FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Vliv odstupu signálu od šumu na úspěšnost útoku postranním kanálem
Student:	Adam Rektořík
Vedoucí:	Ing. Petr Socha
Studijní program:	Informatika
Obor / specializace:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je experimentální vyhodnocení vlivu odstupu signálu od šumu (signal-to-noise ratio) na úspěšnost útoku postranním kanálem.

- Navrhněte a realizujte útok postranním kanálem na implementaci šifry PRESENT na mikrokontroleru a FPGA.
- Navrhněte a realizujte měření odstupu od šumu pro navržené útoky.
- Pokuste se ovlivnit odstup od šumu pomocí vnějších (např. odstínění měřeného zařízení, či naopak cílené rušení mobilním telefonem, aj.) i vnitřních vlivů (např. volně běžící časovače/generátory náhodných čísel uvnitř čipu, aj.).
- Pomocí validní experimentální metodologie (např. míra úspěšnosti útoku, entropie hádání, aj.) vyhodnoťte vliv odstupu signálu od šumu na úspěšnost navržených útoků, včetně vlivu konkrétních opatření navržených v předchozím bodu.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 23. září 2021 v Praze.

Bakalářská práce

VLIV ODSTUPU SIGNÁLU OD ŠUMU NA ÚSPĚŠNOST ÚTOKU POSTRANNÍM KANÁLEM

Adam Rektořík

Fakulta informačních technologií
Katedra číslicového návrhu
Vedoucí: Ing. Petr Socha
12. května 2022

Odkaz na tuto práci: Rektořík Adam. *Vliv odstupu signálu od šumu na úspěšnost útoku postranním kanálem.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svým rodičům, kteří mě v mém studiu vždy podporovali. Rád bych také poděkoval mému vedoucímu práce Ing. Petru Sochovi, který byl velice ochotný a nápomocný při tvorbě.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

Abstrakt

Šifra PRESENT šifruje a dešfruje blok textu. Korelační odběrová analýza je metoda útoku na šifru, která vytvoří model spotřeby energie a tento model koreluje se skutečnou spotřebou při šifrování. Během šifrování mohou nastat rušivé elementy, které by zamaskovaly velikost korelace. Odstup signálu od šumu určuje poměr požadovaného signálu od nežádoucího jevu. Při výskytu řušivých elementů poklesne odstup signálu od šumu. Práce cílenou změnou velikosti šumu zkoumá vztah velikosti odstupu signálu od šumu a úspěšnost korelační analýzy na prolomení šifry.

Klíčová slova PRESENT, XMEGA, SPARTAN 6, FPGA, SNR, CPA, kryptografie, postranní kanály, vestavné systémy, mikrokontroler, FPGA

Abstract

PRESENT cipher encrypts and decrypts a text block. Correlation power analysis introduces power consumption model to correlate against real consumption during encryption. During encryption can be introduces undesired phenomenons, that could decrease the correlation. Signal-to-noise ratio is a ratio of a signal against and undesired events. In the event of increase of undesired phenomenons, signal-to-noise ratio will decrease. This work deliberately changes the power of noise and encloses the relationship between signal-to-noise ratio and correlation power attack success rate.

Keywords PRESENT, XMEGA, SPARTAN 6, FPGA, SNR, CPA, Cryptography, side-channels, embedded systems, microcontroller, FPGA

Seznam zkratek

ASCII	American Standard Code for Information Interchange
AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DES	Data Encryption Standard
GPIO	General-purpose input/output
LSB	Least Significant Bit
MSB	Most Significant Bit
SNR	Signal to Noise Ratio

Kapitola 1

Úvod do problematiky

1.1 PRESENT

Šifra PRESENT [1] vznikla v reakci na rozvíjející se trendy v oblasti mikrokontrolerů a programovatelných logických hradel, kterými jsou především vývoj nízkokapacitních zařízení, který s sebou přináší omezení výpočetní a paměťové kapacity. Jedná se o symetrickou substitučně permutační [2] blokovou šifru [3], která klade důraz na minimalizaci paměťové a časové náročnosti na zařízení při šifrování. Před příchodem této šifry byla k dispozici především symetrická bloková šifra Rijndael [4]. Ta se 26.5.2002 stala novým šifrovacím standardem AES [5]. Šifra PRESENT vznikla jako hardwarově odlehčená alternativa pro velmi malá výpočetní zařízení, kde již použití šifry Rijndael nebylo z výpočetních a časových důvodů vhodné.

Pomocí této šifry lze zašifrovat blok textu délky 64 bitů pomocí tajného šifrovacího klíče délky 80 nebo 128 bitů. Zašifrování textového bloku probíhá v 31 cyklech, které se nazývají rundy. Před začátkem šifrování dojde k vygenerování rundovních klíčů. Ty vzniknou výpočtem z tajného klíče a každý z těchto klíčů bude využit pro šifrování v jednotlivých rundách k postupnému šifrování textu. Šifrovací runda provádí nad blokem textu 3 specifické operace. První operací je exkluzivní OR mezi příslušným rundovním klíčem a šifrovaným blokem textu. Dále proběhne substituční vrstva, která nahrazuje čtveřice bloku jinou hodnotou podle specifického pravidla. Třetím krokem je vrstva permutační, ta v bloku šifrovaného textu prohazuje jednotlivé bity, opět podle specifického pravidla. Na závěr šifrování, po provedení všech 31 rund, ještě dojde k operaci exklzivní OR s posledním rundovním klíčem. Průběh šifrování přibližuje následující ukázka kódu:

Výpis kódu 1.1 Průběh šifry PRESENT

```
generovaniRundovnichKlicu ()
for ( i = 1; i <= 31; i++ )
{
    XORRundovnihoKlice ( blok, rundovniKlic[i] )
    substitucniVrstva ( blok )
    permutacniVrstva ( blok )
}
XORRundovnihoKlice ( blok, klic[32] )
```

1.1.1 Generování rundovních klíčů

K zašifrování bloku potřebujeme znát tajný šifrovací klíč. Z tohoto klíče se generuje 32 rundovních klíčů. Ty jsou využity v jednotlivých šifrovacích rundách, kromě posledního. Poslední z těchto klíčů slouží k závěrečné operaci exkluzivní OR s blokem. Pro výpočet prvního rundovního klíče potřebujeme znát pouze tajný klíč. Pro vygenerování zbylých rundovních klíčů potřebujeme znát hodnotu rundovního klíče předchozího a index, pro kterou rundu má být daný rundovní klíč využit. Jelikož generování nezávisí na jiných faktorech, než na znalosti tajného klíče, lze všechny tyto rundovní klíče vygenerovat již před začátkem šifrování.

Již bylo zmíněno, že šifra existuje ve dvou verzích. První verze má délku tajného klíče 80 bitů a druhé verze přijímá délku tajného klíče 128 bitů. Pravidla generování rundovních klíčů se pro tyto dvě délky liší. Obě varianty mají společnou délku rundovních klíčů, která je 64 bitů. V této kapitole je popsán algoritmus generování rundovních klíčů z tajného klíče délky 128 bitů, jelikož tato varianta bude také později realizovaná k měření. Předpis generování rundovních klíčů pro verzi šifry s délkou klíče 80 bitů lze nalézt v dokumentaci [1].

Proces generování klíčů má 4 kroky. První rundovní klíč se podle těchto kroků generuje z tajného klíče a ostatní se generují z rundovního klíče předchozího. Nejdříve dojde k cyklickému posunu rundovního klíče o 61 pozic doleva. Po této operaci budou čtveřice bitů na pozicích 127–124 a 123–120 nahrazeny jinou hodnotou podle předpisu substituční tabulky 1.1, přičemž MSB se v klíči standardně nachází na pozici úplně vlevo a LSB úplně napravo [6]. Dále je nad bity na pozicích 66-62 provedena operace exkluzivní OR s binární reprezentací indexu rundy, pro kterou je klíč generován.

1.1.2 Substituční Vrstva

Tato vrstva přijímá šifrovaný blok textu, který má délku 64 bitů a tento blok modifikuje. Ve vrstvě se šifrovaný blok rozdělí na 16 čtveřic sousedících bitů. Tyto čtveřice jsou nahrazeny na základě hodnot jednotlivých bitů novou hodnotou podle následující substituční tabulky.

Tabulka 1.1 Substituční tabulka

Vstup	0000	0001	0010	0011	0100	0101	0110	0111
Výstup	1100	0101	0110	1011	1001	0000	1010	1101
Vstup	1000	1001	1010	1011	1100	1101	1110	1111
Výstup	0011	1110	1111	1000	0100	0111	0001	0010

1.1.3 Permutační vrstva

Permutační vrstva šifrování také přijímá a modifikuje šifrovaný blok. A to přesunem bitů v rámci bloku na různé pozice. Pravidla pro přesun bitů jsou předem jasně daná v permutační tabulce 1.2, bit na pozici i je přesunut na pozici $P(i)$. Je potřeba si uvědomit, že k přesunu musí dojít pro všechny pozice najednou. Nelze procházet blok a postupně v něm provádět bitové přesuny. Při tomto postupu by docházelo posunu bitů na chybné pozice. Například bit na pozici 1 by se po vykonání permutační vrstvy měl nacházet na pozici 16, bit z pozice 16 by se měl po přesunu nacházet na pozici 4. Při postupném posunu by se první bit posunul na pozici 16 a tento stejný bit by byl následně posunut na pozici 4. Ve výsledku by tedy bit z první pozice skončil na pozici čtvrté.

Tabulka 1.2 Permutační tabulka

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

1.2 Kryptoanalýza

Algoritmy blokových šifer si zakládají na znalosti tajného klíče, pomocí kterého nad textovým blokem provádí specifické šifrovací operace. Předpokládá se, že klíč je tajný a jeho hodnotu zná pouze oprávněný majitel. Při návrhu těchto typů šifer je kladen důraz na to, aby při znalosti klíče bylo možné efektivně šifrovat a dešifrovat texty, ale hlavně, aby šifrování a dešifrování textů bez znalosti klíče nebylo možné. Proto můžeme v substitučně permutačních šifrách nalézt mimo XORu s tajným klíčem substituční a permutační vrsty. Ty jsou zavedeny pro konfuzi a difuzi potenciálního útočníka, který by se snažil tajný klíč získat. Metod získání tajného klíče existuje několik.

1.2.1 Známé útoky

K základním technikám patří útok hrubou silou, ve kterém dochází k hádání hodnoty klíče. Útočník zná hodnotu nějaké otevřeného textu a jeho zašifrovanou ekvivalentu. Postupně prochází všechny možné hodnoty klíčů, pomocí nich šifruje textový blok a porovnává výsledek oproti zašifrovanému textu. Útok lze koncipovat i opačně, kdy bude útočník dešifrovat zašifrovaný text a porovnávat výsledek s blokem otevřeného textu. Jako ochranu proti této metodě útoku dnešní šifry používají klíče velikých délek, aby navýšily možné hodnoty klíče. K vyzkoušení všech možných hodnot klíče délky 128 bitů hrubou silou je početné velmi náročné. Dnešní počítače nedisponují dostatečnou výpočetní kapacitu k získání klíče touto metodou v rozumném čase.

Mezi pokročilejší metody kryptoanalýzy patří matematické zkoumání vztahů mezi otevřenými i zašifrovanými bloky textu. Útočník má k dispozici bloky otevřených textů a k nim odpovídající zašifrované texty. Nad nimi provadí matematickou analýzu k získání klíče. Diferenční analýza [7] zkoumá rozdíly dvojic otevřeného textu a rozdíly dvojic k nim odpovídajících šifrových textů. Analýzou velikosti rozdílů dojde k přiřazení pravděpodobností možným klíčům. Klíč s největší pravděpodobností je následně identifikován jako správný tajný klíč. Lineární analýza [7] využívá matematickou metodu lineární aproximace dvojic otevřeného textu, dvojic šifrového textu a dvojic částí klíče k zformulování modelu činnosti částí blokové šifry. Aproximace funkce části šifry se společně s otevřeným a šifrovaným textem využije pro získání části klíče. Po získání části klíče lze použít lineární aproximaci k získání zbývajících částí klíče nebo dopočítat zbytek klíče hruhou silou. K použití těchto metod kryptoanalýzy obvykle potřebujeme velký soubor dat otevřených textů a k nim odpovídajících šifrových textů.

Další metodou získání tajného klíče je technika odběrové analýzy. Útočník zasílá zařízení textové bloky k zašifrování a při provádění algoritmu šifrování provádí měření konkrétní fyzikální veličiny zařízení. Analýzou chování veličiny pro daný textový blok se snaží získat tajný klíč. Metoda SPA [8] přímo interpretuje spotřebu proudu zařízení při šifrování. K tomuto útoku se předpokládá detailní znalost konkrétního zařízení a šifrovacího algoritmu. Diferenční odběrová analýza DPA [8] na základě znalosti vstupního textu a části šifrovacího algoritmu rozdělí hodnotu veličiny naměřenou při šifrování do dvou skupin podle očekávané spotřeby a vyhodnocuje rozdíl

těchto skupin. Korelační odběrová analýza [9] porovnává naměřenou hodnotu s odhadnutou pro různé kandidáty na klíč.

1.2.2 Útok CPA

Návrh korelační odběrové analýzy tvoří značnou část této práce a proto její princip bude vysvětlen podrobně. Jedná se o metodu odběrové analýzy. Při analýze dochází k opakovanému zasílání otevřených textových bloků zařízení, které nad těmito texty provádí šifrovací algoritmus. Při provádění algoritmu dochází k zaznamenávání spotřeby energie zařízení.

Každé elektronické zařízení potřebuje ke své funkci zdroj napájení. Proud dodává komponentám, které zajišťují běh zařízení, ty mají konstantní spotřebu energie v průběhu času. Dále zařízení potřebuje dodat energii datové struktuře vždy, když je potřeba v datové strutuře změnit hodnotu. Velikost dodatečné vynaložené energie při změně v datové struktuře je závislá na velikosti změny v dané datové struktuře.

Pro útok předpokládáme znalost části šifrovacího algoritmu. Část algoritmu provádí logickou operaci nad blokem textu pomocí tajného klíče, případě s nějakou částí tohoto klíče. Výsledek operace uloží do datové struktury. K útoku potřebujeme být schopni vyjádřit hodnotu bloku použitého jako vstup části šifry, případně hodnotu výstupu části šifry v závislosti na části klíče jako parametru.

Při útoku zformulujeme Hammingův model spotřeby energie. Model vyjadřuje odhad velikosti spotřeby zařízení při změně pracovních proměnných dané části šifry pracující s tajným klíčem. Tento model je vytvořen pro každou možnou hodnotu tajného klíče. Vyjádřit všechny možné hodnoty celého tajného klíče je z výpočetního hlediska velmi náročné, proto je vhodné znát část šifrovacího algoritmu pracujícího pouze s částí tajného klíče. Útok hledá provázanost modelu se skutečnou naměřenou spotřebou energie. K vyjádření míry korelace se využívá Parsonův korelační koeficient. Jako správný klíč, popřípadě část klíče, bude identifikován ten, který má největší korelaci mezi vytvořeným modelem spotřeby a skutečnou spotřebou.

1.2.2.1 Hammingův model spotřeby

Hammingova váha slova je počet bitů v binárním zápisu slova, které mají hodnotu 1. Mějme tedy slovo D, Hamingovu váhu tohoto slova vyjádříme jako $H(D) = \sum_{i=0}^{m-1} d_i$, kde m značí délku slova, d_i je bit slova na pozici i.

Hammingova vzdálenost [10] mezi dvěma slovy vyjadřuje počet lišicích se bitů v binární zápisu těchto slov. Tato hodnota je vyjádřena jako Hammingova váha výsledku logické operace exkluzivní OR nad těmito dvěma slovy. Mějme tedy slova D a R. Hammingova vzdálenost těchto slov je $H(D,R) = H(\ D \bigoplus R\ )$. Operace exkluzivní OR je komutativní [11], Hammingova vzdálenost slova D od slova R je stejná, jako Hammingova vzdálenost slova R od slova D. Pokud bude mít jedno slovo nulovou hodnotu, Hammingova vzdálenost mezi tímto nulovým slovem a slovem druhým bude Hammingova váha druhého slova.

Hammingův model spotřeby předpokládá lineární závislost mezi nárůstem spotřeby energie a Hammingovou vzdáleností zapisovaného slova a slova nahrazovaného.

1.2.2.2 Korelační koeficient

Pro vyjádření velikosti lineární závislosti mezi dvěmi veličinami lze využít Pearsonův korelační koeficient [12]. Ten je definován vztahem $corr(x,y) = \frac{cov(x,y)}{\sigma_x \sigma_y}$. Výpočet koeficientu je tedy kovariance dvou veličin normalizovaná jejich směrodatným odchylkami.

Kovarianci dvou veličin x a y definujeme jako $cov(x,y) = \sum[(x-\overline{x})(y-\overline{y})]$, kde $\overline{x}$ značí střední hodnotu veličiny x a $\overline{y}$ značí střední hodnotu hodnotu veličiny y.

Už tato kovariance udává lineární závislost veličin, která bude pozitivní v případě přímé úměry a negativní v případě nepřímé úměry. Také může být nulová, pokud nebyl nalezen lineární vztah mezi veličinami. Nemusí to nutně znamenat, že mezi veličinami vztah neexistuje, pouze nebyl nalezen lineárním koeficientem.

Směrodatná odchylka [13] udává míru variablility vztaženou k jednotce veličiny. Směrodatná odchylka je odmocnina rozptylu [14]. I rozptyl samotný udává míru variability. Rozptyl je definován vztahem: $\sigma_x^2 = \sum(x - \overline{x})^2$. K vyjádření rozptylu veličiny nejdříve vypočítáme její střední hodnotu. Poté provedeme rozdíl každého prvku veličiny vůči zjištěné střední hodnotě, který následně umocníme na druhou, aby všechny hodnoty rozdílu měly pozitivní hodnotu. Rozptyl vyjadřuje pouze vzdálenost prvku od střední hodnoty. Neřeší, kterým směrem je prvek od střední hodnoty vzdálen. Na závěr zjistíme střední hodnoty zjištěných vzdáleností, tím získáme průměrnou vzdálenost prvků od střední hodnoty, neboli rozptyl.

Tento výsledek nebere v potaz jednotku veličiny, pohlíží na míru variability pouze z matematického hlediska. Hodnota směrodatné odchylky pak bude odmocnina tohoto výsledku. Odchylka je vztažena k jednotce veličiny a je přímo úměrná s velikostí variability v datech.

Výběrový Pearsonův korelační koeficient lze tedy zapsat následovně

$$corr(x, y) = \frac{\sum[(x - \overline{x})(y - \overline{y})]}{\sqrt{\sum(x - \overline{x})^2 \sum(y - \overline{y})^2}}$$

1.2.2.3 Poměr signálu a šumu

K vyjádření, jak silný je v souboru dat očekávaný jev lze využít metriku poměru signálu a šumu [15] který, jak už název napovídá, porovnává míru signálu oproti nežádoucímu šumu v souboru dat. Pojmy signál a šum jsou velmi obecné, jejich definice záleží na konkrétním použití. Signál je nějaký očekávaný jev v souboru dat definovaný konkrétní fukncí. Šum naopak vyjadřuje jevy v souboru dat, které s definovanou funkcí nejsou spojeny.

Poměr lze matematicky zapsat jako $SNR = \frac{P_s}{P_s}$, kde P_s je míra signálu a P_n je míra šumu. Pokud bude toto číslo mít hodnotu větší než 1, míra požadovaného signálu je silnější, než míra nežádoucího šumu. A analogicky, při hodnotě SNR menší než 1 šum převažuje nad signálem. Poměr signálu a šumu lze také vyjádřit v jednotce decibelů, pro takovéto vyjádření platí vztah $SNR_{db} = 10 \log_{10} \frac{P_n}{P_s}$

1.3 Analýza návrhu

Měření budou prováděna pomocí sady ChipWhisperer Level 1 Kit od firmy Newae Technology Inc., která slouží k provádění odběrové analýzy na různých zařízeních. Sada se skládá ze dvou hlavních komponent. První komponentou je univerzální deska Chipwhisperer CW308 UFO Board. Univerzální ve smyslu, že do ní lze zapojit různá cílová zařízení, nad kterými uživatel plánuje provádět odběrovou analýzu. V práci bude provedeno měření na mikrokontroleru ATXmega128D4-AU a na zařízení FPGA Spartan 6 LX9, obě tyto zařízení budou zapojeny v desce UFO Board. Odběr spotřeby lze provést buď nad zařízením UFO Board samotným pomocí osciloskopu, nebo k ní lze připojit jiné vzorkující zařízení provádějící odběr spotřeby. Sada Level 1 Kit dodává pro odběr spotřeby zařízení desku Chipwhisperer-Lite, která bude k měření využita.

1.3.1 API rozhraní

Výrobce poskytuje API [16] rozhraní používající programovací jazyk Python, do kterého integruje svou open source knihovnu. Pomocí tohoto rozhraní lze ovládat vzorkující zařízení, a také komunikovat s cílovým zařízením. Rozhraní také obsahuje nástroje pro provádění odběrové

analýzy, umožňuje generování textových řetězců a obsahuje vhodné struktury pro reprezentaci naměřených dat, včetně možnosti jejich uložení. Generovat otevřené texty a klíče umožňuje třída `ktp.Basic ()`. K vygenerování třída obsahuje metodu `next ()`, návratovou hodnotou této metody jsou dvě pole velikosti 16 bajtů s pseudonáhodnými daty.

Naměřená data lze také ukládat do projektů. Nový projekt vytvoříme příkazem `new_project ( filename, overwrite )`. Parametrem je název projektu a specifikátor určující, zda má být nahrazen již existující projekt stejného jména. Do projektu lze uložit instance třídy Trace obsahující uspořádané čtveřice stop, otevřeného text, klíče a zašifrovaného textu. Příkazem `save ()` uložíme projekt zapsáním dat na disk. Uložený projekt otevřeme příkazem `open_project ( filename )`, paramerem je lokace uloženého projektu.

1.3.2 Připojení k vzorkujícímu zařízení

Zařízení Chiphisperer-Lite je k počítači připojeno pomocí Micro USB kabelu. Zařízení má zabudovaný USB kontroler pro komunikaci přes tento kabel a také obsahuje firmware, pomocí kterého komunikuje s zařízením, nad kterým provádí měření. K připojené desce se lze připojit pomocí API funkce `chipwhisperer.scope ()`. Ta vrátí instanci třídy obsahující veřené rozhraní právě připojené desky. Příkaz by měl automaticky rozpoznat připojenou desku a na základě detekovaného typu vrátit vhodnou instanci třídy dané desky. Pokud by k automatické detekci nedošlo, můžeme instanci manuálně specifikovat a to konkrétně u desky Chipwhisperer-Lite parametrem `scope_type = cw.scopes.OpenADC`.

Při zapojení více desek k počítači současně musí být v parametru funkce dále specifikováno sériové číslo daného zařízení. Při měření bude k počítači připojena k počítači pouze deska Chipwhisperer-Lite. Informace ke specifikaci sériového čísla při zapojení více desek může čtenář nalézt v dokumentaci API [16].

Ve veřejném rozhraní můžeme konfigurovat vzorkující desku. Nakonfigurování lze provést automaticky pomocí příkazu `scope.default_setup ()` konkrétní konfigurace bude zvolena podle rozpoznaného typu desky, pro kterou byla vytvořena instance rozhraní. Výpis některých proměnných automatického nastavení pro desku Chipwhisperer-Lite lze nalézt v tabulce 1.3.

Tabulka 1.3 Inicializace vzorkujícího zařízení

Proměnná	Hodnota
scope.adc.state	False
scope.adc.presamples	0
scope.adc.offset	0
scope.adc.samples	5000
scope.adc.trig_count	0
scope.clock.adc_src	clkgen_x4
scope.clock.adc_freq	29538459
scope.clock.clkgen_freq	7384615
scope.io.extclk_src	hs1
scope.trigger.triggers	tio4

Z tabulky lze vidět, že vzorkující zařízení využívá externí zdroj hodinového signálu v součinnosti s čtyřnásobným hodinovým násobičem. Také vidíme, že signál použitý jako vstup externího zdroje hodinového signálu je pin `hs1`, což je hodinový signál cílového zařízení. Ten ke generování signálu využívá krystalový oscilátor s rezonanční frekvencí 7,38 MHz. Capture board bude generovat hodinový signál s čtyřnásobnou frekvencí, tedy 29,54 MHz.

Vzorkování je při inicializaci nastaveno na hodnotu vypnuto. Signál spuštění a zastavení vzorkování je připojený k pinu `io4`. Jedná se o GPIO port, který je nastaven jako vstupní. Pomocí proměnných `presamples` a `offset` můžeme posunout začátek vzorkování.

Výpis kódu 1.2 Komunikace protokolu SimpleSerial

```
ktp = cw.ktp.Basic()
key, text = ktp.next()
target.simpleserial_write('k', key)
scope.arm()
target.simpleserial_write('p', text)
trace = scope.get_last_trace()
output = target.simpleserial_read('r', 16)
```

Proměnná `trigcount` je při inicializaci nastavena na hodnotu 0, do této proměné se po ukončení vzorkování uloží skutečný počet obdržených vzorků při aktivním spuštění vzorkování. Pomocí proměnné `samples` můžeme nastavit, kolik vzorků se má naměřit od spuštění vzorkování. Maximální počet vzorků závisí na modelu vzorkujícího zařízení. Chipwhisperer-Lite umožňuje zaznamenání maximálně 24 400 vzorků.

Pokud tedy bude šifrování trvat kratší dobu, než po kterou se vzorkuje, můžeme vzorkování zkrátit. A také, pokud šifrování bude trvat déle, lze zvýšit počet vzorků, popřípadě manipulovat s počáteční pozicí vzorkování.

1.3.3 Připojení k měřícímu zařízení

Jakmile dojde k obdržení veřejného rozhraní vzorkujícího zařízení a jeho konfiguraci, může se uživatel připojit k cílovému zařízení. Připojení je provedeno příkazem `chipwhisperer.target(scope)`. Ten vrátí instanci třídy pro komunikaci s cílovým zařízením. Parametrem příkazu je instance veřejného prostředí vzorkující desky, přes kterou se k cílovému zařízení připojuje. Komunikace s cílovým zařízením probíhá přes vzorkující zařízení přes sériovou linku rychlostí 230 400 baudů. Výrobce specifikuje komunikační protokol SimpleSerial, který integruje sériovou komunikaci a umožňuje zasílat a přijímat datové bloky.

Protokol je integrován v API rozhraní výrobce. Na cílovém zařízení tento protokol musí být integrován v programu. Jeho implementaci výrobce pro jednotlivá cílová zařízené také poskytuje.

1.3.4 Protokol SimpleSerial

S cílovým zařízením lze komunikovat pomocí veřejného rozhraní třídy cílového zařízení. Protokol po sériové lince nejdříve zašle jedno písmeno, které souží jako identifikátor typu příkazu. Identifikátor `p` určuje zaslání bloku otevřeného textu, načež po zaslání tohoto identifikátoru dojde k přenosu bloku otevřeného textu přes sériovou linku. Stejně tak funguje identifikátor `k` pro zaslání šifrového textu.

Pokud protokol zasílá pole bajtů, vnitřně dojde k jejich převedení na ASCII znaky, ty jsou zaslány přes sériovou linku a po přijetí jsou zpětně dekódovány do formátu bajtového pole. Mějme instanci této třídy target a instanci vzorkujícího zařízení scope. Ukázku zaslání otevřeného textu a klíče, spuštění vzorkování a obdržení zašifrovaného textu lze nalézt v ukázce kódu.

V ukázce nejdříve dojde v vygenerování dvojice textu a klíče. Následně je klíč zaslán protokolem cílovému zařízení. Klíč má formu pole délky 16 bajtů, toto pole je interně překonvertováno na ASCII znaky, které jsou postupně zaslány přes sériovou linku.

Po zaslání vstupního textu dojde k přípravě vzorkujícího zařízení pro vzorkování. Funkce připraví ADC převodník vzorkovacího zařízení tak, že čeká na trigger signál, který spustí měření.

Následně dojde k zaslání otevřeného textu, který má stejný formát, jako klíč. Předpokládá se, že po zaslání tvojice textu a klíče dojde k zahájení šifrování. Do proměnné `trace` se uloží poslední zaznamená stopa. Ta má formu pole čísel, hodnoty těchto čísel reprezentují spotřebu

cílového zařízení v čase. Velikost pole je dána počtem nameřených prvků stopy. Na závěr dojde k načtení zaslaného zašifrovaného bloku textu z cílového zařízení protokolem.

Všechy tyto jednotlivé kroky lze provést jedním příkazem API rozhraní `capture_trace (scope, target, text, key)`. Návratovou hodnotou příkazu bude uspořádaná čvteřice obsahující naměřenou stopu, otevřený text, klíč a zašifrovaný text.

1.3.4.1 Implementace na mikrokontroleru

Implementace protokolu pro mikrokontroler ATXmega128D4-AU je dodána formou knihoven a je implementována v programovacím jazyce C. V těchto knihovnách je důležitá zejména funknce `simpleserial_addcmd ( com, len, action )`. Zavoláním této funkce dojde k přidání nového příkazu do protokolu SimpleSerial v cílovém zařízení.

Parametr `simpleserial_addcmd com` je písmeno identifikující typ příkazu, který má být do protokolu přidán. Parametr `action` je pak odkaz na funkci, která má být zavolána, pokud cílové zařízení přijme protokolem typ příkazu daného identifikátoru. Parametr `len` je délka vstupního parametru funkce `action`.

Pro načtení otevřeného textu budou vytvořeny dvě funkce. Ty budou předány jako parametr funkci `simpleserial_addcmd` společně se specifickým identifikátorem, tím dojde k jejich přidání do registru známých příkazů protokolu. Jakmile budeme chtít zaslat text a klíč, použijeme stejné identifkátory, které jsme použili při přidání do registru známých funkcí. K zaslání zašifrovaného bloku textu poslouží funkce `simpleserial_put('r', len, ct)`.

Dalšími významnými příkazy knihoven je trojice funkcí, `trigger_init()`, `trigger_high()`, `trigger_low()`. První funkce nastaví nastaví port GPIO4 jako výstup. Druhá funkce nastaví hodnotu portu na 1, tato akce informuje o začátku šifrování a je signálem pro spuštení vzorkování pro vzorkující zařízení. Třetí funkcí dojde k nastavení portu na 0, číž informuje vzorkující zařízení o ukončení šifrování.

1.3.4.2 Implementace na FPGA

Implementace protokolu je implementována i pro zařízení Spartan 6 LX9, a to v jazyce Verilog. Program obsahuje jeden TOP modul a také `ucf` soubor pro mapování pinů. Top modul obsahuje jeden submodul nazvaný `SimpleSerial`, který implementuje komunikační protokol včetně komunikace přes sériovou linku. TOP modul definuje trojici registrů pro otevřený text, klíč a šifrový text. Také definuje řídící signály, které jsou uvedeny v tabulce 1.4. Do TOP modulu bude přidán druhý modul provádějící šifrování.

Tabulka 1.4 Vnitřní datové struktury TOP Verilog modulu

Proměnná	funkce
enc_input	registr pro předání otevřeného textu
enc_output	0 registr pro předání šifrovaného textu
enc_key	registr pro předání klíče
load_input	signál pro načtení textu, aktivní po 1 CLK
load_key	signál pro načtení klíče, aktivní po 1 CLK
enc_go	spouštěč vzorkování

Protokol načte klíč, po jeho načtení jej vystaví do registru a nastaví signál `load_key` na hodnotu 1 po dobu jednoho hodiného signálu. Tímto impulzem šifrovací jednotka načte vstupní klíč z registru. Načtení otevřeného textu funguje obdobně. Avšak jakmile protokol vystaví signál `load_pt`, v dalším hodinové cyklu vystaví signál `enc_go`, který jednotce indikuje, že má začít šifrovací algoritmus.

Jakmile šifrovací jednotka obdrží impuls k začátku šifrování, nastaví signál `enc_busy` na hodnotu 1. Tento impulz značí, že jednotka provádí šifrování. Signál je vyveden z TOP modulu

a je zapojen jako vsupní signál vzorkujícího zařízení tio4. Tento signál tedy bude spouštěčem vzorkování.

Po ukončení šifrování jednotka vystaví zašifrovaná data do výstupního registru a nastaví signál `enc_busy` na 0. Tento signál sleduje modul komunikačního protokolu a jakmile zjistí, že bylo ukončeno šifrování, zašle data z registru ven přes sériovou linku.

1.3.5 Naprogramování cílového zařízení

API rozhraní obsahuje programátory, pomocí kterých lze do cílového zařízení nahrát program. Programátory jsou implementovány pro mikrokontrolery ARM, AXMEGA a AVR. Pro nahrání programu do přípravku je nutné definovat správný programátor pro zařízení, který je parametrem příkazu pro naprogramováni. Příkaz má rozhraní
`program_target(scope, programmers.XMEGAProgrammer, path)`. Scope je vzorkující zařízení, skrz které dojde k naprogramování a path je cesta k hexadecimálnímu zkompilovanému programu.

Pro zařízení Spartan 6 rozhraní žádný programátor nenabízí. K nahrání do přípravku bude muset být využit externí JTAG programátor. Ten bude zapojen podle návodu manuálu **??** a naprogramování se provede nástrojem Xilinx Impact.

Kapitola 2
Implementace šifry

Šifru PRESENT jsem navrhoval ve dvou programovacích jazycích. V jazyce C pro mikrokontroler a v jazyce Verilog pro FPGA. Při návrhu jsem bral v úvahu fakt, že budou šifry následně integrovány s moduly komunikačního protokolu SimpleSerial a držel jsem se vstupního a výstupního formátu dat umožňující následnou integraci. Implementace šifer jsem po návrhu otestoval a integroval s komunikačním protokolem.

2.0.1 Implementace v jazyce C

V této kapitole popisuji návrh šifry pro mikrokontroler v jazyce C. Při návrhu jsem bral ohled na následnou integraci s komunikačním protokolem. Z poznatků funkce protokolu jsem věděl, že tajný klíč a vstupní text bude vhodné přijímat dvěma samostatnými funkcemi, ty budou později společně s identifikátorem přidány do registru známých příkazů protokolu. Byly vytvořeny dvě hlavní funkce. Jedna zpracovávala načtení tajného šifrového klíče. Druhá funkce zpracovala vstupní text a provedla šifrování.

2.0.1.1 Zpracování tajného klíče

Klíč bude mít podobu pole velikosti 16 bajtů, které budou přijímány v parametru funkce `getKey()`. Vytvořil jsem globální proměnnou `m_key` stejného rozměru, do které ve funkci uložím vstupní klíč. Uložení proběhne ve smyčce pro každý bajt klíče.

Po načtení tajného klíče dojde k vygenerování všech rundovních klíčů funkcí `generateKeys()`. Tyto klíče budou uloženy v globální 2D struktuře, mající rozměr 32x8 bajtů.

Rundovní klíče mají sice délku 128 bitů, nicméně k šifrování je z rundovního klíče využito pouze horních 64 bitů daného klíče. Znalost celého rundovního klíče je nutná k vygenerování následujícího rundovního klíče. Generování klíčů probíhalo ve smyčce. Plnou hodnotu rundovního klíče jsem zachovával pouze do doby vygenerování následujícího rundovního klíče, následně již nebyla znalost spodních 64 bitů klíče nutná.

Klíče jsem generoval algoritmem popsaným v kapitole 1.1.1. Prvním krokem generování bylo nahrání vstupního klíče jako nultý rundovní klíč. Jelikož je každý rundovní klíč posunut cyklicky vlevo o 61 pozic, neprišla mi reprezentace bajtového pole pro tuto operaci vhodná. Vstupní klíč jsem před začátkem generování překonvertoval do dvou proměnných délky 64 bitů. Jedna proměnná uchovávala vrchní část klíče a druhá spodní část. Generování jsem prováděl pomocí těchto proměnných. Vygenerovaný rundovní klíč jsem následně uložil do struktury překonvertováním proměnné uchovávající horní část rundovního klíče na 8 částí délky 8 bitů.

Jedním z kroků generování klíčů je nahrazení dvou čveřic bitů klíče podle pravidla substituční tabulky 1.1. Tu jsem reprentoval jako jednorozměné pole. Čtveřice bitů byla nahrazena obsahem

Výpis kódu 2.1 Rozhraní šifrovacího modulu

```
input CLK,
input ENC_START,
input LOAD_PT,
input LOAD_KEY,
input [127:0] INPUT_KEY,
input [63:0] INPUT_PT,
output BUSY,
output [127:0] OUTPUT_CT
```

pole na pozici hodnoty původní čtveřice. Mikrokontroler má nejmenší adresovatelnou jednotku 8 bitů a substituční tabulka pracuje s čtveřicemi bitů. Musel jsem tedy 4 bity získat uložením 8 bitů do proměnné a následným vynulování vrchních 4 bitů.

2.0.1.2 Šifrování

Pro načtení vstupního textu jsem vytvořil funkci `getPt`. Tato funkce také přijímá 16 bloků délky 8 bitů. To je ve výsledku 128 bitů dlouhý otevřený text, šifrování však probíhá jen nad najvyššími 64 bity vstupního textu.

Funkce spustí algoritmus šifrování dat. Zde již ponechávám vstupní text v jeho původní struktuře, která byla předána funkci jako parametr. Tuto datovou strukturu postupně při šifrování nahrazuji výstupy jednotlivých šifrových kroků. Šifrování probíhá ve smyčce, způsobem popsaným v ukázce kódu v úvodu do problematiky 1.1.1. Po ukončení šifrování je zašifrovaný blok vypsán.

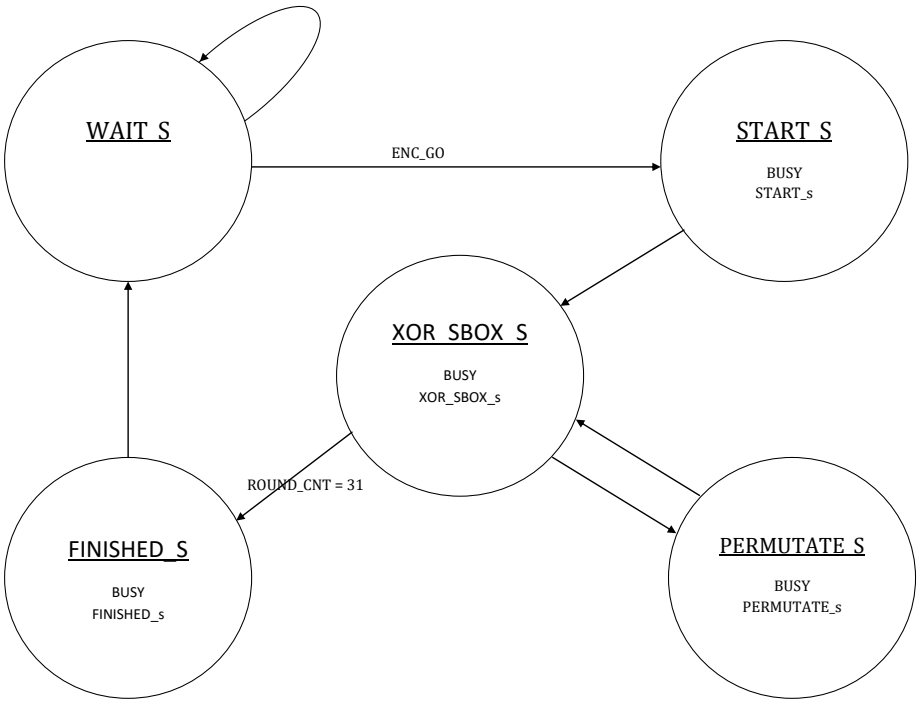
K tomuto programu jsem po otestování správné funkce šifrování integroval komunikační protokol. Modifikace, které jsem musel provést byl způsob načítání dat. Do registru příkazů komunikačního protokolu jsem přidal dva příkazy. Přikaz s identifikátorem `k` předal funkci `getKey` text přijatý protokolem a identifikátor `p` předal text funkci `getPt`. Dále jsem na začátek této funkce přidal volání funkce `trigger_high`, která na vzorkujícím zařízení spustila šifrování a po konci šifrování jsem příznak šifrování zase vypnul a zašifrovaná data jsem zaslal pomocí příkazu `simpleserial_put('r', 16, pt)` přes protokol do počítače.

2.0.2 Implementace ve Verilogu

V této kapitole je vysvětlena realizace šifry PRESENT v jazyce Verilog. Tento jazyk jsem zvolil, protože komunikační protokol SimpleSerial je také implementován v tomto jazyce. Díky tomuto protokolu má šifrovací jednotka na vstupu připravená vstupní data ve vyhovující formě. Šifru jsem pro desku Spartan 6 realizoval ve vývojovém prostredí ISE 14.7. Tomuto prostředí již skončila technická podpora a nemám dobré zkušenosti s hybridním návrhem, proto jsem se rozhodnul také implementovat jednotku v jazyce Verilog, a ne v jazyce VHDL.

Rozhraní jednotky je znázorněno v ukázce kódu. Prvním vstupem jednotky je hodinový signál `CLK`. Ten je generován deskou UFO Board s frekvencí 7,37 MHz. K implementaci šifry jsem zvolil synchronní návrh. To znamená, že ke změnám všech impulzů uvnitř jednotky dochází při změně hodnoty hodinového signálu, v tomto případě se jedná o nábežnou hranu hodin.

Dále máme na vstupu 64 bitový vstupní text. Tento text budeme pomocí jednotky šifrovat. Pokud je signál `LOAD_PT` nastaven na hodnotu 1, znamená to, že hodnota vstupního textu je platná. Tento signál bude nastaven na hodnotu 1 pouze po dobu jednoho hodinového cyklu. Signál je generován komunikačním protokolem SimpleSerial a je vystaven vždy při obdržení nového vstupního textu sériovou komunikací. Protokol SimpleSerial přijímá vstupní text fixní délky 127 bitu. Šifra PRESENT pracuje s textem délky 64 bitů. Pokud bych protokolem nutně chtěl přijímat text délky 64 bitů, musel bych upravit jeho implementaci ve Verilogu.



Obrázek 2.1 Konečný automat řídící jednotky

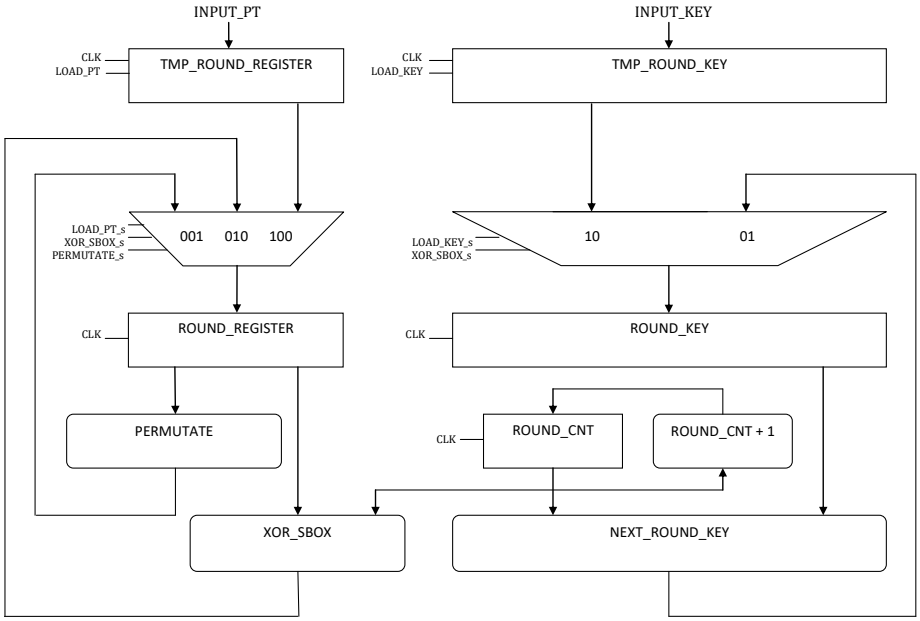
Podobně funguje načtení vstupního klíče. Klíč mé implementace šifry PRESENT pracuje s délkou klíče 128 bitů, což v tomto případě odpovídá délce klíče přijímaného protokolem. O platnosti vstupního klíče je jednotka opět informována signálem `LOAD_KEY`, který bude při platnosti hodnoty klíče nastaven na hodnotu 1 po dobu jednoho hodinové signálu.

Jakmile jednotka dostane příkaz k zahájení šifrování `ENC_START` nastavený na hodnotu 1 po dobu jednoho hodinového cyklu, nastaví signál `BUSY` na hodnotu 1. Signál je připojen na pin P95, který je namapován jako logický vstup desky Chipwhisperer Lite a slouží jako impulz pro spuštení vzorkování. Po provedení šifrovacího algoritmu je zašifrovaný text vystaven na výstupní signál `OUTPUT_CT` a signál `BUSY` je nastaven na hodnotu 0. Shozením signálu BUSY protokol SimpleSerial zašle výstupní zašifrovaný text zpět po sériové lince.

Implementaci šifry jsem se snažil rozdělit na dvě části – Řídící a datovou jednotku. Řídící jednotka na základě vnitřních stavů generuje řídící signály. Podle hodnot těchto signálů datová jednotka zapisuje příslušná data do datových struktur. Top modul šifry tedy obsahuje pouze instanci modulu řídící jednotky a instanci modulu datové jednotky, společně s deklaracemi signálů, které tyto dvě jednotky propojují.

2.0.2.1 Řídící jednotka

Řídící jednotka, které se také říká řadič, je realizována konečným automatem typu Moore. Automat je znázorněn na obrázku. Stavy a přechody tohoto automatu jsou znázorneny na obrázku. Vnitřní stav řadice je uložen ve stavovém registru. Hodnota tohoto registru je při inicializaci zařízení nastavena na vnitřní stav `WAIT_S`, ve kterém setrvá, dokud nepřijde komunikační proto-



Obrázek 2.2 Datová cesta

kolem vygenerovaný signál `ENC_START_S`. Jednotka se skládá ze tří částí.

První jednotkou je čistě kombinační proces. Ten sleduje aktuální vnitřní stav a vstupní signály a na základě jejich hodnot rozhodne o hodnotě vnitřního stavu následujícího. Aby byl tento proces syntetizátorem vysyntetizován jako kombinační, musí být v jednotce pro každou možnou hodnotu vnitřního stavu a vstupních signálu definována konkrétní akce. Pokud návrhář zamýšlí, aby řadič setrval v aktuálním stavu, musí explicitně nastavit hodnotu následujícího stavu na hodnotu stavu aktuálního. Tento kombinační proces ovšem samotný přechod nerealizuje. Pouze říká, který má následující stav být.

O přechod do následujícího stavu se stará druhá část jednotky. Tou je čistě sekvenční proces, který při každé nábežné hraně hodinového signálu přejde do nového stavu. Proces je interně realizován jako flip-flop a v procesu se pouze jeho hodnota nahradí hodnotou vygenerovanou dříve zmíněným kombinačním procesem. Přechod do dalšího stavu nastane při každé náběžné hraně hodin. Pokud bude kombinační jednotkou rozhodnuto, že má řadič setrvat v aktuálním stavu, do stavového registru se sekvenční jednotkou zapíše stejná hodnota.

Třetí částí jednotky je výstupní proces. Ten sleduje současný stav průběžně aktualizovaný sekvenční jednotkou. Na základě stavu nastavuje příslušné řídící signály. Tyto signály jsou předávány datové jednotce a ovládají zápis do datových struktur.

2.0.2.2 Datová jednotka

Datová jednotka se stará o zápis do registrů. Zápis do příslušného registru se provádí na základě vstupních signálů generovaných řadičem. Piny vládající jednotlivé piny lze nalézt v obrázku.

Jedná se o synchronní návrh, tedy pokud je nastaven signál k zápisu do daného registru, k zápisu vždy dojde při náběžné hraně hodinového signálu. Jedním ze vstupních signálů jednotky proto bude hodinový signál. Dalšími vstupy jsou řídící signály generované řadičem, otevřený text a klíč předaný protokolem SimpleSerial a také protokolem předané signály `LOAD_PT` a `LOAD_KEY`.

Výstupem jednotky je rundovní čítač a jeho hodnotu sleduje řadič kvůli ukončení šifrování. Inkrementaci rundovního čítače provádí datová jednotka. Dalším výstupem je zašifrovaný text, který je vystaven ven z celého šifrovacího modulu. Základním kamenem jednotky je rundovní registr uchovávající průběžně šifrovaná data. Také v jednotce uchováváme čítač, který je zpočátku nastaven na hodnotu 0.

Návrh datové jednotky jsem rozdělil na několik modulů. Modul `KEYGEN` zajišťuje generování rundovních klíčů. Algoritmus generování je vysvětlen v popisu šifry 1.1.1. Vygenerování dalšího rundovního klíče je spuštěno signálem řadiče, který je vstupním signálem modulu společně s čítačem, jehož znalost je nutná pro vygenerování dalšího rundovního klíče. Pro vygenerování musíme znát všech 128 bitů aktuálního rundovního klíče, k šifrování se používá pouze horních 64 bitů. Proto je celý rundovní klíč interně uložen pouze v modulu `KEYGEN` a na výstup je vyvedeno pouze horních 64 bitů klíče.

Dále jsou v jednotce implementovány dva další moduly. Jeden modul zajišťuje vrstvy exkluzivní OR nad rundovním registrem a rundovním klíčem společně se substituční vrstvou. Tuto jednotku jsem si pracovně nazval `XBOX`. Druhý modul implementuje permutační vrstvu. Obě jednotky jsou pouze jednoduché kombinační obvody, k jejich provedení a vystavení hodnot na dráty dojde vždy při změně vstupních dat. O zápis těchto dat se starají až řadičem generované signály.

2.0.2.3 Souhra jednotek

V ideálním případě jsou všechny signály pro zápis do datových struktur generovány řídící jednotkou. K takovému návrhu jsem prvotně přistoupil. Z implementace komunikačního protokolu víme, že platnost vstupního textu je dána vstupním signálem `LOAD_PT`, který je nastaven na hodnotu 1 po dobu jednoho hodinového cyklu. Stejně tak je dána platnost hodnoty klíče pomocí signálu `LOAD_KEY`. Řadič měl původně o dva stavy více. Jeden stav pro načtení otevřeného textu a druhý text pro načtení klíče. Jakmile byl z protokolu obdržen signál pro načtení, řadič se přesunul do příslušného stavu a vystavil vnitřní řídící signál pro načtení. Ten přijímala datová jednotka a uložila vstup do příslušného registru. Protokol ve svém konečném automatu neřeší pořadí přijetí vstupního textu a klíče. Takto jsem konečný automat implementoval také. Má implementace byla schopna přijímat vstupní text a klíč bez ohledu na pořadí. Během šifrování načtení nebylo možné.

Tato implementace ovšem negenerovala správný šifrový text. Behaviorální simulace testující funkčnost šifry ovšem očekávané šifrové texty generovala, zaměřil jsem se tedy na přijímání vstupních dat z komunikačního protokolu. Detailní funkce komunikačního protokolu není rozsáhle popsána a jeho implementace ve Verilogu mi přijde chaotická. Postupně jsem tedy upravoval způsoby a možnosti načítání dat. Úpravy jsem testoval zasíláním dat komunikačním protokolem a porovnáváním obdržených šifrových textů v počítači oproti očekávaným.

Nejdříve jsem umožnil načtení nových vstupních dat při šifrování, impuls k načtení data načetl a zastavil probíhající šifru. Tato úprava opět negenerovala správný šifrový text, načež jsem z automatu odstranil vnitřní stavy pro načtení vstupů. Signály pro načtení jsem přivedl rovnou do datové jednotky a ta uložila vstup do registrů. Odstranil jsem tím zbytečný mezikrok, který dle mé úvahy mohl vést ke zpožděním, které behaviorální simulace neodhalí. Nicméně ani tato úprava nevyústila ve správnou funkčnost.

Následně jsem při zasílání dat protokolem objevil, že při zaslání stejných vstupních dat dvakrát za sebou generuji různé šifrové texty. Tento poznatek mě navedl k úvaze, že při stejném klíči z minulého šifrování dojde pouze k zaslání vstupního textu a následnému vygenerování příkazu pro spuštění šifrování. To by vysvětlovalo důvod generování různých textů při opakovaném zasílání stejných dat, jelikož v návrhu byl registr uchovávající klíč při šifrování modifi-

kován.

Koneckonců, příkaz API rozhraní `capture_trace()` uvádí parametr zaslání klíče jako volitelný parametr. Šifru jsem tedy upravil tak, aby uchovávala poslední načtenou hodnotu klíče a vstupního textu. Přidal jsem dva záchytné registry, do který jsem vždy při vystavení signálů pro načtení uložil vstupní data a dále s registry nemanipuloval. Tato úprava již generovala očekávané výstupní texty. Šifra tedy mohla být navrhnuta tak, aby načítala vstupy přechodem do příslušného stavu automatu a vygenerováním vnitřních řídících signálů pro datovou jednotku.

2.0.3 Porovnání návrhů

V implementaci šifry v jazyce C dojde při načtení tajného klíče k vygenerování rundovních klíčů. Nasledně je při obdržení bloku otevřeného textu vydán impulz k spuštění vzorkování a proběhne šifrovací algoritmus.

Naopak v implementaci v jazyce Verilog jednotka dostane příkaz ke spuštění, tím nastaví signál pro zahájení vzorkování a jednotka začne šifrovat blok textu. Ke generování rundovních klíčů dochází postupně v průběhu rundy.

2.0.4 Oveření návrhu

Implementaci šifry jsem v průběhu návrhu testoval. Ověření návrhu jsem provedl při navžených implementacích bez integrace s komunikačním protokolem SimpleSerial. Jakmile jsem se ujistil, že je návrh algoritmu správný, integroval jsem jednotku do protokolu a otestoval správnou integraci jednotky do protokolu.

2.0.4.1 Ověření návrhu v programu C

Testování šifry jsem nejdříve provedl pro program implementovaný v jazyce C. Naprogramoval jsem dešifrovací jednotku. Děšifrování bloku probíhá provedením šifrovacích kroků v reverzním pořadí. Neprovádí se reverzně jen kroky šifrování, ale i funkce jednotlivých kroků. Na začátku dešifrovacího algoritmu jsem vygeneroval rundovní klíče. Při dešifrování nelze klíče generovat za běhu, jelikož v prvním kroku dešifrování potřebujeme znát hodnotu posledního rundovního klíče.

Dešifrování také probíhá v rundách. První operací je XOR s rundovním klíčem. Zde se používají klíče v opačném pořadí, tedy v prvním kroku se provede XOR s posledním rundovním klíčem. Dále je blok permutován podle inverzní permutační tabulky. Při šifrování byly bity podle tabulky1.2 z pozic i přesunuty na pozice $P(i)$. Při dešifrování budou bitz přesunuty z pozic $P(i)$ na pozice i.

Dalším krokem rundy je inverzní substituční vrstva. Blok je rozdělen na čtveřice a jejich hodnoty jsou nahrazeny hodnotou inverzní sustituční tabulky. Inverzní substituční tabulka má podobu substituční tabulky 1.1, nahrazování probíhá opačným způsobem.

Tuto dešifrovací jednotku jsem navrhoval nezávisle na návrhu šifrovací jednotky. Testování šifrovací jednotky jsem provedl zašifrováním blokových textů pomocí tajného klíče. Výstupy jsem dešifroval pomocí stejného klíče a porovnal, že dešifrovaný blok textu se shoduje v textem původním. Tyto testy probíhaly kompilací programů v počítači kompilátorem `gcc`.

Po ověření implementace jsem šifru integroval to komunikačního protokolu. Následně jsem program nahrál do přípravku ATXmega128D4-AU a přes komunikační protokol zasílal dvojice textu a klíče, u kterých jsem ze softwarové verze programu znal tvar zašifrovaného textu. Tento text jsem následně porovnal s blokem textu zaslaným komunikačním protokolem y jednotky.

2.0.4.2 FPGA

Návrh modulu šifry jsem testoval behaviorální simulací. Opět jsem jako vstupy použil data ze sofwarové verze šifrovací a dešifrovací jednotky. Jakmile jsem měl správně navrženou jednotku,

integroval jsem ji do komunikačního protokolu a následně testoval, že zaslání vstupního bloku textu a klíče vrátí očekávaný zašifrovaný text.

Při návrhu programu jsem se setkal s několika problémy. Jeden problém je popsán přímo v kapitole návrhu 2.0.2. U této chyby návrhu si stále nejsem jist, zda jsem opravdu neporozuměl implementaci komunikačního protokolu nebo nevědomky odstranil jinou chybu. Každopádně si myslím, že protokol není vůbec přehledně naimplementován a při syntéze hlásí spoustu warningů, které jsou celkem matoucí při ladění programu.

Kapitola 3
Útok CPA

V této kapitole je popsán postup provedení korelační odběrové analýzy na implementované šifry. Byl vytvořen Hammingův model spotřeby na základě znalosti implementace těchto šifer, model byl následně použit k útoku a získání tajného šifrovacího klíče.

Pro vytvoření Hammingova modelu potřebujeme identifikovat část algoritmu šifry, která provádí operaci nad šifrovaným blokem textu pomocí tajného klíče. A také, musíme znát buď hodnotu šifrovaného bloku na vstupu této operace, nebo jeho hodnotu na výstupu.

Předpokládejme, že máme k dispozici soubor záznámů spotřeby energie při povádění šifrovacího algoritmu. Dále, ke každé této stopě známe vstupní blok textu.

Postupně budeme procházet všechny možné hodnoty klíče. Pro každou z těchto možných hodnot provedeme nad vstupním blokem a daným odhadem klíče část algoritmu, na kterou útočíme a tím obdržíme výsledek operace. Pokud byl odhad klíče správný, mezi Hammingovou vzdáleností těchto slov s spotřebou energie bude existovat vztah.

Když s jednou hodnotou odhadu klíče provedeme operaci nad více vzorky, můžeme následně sledovat provázanost spotřeb energie a Hammingových vzdáleností. K vyjádření provázanosti slouží Pearsonův korelační koeficient, pomocí něhož určíme míru provázanosti.

Tento koeficient vyjádříme pro každou možnou hodnotu klíče a jako správný klíč identifikujeme ten s největší korelací mezi reálnou a vymodelovanou spotřebou.

U tohoto postupu nastává jeden problém. Již bylo zmíněno, že výpočet všech možných hodnot klíče délky 128 bitů je výpočetně velmi náročná operace. Proto by bylo vhodné zaútočit na část šifry provádějící operaci jen s částí klíče. Tím se sníží počet možných hodnot klíče na přípustnou hodnotu.

U šifry PRESENT je vhodným bodem pro útok substituční vrstva. Ta přijímá pouze 4 bity, které nahradí dle substituční tabulky 1.1. Útok závisí na implementaci algoritmu a způsobu zápisu do datových struktur.

3.1 Mikrokontroler

V kapitole popisuji útok na implementaci šifry, kterou jsem naprogramoval pro mikrokontroler. V nulté rundě bude mít rundovní klíč podobu tajného klíče v nezměněné podobě. Nad nejvyššími 64 bity klíče a vstupním blokem textu se provede operace exkluzivní OR. Následně se výsledek této operace rozdělí na čtveřice bitů, které budou substituční tabulkou nahrahrazeny novými hodnotami.

Na mikroprocesoru neútočím na změnu hodnoty v datové struktuře, ale na datové sběrnice. Ty jsou přednabité vždy na nulu nebo jedničku a při přístupu k proměnné se výstup objeví na datové sběrnici.

Výpis kódu 3.1 Korelační odběrová analýza pro jeden subklíč

```
for odhad_subklice in 0, 16
        for stopa in vsechny_stopy
            hypotezy[stopa] = ham_vaha(SBOX(pt(stopa)(0)^odhad_subklice))
    stredni_hodnota_hypotezy = prumer ( hypotezy )
    stredni_hodnota_stop = prumer ( vsechny_stopy, osa 0 )
    for i in pocet_stop
        odchylka_hypotezy = hypotezy[i] - stredni_hodnota_hypotezy
        odchylka_stop = vsechny_stopy - stredni_hodnota_stop

        kovariance += (odchylka_hypotezy + odchylka_stop)
        rozptyl_hypotezy += odchylka_hypotezy * odchylka_hypotezy
        rozptyl_stop += odchylka_stop * odchylka_stop
    CPA = kovariance / sqrt ( rozptyl_hypotezy + rozptyl_stop )
    maxCPA = max(abs(CPA))
subklic = max ( maxCPA )
```

Jelikož nedochází k nahrazení původní hodnoty novou hodnotou, nevytvořil jsem pro útok model Hammingových vzáleností, ale model Hammigových vah. Útočím na nultou rundu šifrovacího algoritmu, zde budou jako vstup pro substituční tabulku 4 bity otevřeného bloku textu XORované s 4 bity tajného klíče. Výstup substituce bude v daném modelu hypotéza.

Budeme postupně útočit na subklíče, to jsou 4 bitové části tajného klíče. Pro jednoduchost vysvětlení nastíním útok na nultý subklíč délky 4 bitů. Útok přibližuje následující úkázka kódu:

Máme k dispozici soubor naměřených stop a k nim odpovídající otevřené texty. Postupně projdeme všechny možné hodnoty tohoto subklíče a vždy při dané hodnotě subklíče přiřadíme všem stopám očekávanou Hammingovu váhu výsledku operace nad subtextem a subklíčem. Subtext je v tomto případě část vstupního textu délky 4 bitů na stejných pozicích, jako pozice bitů subklíče v klíči.

Následně pro danou hodnotu subklíče mezi tímto modelem a stopami vyjádřím Pearsonův korelační koeficient. Zde je vhodno zmínit, že tyto dvě porovnávané veličiny jsou různých rozměrů. Model Hammingovy váhy pro stopu je vyjádřen jedním číslem, kdežto stopa může obsahovat stovky až tisíce vzorků. Ve výpočtu stále počítáme hodnoty pro všechny body stop. Naštěstí Python umožňuje elegantně takové aritmetické provádět. Až v předposledním kroku výpočtu vyjádříme korelace pro možné hodnoty subklíčů a následně vybereme hodnotu sabklíče s největší korelací.

Pro útok na všechny suklíče se algoritmu bude provádět ve smyčce. Tajný klíč má délku 128 bitů, v nultém kroku se využije pouze nejvyšších 64 bitů. Pomocí tohoto modelu jsme schpni získat jen tuto polovinu klíče využitého jako nultý rundovní klíč.

Smyčka tedy poběží 16 krát. Otevřený text je uložen v bajtech, v útoku útočím na čtveřice bitů. Pokud budeme útočit na nultý klíč, budeme vytvářet model s vrchními 4 bity nultého bloku textu. Pokud budeme útočit na první subklíč, model vytvoříme s 4 spodními bity nultého textu. Pro výpočet všech subklíčů jsem model upravil.

Šifrování na mikrokontroleru trvá velmi dlouho, po provedení algoritmu je proměnná API rozhraní trig_count nastavena na hodnotu 658 248. Cílovému zařízení tedy trvalo zašifrování přibližně 165 000 hodinových cyklů. Vzorkující deska Chipwhisperer-Lite umožňuje záznam maximálne 24 400 prvků.

Operace, podle které jsem vytvořil Hammingův model se provede na počátku šifry. Proměnnou rozhraní samples jsem nastavil na maximální možnou hodnotu 24 400 a pokud by se operace provedla později, přidal bych offset, aby se začalo vzorkovat později.

Nicméně tato operace proběhla v době záznamu spotřeby, jelikož jsem šifru prolomil bez potřeby zpožďovat start vzorkování. Nicméně takto neproběhl útok nad celou šifrou, ale pouze nad její částí. Pro útok nad celým záznamem šifry bych potřeboval přípravek Chipwhisperer

Výpis kódu 3.2 Korelační odběrová analýza pro všechy subklíče

```
subklice[16]
for pozice_subklice in 0, 16
 for odhad_subklice
  for stopa in vsechny_stopy
   if (pozice_subklice % 2) == 0
    hypotezy[stopa] = ham_vaha(SBOX(pt(stopa/2)(0)>>4^odhad_subklice))
   else
    hypotezy[stopa] = ham_vaha(SBOX(pt(stopa/2)(0)&0x0f^odhad_subklice))
   stredni_hodnota_hypotezy = prumer ( hypotezy )
        .
        .
        .
   subklice[pozice_subklice] = argmax ( maxCPA )
```

1200, který umožňuje naměřit až 960 000 vzorků.

Šifru jsem se zpočátku snažil prolomit nad 5 000 stopami. Byl jsem úspěšný. Postupným ubíráním stop jsem zjistil, že k získání klíče potřebuji přibližně 50 stop.

3.2 FPGA

Pro provedení korelační odběrové analýzy FPGA jsem využil jiný Hammingův model, než na mikrokontroleru. Zde útočím na změnu hodnot v registrech. Na začátku každého šifrování dojde k zápisu vstupního textu do záchytného registru. Při šifrování jsou nad hodnotou tohoto registru prováděny šifrové operace a jeho hodnota je průběžně aktualizovaná.

V prvním kroku šifrování bude registr aktualizován jednotkou XBOX. Tato jednotka je popsána v návrhu šifry 2.0.2.2. Vím tedy, že v záchytném registru bude na začátku šifrování uložen vstupní text, nad tímto textem se provedou operace XOR s rundovním klíčem, výsledek této operace bude rozdělen na bitové čtveřice, jejichž hodnota bude nahrazena posle substitoční tabulky. Výsledek těchto operací pak bude nahrán do záchytného registru.

Nemohu v tomto případě zvolit model Hammingových vah. V registru jsou nahraná nenulová data, avšak znám hodnotu těchto dat. Zvolil jsem v tomto případě model Hammingových vzdáleností. Podel spotřeby budu modelovat podle vzálenosti původní a nové hodnoty záchytného registru. Původní hodnota registru bude vstupní text a nová hodnota registru bude výsledek operace SBOX(text$\hat{}$keyguess).

Šifra je ve Verilogu navržena tak, že každé dva hodinové cykly provede jednu rundu. V případě FPGA bude pořet hodinových cyklů nutných k provedení šifry velmi malý. Po zašifrování textového bloku a provední měření nad cílovým přípravkem FPGA vypisovalo rozhraní hodnotu proměnné trg_count 260. Jelikož hodinový signál vzorkujícího zařízení běží čtyřikrát rycheleji, než hodinový signál cílového zařízení a vzorkující zařízení provede zaznamenání spotřeby při každé náběžné hraně hodinového cyklu, ja toto číslo naprosto předpokládaná hodnota.

Pro další meření jsem tedy věděl, že bude vhodné nastavit proměnnou API rozhraní samples na hodnotu 260, jelikož pouze po tuto dobu skutečné probíhalo šifrování.

Z poznatků provedené korelační odběrové analýzy na mikrokotroleru jsem zkusil prolomit šifru nad 100 stopami, zde jsem byl neúspěšný a postupně jsem přidával počet stop. Zjistil jsem, že k úspěšnému prolomení a získání celého klíče potřebuji přibližně 2 500 stop.

Kapitola 4
SNR

Po uspěšném provedení korelační odběrové analýzy a získání klíče na mikrokontroleru i FPGA jsem vyjádřil poměr signálu a šumu. K tomu jsem využil Hammingův model vymodelovaný při útoku CPA.

4.0.1 Třídění stop

Pro výpočet SNR jsem stopy rozdělil do skupin podle Hammingových vah vstupního textu a strávného klíče. Pro šifru PRESENT budeme mít 5 tříd Hammingových vah, váha čtveřice bitů může být nulová až maximální váhy 4. Jakmile budeme mít stopy rozděleny do skupin, identifikujeme třídu s největším počtem stop.

Signál vypočítáme jako rozptyl středních hodnot všech tříd, kromě tříd neobsahující žádnou stopu. Za signál v tomto případě považujeme únik dat formulovaný Hammingovým modelem. Vypočítáme jej jako rozptyl středních hodnot všech tříd. Šum bude rozptyl třídy s největím počtem prvků.

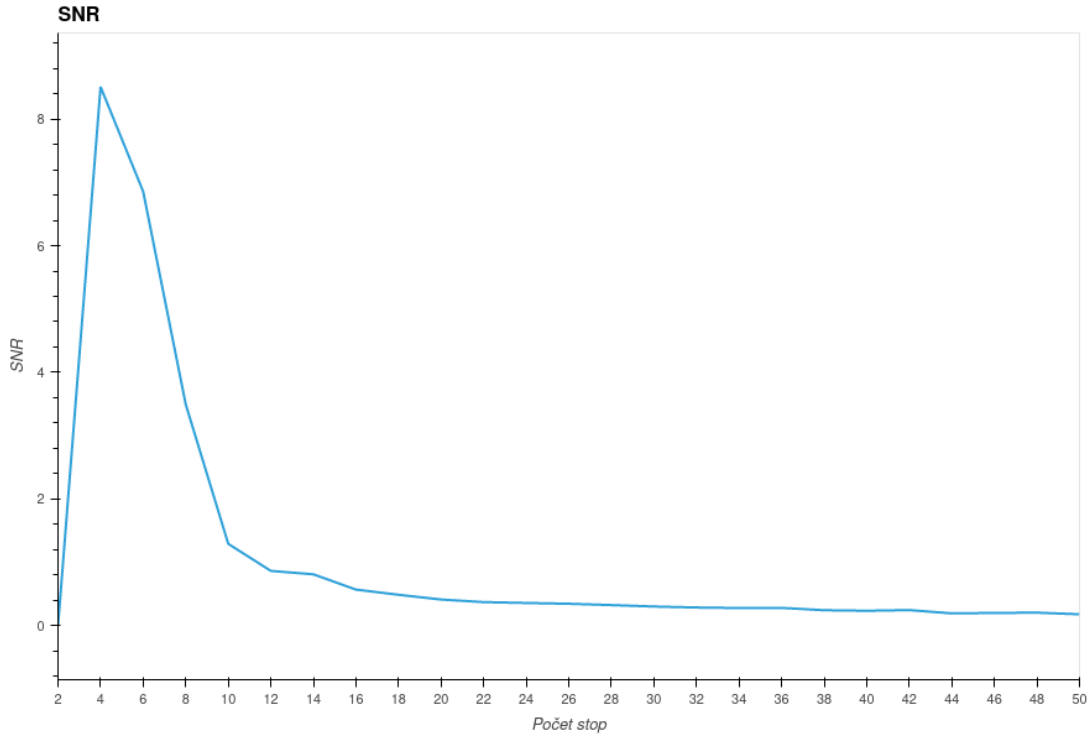
Vyjádřením SNR touto metodou jsem pozoroval pochybné výsledky. Graf ukazuje trend poměru signálu a šumu vůči narůstajícímu počtu stop.

Postupnou analýzou jsem objevil, že se v rozptylu třídy s největší korelací objevují nulové hodnoty. Tato hodnota je při výpočtu SNR použita jako dělitel. Pokusil jsem se dělění těmito nulovými hodnotami při výpočtu SNR vynechat. Po této úpravě jsem dostával stále podobé výsledky.

Následně jsem nerozděloval stopy to pěti tříd podle čtveřic bitů, ale vytvořil jsem Hammingův model pro 8 bitů a rozděloval stopy do devíti tříd. Ani tato úprava neeliminovala trend rapidního klesání SNR k nule.

4.0.2 Standard Score

Výpočet SNR z minulé kapitoly jsem identifikoval jako nevhodný pro mé použití a vypočítal jsem SNR metodou Standard Score. Signál vypočítáme jako hodnota korelace pro správný klíč mínus průměrná hodnota korelace pro správný klíč. Šum je odmocnina rozptylu korelace všech klíčů. Toto měření provedeme opakovaně a vyjádříme střední hodnotu, která bude reprezentovat SNR.



Obrázek 4.1 Chybný model výpočtu SNR

Kapitola 5
Měření

Jakmile jsem nad mikrokontrolerem i FPGA byl schopen úspěšně provést korelační odběrovou analýzu a vytvořil algoritmus pro výpočet poměru signálu a šumu, přešel jsem k měření dat v různých podmínkách.

Již jsem měl hrubou představu, jak budu následně naměřená data interpretovat a věděl jsem, že bude potřeba naměřit velké množství dat. Pro mikrokontroler jsem vždy v daném prostředí zaznamenal 3 000 stop. Naměření takového množství stop trvalo v řádu minut. Pro FPGA jsem původně 65 000 stop, toto měření trvalo přibližně jednu hodinu.

5.0.1 Měření v běžných podmínkách

Nejdříve jsem provedl měření v běžných podmínkách provozu zařízení. Cílové i vzorkující zařízení jsem měl položeno na pracovním stole vedle počítače a mobilního telefonu, obě tyto zařízení byly připojeny k bezdrátové WiFi síti komunikující s frekvencí 2,4 GHz. Ke komunikaci zařízení s WiFi sítí docházelo sporadicky, toto měření odpovídalo bežnému provozu.

5.0.2 Odstínění od vnějších vlivů

Druhým typem měření, které jsem provedl bylo měření s maximálním odstíněním přípravků od okolních jevů. Pro měření jsem navštívil Petřínské sady, ve kterých jsem našel vhodné místo bez signálu WiFi sítě, což jsem si ověřil mobilní aplikací zobrazující dostupnost okolních sítí. Započal jsem měřením na přípravku Spartan 6. Tento přípravek jsem důkladně obalil alobalem, abych zařízení odstínil od vnějších vlivů. Mobilní telefon i počítač byly po dobu měření nastaveny v režimu Letadlo.

Přibližně v polovině měření jsem zaznamenal zvedající se vítr a stahující se mračna. Naštěstí jsem neprováděl 110 000 měření v jednom cyklu, ale prováděl jsem 10 000 měření v 11 cyklech. Toto opatření jsem zvolil právě pokud bych byl nucen z nenadálých okolností rušení přerušit. Nejsem si jistý, zda by při přerušení měření v průběhu zachytávací smyčky nedošlo ke korupci dat. Nechal jsem doběhnout měřící cyklus a Petřínské sady jsem opustil, abych uchránil přípravky před potenciálním děštěm.

Druhou část měření odstíněného přípravku jsem provedl v domácích podmínkách. Vytvořil jsem provizorní Faradayovu klec, což byla krabice důkladně vystlaná alobalem a také důkladně obalená. Do této krabice jsem vložil cílové zařízení a také vzorkující zařízení. Měření v Petřínských sadech probíhalo za odstínění cílového zařízení i od záznamového zařízení. V případě provizorní faradaovy klece jsem musel oba přípravky vložit dovnitř a cílové zařízení nebylo odstíněno od vzorkujícího zařízení.

Výpis kódu 5.1 Čítač v jazyce Verilog

```
module COUNTERS (
        input      CLK,
        output reg OUTPUT_BIT
);
    reg [15:0] COUNTER = 16'h000;
    always @( posedge CLK ) begin
        COUNTER <= COUNTER + 1;
        OUTPUT_BIT <= COUNTER[15];

    end
endmodule
```

S touto krabicí jsem se přesunul na toalety Bloku 6 Strahovských kolejí, do nultého patra na severní straně. Z dřívějších zkušeností vím, že poslední kabinka je odstíněna od veškerého WiFi signálu. To jsem si opět ověřil mobilní aplikací zobrazující dostupnost okolních sítí. Počítač i mobilní telefon byly opět po dobu měření nastaveny v režimu Letadlo. Měření mikrokontroleru proběhlo obdobným způsobem.

5.0.3 Rušení signálu

Pro generování maximálního rušení zařízení jsem se přesunul do kuchyně Strahovských kolejí, opět nultého patra severní strany. Zde jsem přípravek položil na mikrovlnnou troubu. Ta byla po dobu měření zapnuta na maximální výkon, přesnou hodnotu výkonu trouby jsem nenašel na trobě samotné ani na stránkách výrobce. Při měření byl také na přípravku položen mobilní telefon, který přijímal data přes mobilní 4G síť rychlostí 20 Mbitů/s. Počítač byl umístěn vedle mikrovlnné trouby a také přijímal data, rychlost přijmu dat na počítači jsem si nezaznamenal.

5.0.4 Vnitřní šum

Na FPGA jsem ve Verilogu vytvořil modul 16 bitového čítače. Vstupem toho modulu je zdroj hodinového signálu a výstupem modulu je nejvyšší bit čítače.

V TOP modulu šifry PRESENT jsem tento modul inicializoval 15 krát. Také jsem nad všemi jednobitovými výstupy čítačů provedl operaci exkluzivní OR a výsledek této operace jsem vyvedl na výstupní pin. Tento krok jsem provedl, abych oklamal syntetizátor a on ponechal tyto čítače v návrhu.

Touto verzí programu jsem provedl měření v běžných podmínkách. Dále jsem počet těchto čítačů navýšil na hodnotu 69. Rád bych zmínil, že toto číslo jsem zvolil, protože je to rok narození mé maminky. Verze s čítače byzy testovány v běznych podmínkách.

V mikrokontroleru jsem spouštěl vnitřní čítače pomocí vnitřních portů.

Kapitola 6
Vyhodnocení výsledků

Z naměřených dat jsem vyjádřil hodnotu odstupu signálu od šumu a úspěšnost korelační odběrové analýzy. Odstup signálu od šumu jsem vypočítal metodou Standard Score. Uspěšnost korelační odběrové analýzy jsem vyjádřil v procentuální úspěšnosti. Tyto dvě veličiny jsou vykresleny v separátních grafech a zvlášť pro mikrokontroler a FPGA. V každém z těchto grafů je vykreslen průběh veličiny při normálním běhu zařízení, vnějším a vnitřním rušení a odstínění přípravku.

Na mikrokontroleru jsem tyto hodnoty určil pro 2 až 50 stop s přírůskem 2. Aby byly určené hodnoty validní, tuto hodnotu jsem pro daný počet stop vypočítal 30 krát nad různým souborem stop. Následně jsem vyjádřil střední hodnotu veličiny.

Na FPGA jsem hodnoty spočítal obdobným způsobem. Zde jsem hodnoty vyjadřoval pro rozsah 250 až 2500 s přírůstkem 250. Také jsem výpočet opakoval 30 krát a následně vyjádřil střední hodnotu.

6.0.1 Analýza pro FPGA

Z grafů mužeme pozorovat, že došlo k ovlivnění hodnot proměnných představením rušivých elementů. Červená křivka reprezentuje hodnoty veličin při běhu zařízení v normálním prostření.

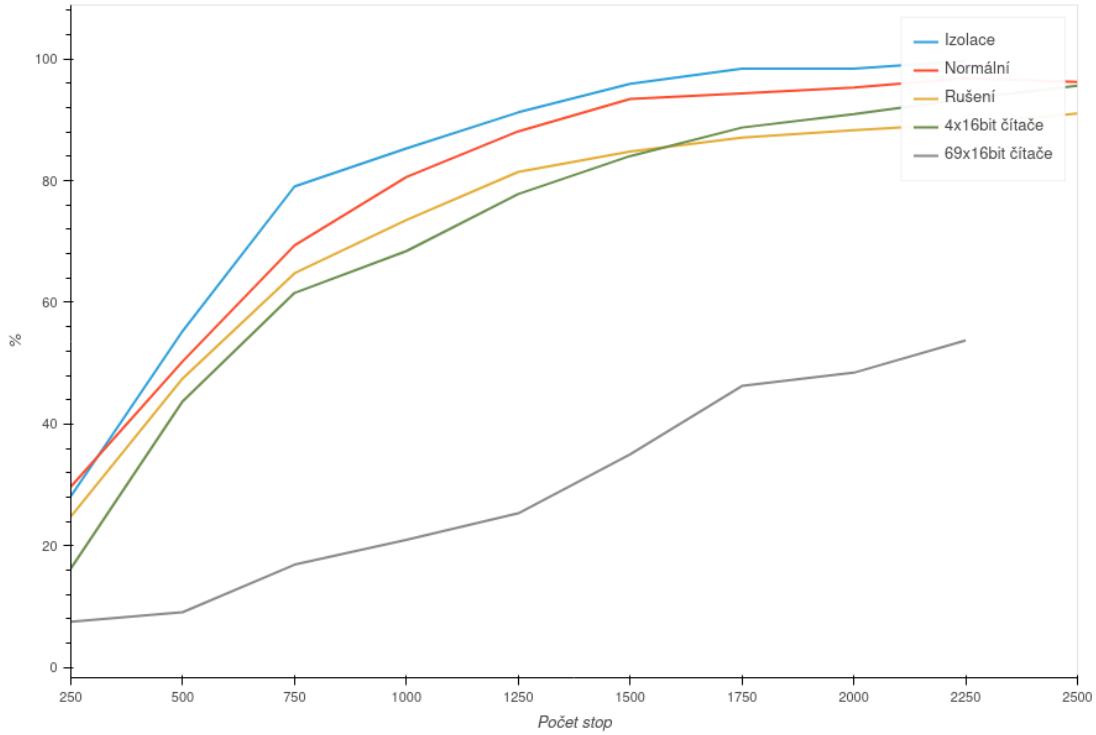
Modrá křivka reprezentuje běh zařízení odstíněného od okolních vlivů. Odstínění zařízení od možných rušivých elementů zvýšilo sílu signálu a došlo k navýšení úspěšnosti korelační odběrové analýzy.

Na žluté křivce můžeme pozorovat, že se vnějšími vlivy povedlo snížit signál a úspěšnost korelační odběrové analýzy. Představení vnějších rušivých elementů dokonce od určitého počtu vzorků generovalo větší šum, než 4 16 bitové volně běžící čítače uvnitř zařízení.

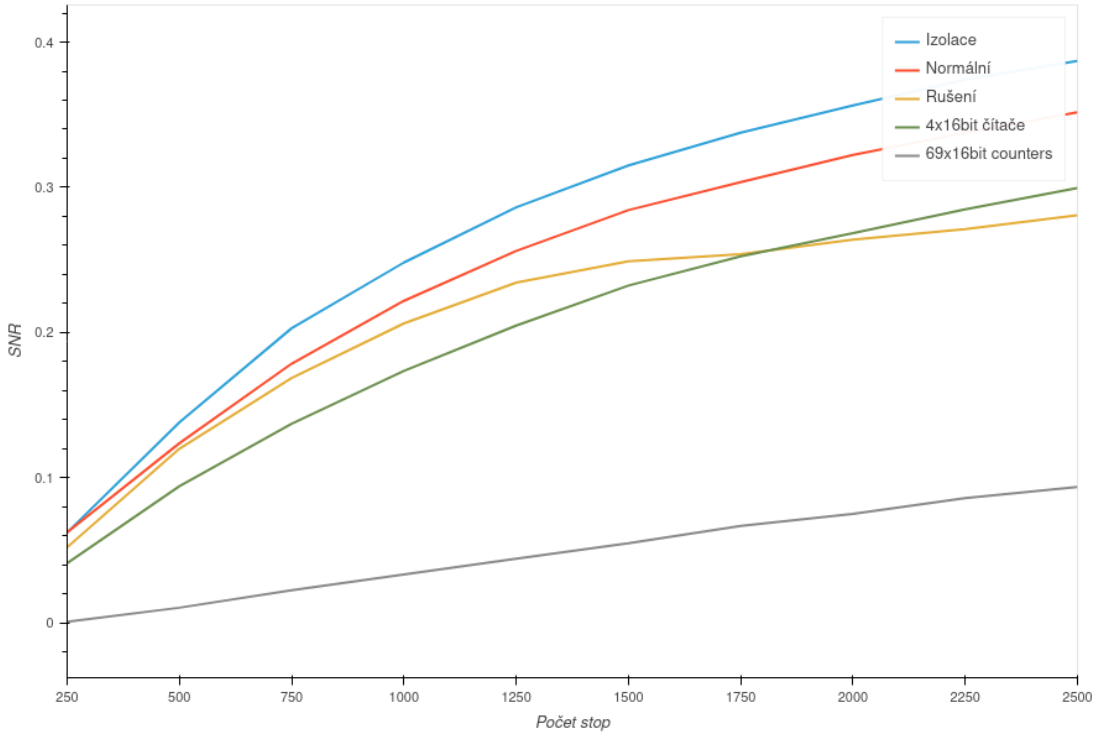
Nejsilnějším generátorem šumu bylo 69 16 bitový volně běžících časovačů. Nicméně tolik čítačů bude paměťově zatěžovat zařízení a bylo by vhodné hledat efektivnější generátor šumu.

Hlavním poznatkem tedy je, že okolní prostředí má vliv na zařízení Spartan 6 LX9 a může ovlivnit jeho bezpečnost.

Graf odstupu signálu od šumu má podobný průběh jako úspěšnost korelační odběrové analýzy. Tyto dvě veličiny mezi sebou korelují. Pro výpočet veličiny pro daný počet stop byla její hodnota vypočítáná 30 krát nad různými soubory stop a následně byla vyjádřena střední hodnota. Věřím, že kdyby došlo k vypočítání střední hodnoty veličiny z více prvků, průběhy grafů by se blížily více.



Obrázek 6.1 Úspěšnost korelační odběrové analýzy na FPGA



Obrázek 6.2 Odstup signálu od šumu na FPGA

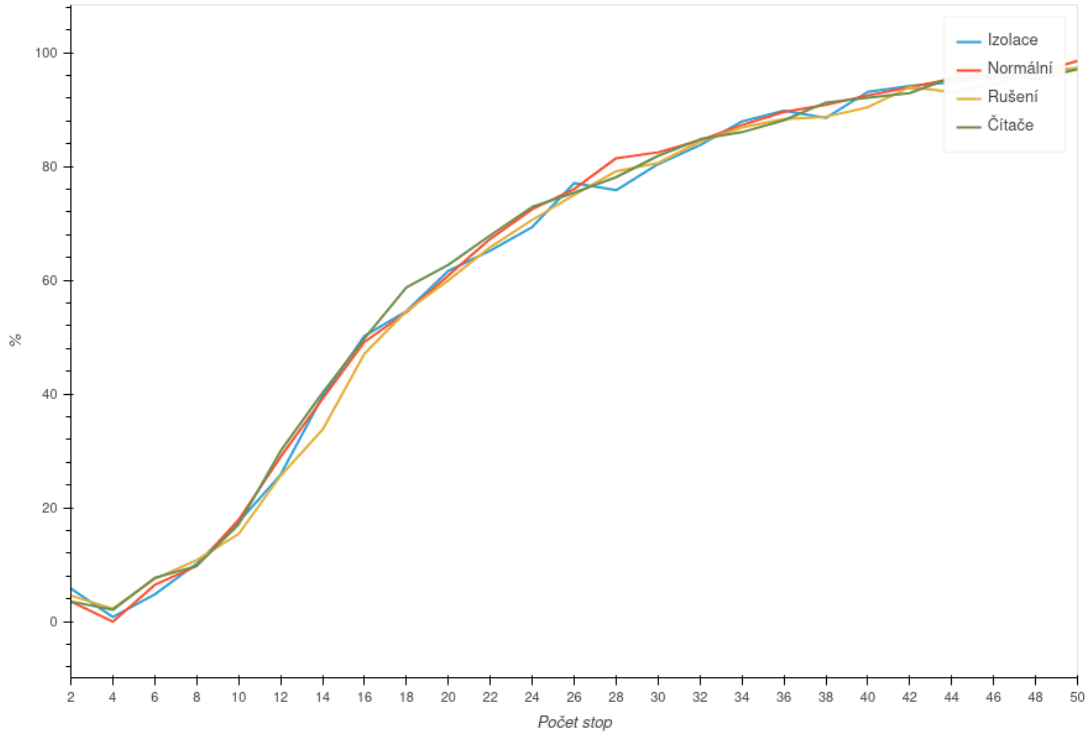
6.0.2 Mikrokontroler

Odstup signálu od šumu má stejný trend jako úspěšnost útoku. Při zvýšení signálu narostl počet získyných suklíčů.

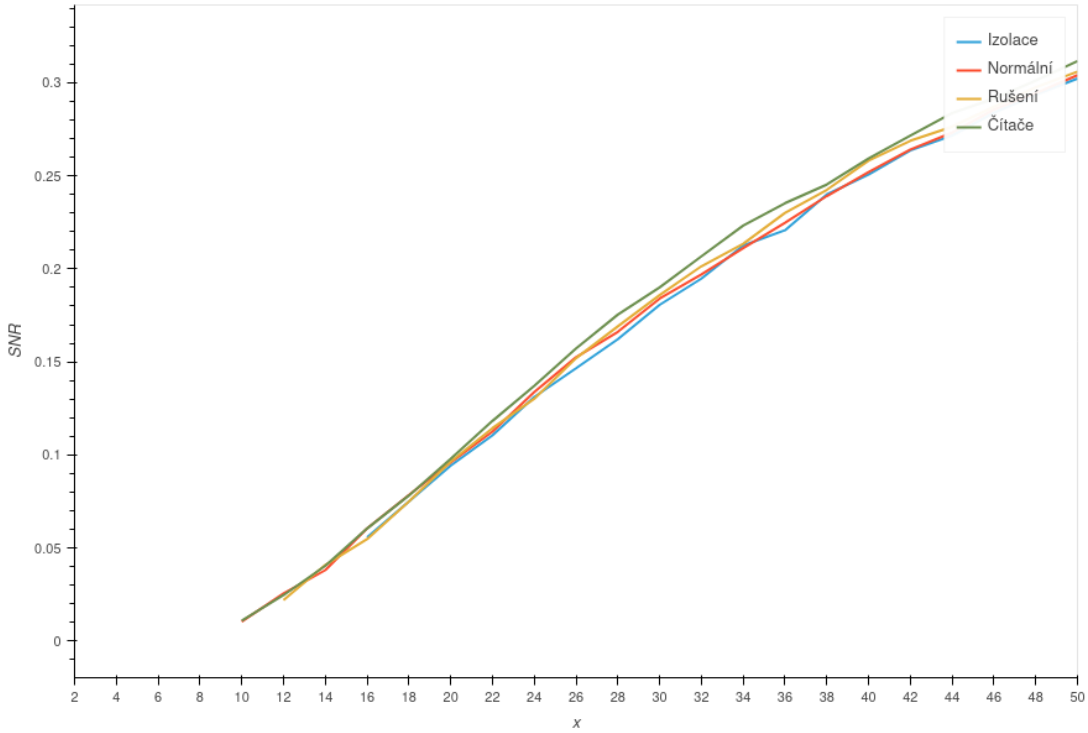
Z grafů úspěchu korelační odběrové analýzy a odstupu signálu od šumu můžeme pozorovat, že se pro mikrokontroler ATXmega128D4-AU nepovedlo generovat dostatečně veliký šum, ani zvýšit signál odstíněním natolik, aby ovlivnil bezpečnost šifry.

Pozorovatelné odchylky v úspěšnosti korelační odběrové analýzy nelze interpretovat jako kompromitaci bezpečnosti šifry. Odstup signálu od šumu nekoreluje s odchylkami a jedná se pouze o odchylky způsobené nedostatečně přesným měřením.

Bezpečnost tohoto zařízení lze zkoumat dále představením silnějšího generátoru šumu, jako je například AD převodník.



Obrázek 6.3 Úspěšnost korelační odběrové analýzy na mikrokontroleru



Obrázek 6.4 Odstup signálu a šumu na mikrokontroleru

Bibliografie

1. BOGDANOV, Andrey; KNUDSEN, Lars R.; LEANDER, Gregor; PAAR, Christof; POSCHMANN, Axel; ROBSHAW, Matthew J. B.; SEURIN, Yannick; VIKKELSOE, C. PRESENT: An Ultra-Lightweight Block Cipher. In: *CHES*. 2007, s. 450–466. Dostupné také z: http://www.iacr.org/archive/ches2007/47270450/47270450.pdf.
2. KNUDSEN, Lars Ramkilde. Block Ciphers: Analysis, Design and Applications. *DAIMI Report Series*. 1994, roč. 23, č. 485. Dostupné z DOI: 10.7146/dpb.v23i485.6978.
3. MENEZES, Alfred J; VAN OORSCHOT, Paul C; VANSTONE, Scott A. *Handbook of applied cryptography*. CRC press, 1997. ISBN 9780429466335. Dostupné z DOI: 10.1201/9780429466335.
4. DAEMEN, Joan; RIJMEN, Vincent. The block cipher Rijndael. In: *International Conference on Smart Card Research and Advanced Applications*. 1998, s. 277–284.
5. DWORKIN, Morris; BARKER, Elaine; NECHVATAL, James; FOTI, James; BASSHAM, Lawrence; ROBACK, E.; DRAY, James. *Advanced Encryption Standard (AES)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards a Technology, Gaithersburg, MD, 2001. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.197.
6. LANGDON JR, Glen G. *Computer Design*. Computeach Press, Incorporated, 1982. ISBN 0960786406.
7. HEYS, Howard M. A TUTORIAL ON LINEAR AND DIFFERENTIAL CRYPTANALYSIS. *Cryptologia*. 2002, roč. 26, č. 3, s. 189–221. Dostupné z DOI: 10.1080/0161-110291890885.
8. KOCHER, Paul C.; JAFFE, Joshua; JUN, Benjamin. Differential Power Analysis. In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Springer, 1999, sv. 1666, s. 388–397. Lecture Notes in Computer Science. Dostupné z DOI: 10.1007/3-540-48405-1_25.
9. BRIER, Eric; CLAVIER, Christophe; OLIVIER, Francis. Correlation power analysis with a leakage model. In: *International workshop on cryptographic hardware and embedded systems*. 2004, s. 16–29.
10. WAGGENER, William M. *Pulse Code Modulation Techniques*. Springer New York, NY, 1995. ISBN 978-0-442-01436-0.
11. LEWIN, FEATURE MICHAEL. All about XOR. *For details of ACCU, our publications and activities, visit the ACCU website: www. accu. org*. 2012, s. 14.

12. BENESTY, Jacob; CHEN, Jingdong; HUANG, Yiteng; COHEN, Israel. Pearson Correlation Coefficient. In: *Noise Reduction in Speech Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 1–4. ISBN 978-3-642-00296-0. Dostupné z DOI: 10.1007/978-3-642-00296-0_5.
13. BI-PST – Pravděpodobnost a statistika. In: [b.r.]. Dostupné také z: https://courses.fit.cvut.cz/BI-PST/media/lectures/BI-PST-Textbook.pdf.
14. Variance: Simple Definition, Step by Step Examples. In: [b.r.]. Dostupné také z: https://www.statisticshowto.com/probability-and-statistics/variance/.
15. Signal-to-noise ratio. In: [b.r.]. Dostupné také z: http://www.scholarpedia.org/article/Signal-to-noise_ratio%20?iframe=true&width=100%&height=100%.
16. Chipwhisperer documentation API. In: [b.r.]. Dostupné také z: https://chipwhisperer.readthedocs.io/en/latest/api.html.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `Verilog` ..... zdrojový kód implementace šifry v jazyce Verilog
- `C` ..... zdrojové kódy v jazyce C
- `jupyter` ..... jupyter notebooky
- `data` ..... naměřená data
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF