data pro AES01 + výsledky
    - `AES1Aa` ..... naměřená data pro AES1Aa + výsledky
    - `AES1Ar` ..... naměřená data pro AES1Ar + výsledky
    - `AES1Ta` ..... naměřená data pro AES1Ta + výsledky
    - `AES1Tr` ..... naměřená data pro AES1Tr + výsledky
    - `AES1p` ..... naměřená data pro AES1p + výsledky
  - `AES_data.zip` ..... naměřená data pro mé varianty šifry AES
    - `AES01` ..... naměřená data pro AES01
    - `AES02` ..... naměřená data pro AES02
- `src`
  - `AES_c`
    - `BHW_SOSSE` ..... program pro čipovou kartu
    - `main.c` ..... AES v c samostatně
  - `AES_vhdl`
    - `AES01` ..... zdrojové kódy AES01
    - `AES02` ..... zdrojové kódy AES02
  - `AES_WITH_UART_vhdl`
    - `01` ..... zdrojové kódy AES_WITH_UART01
    - `02` ..... zdrojové kódy AES_WITH_UART02
  - `mathematica` ..... skripty implementující DPA v Mathematice
- `text` ..... elektronická verze bakalářské práce
  - `src` ..... zdrojové soubory pro LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Nastavení osciloskopu během měření spotřeby FPGA

	ver. rozlišení	hor. rozlišení	posun	vzorků/průběh
AES01	1 μs	260 mV	26.50 μs	20000
AES02	1 μs	260 mV	26.50 μs	20000
AES01	1 μs	260 mV	26.60 μs	1000
AES_1Aa	1 μs	260 mV	26.60 μs	1000
AES_1Ar	1 μs	260 mV	26.60 μs	1000
AES_1Ta	5 μs	270 mV	33.55 μs	1000
AES_1Tr	5 μs	240 mV	33.55 μs	1000
AES_1p	1 μs	290 mV	26.52 μs	1000

Tabulka C.1: Nastavení osciloskopu během měření spotřeby FPGA. Nahoře mé varianty šifry AES, dole varianty šifry AES Tomáše Zimmerhakla.