„Podporujeme výskumné aktivity na Slovensku/Projekt je spolufinancovaný zo zdrojov EÚ“

Názov projektu :

Medzinárodné centrum excelentnosti pre výskum inteligentných a bezpečných informačno-komunikačných technológii a systémov

ITMS : 26240120039

Názov výstupu :

Záverečná vedecká správa v pracovnému balíku č. 4.
(Výskum kybernetickej bezpečnosti - symetrická a asymetrická kryptografia s obmedzeniami)

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek

ARM	Ashton Raggatt McDougall (architektúra procesorov s redukovanou inštrukčnou sadou)
CA	Certification Authority
DPA	Differential Power Analysis
DNS	Domain Name System
DSA	Data Signature Algorithm
DSP	Digital signal processing
FPGA	Field programmable gate array
FQDN	Fully qualified domain name
GPIO	General purpose input/output
HPS	Hard processor system
HW	Hardware
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JTAG	Joint test action group
LQFP	Low profile quad flat package
LVDS	Low-voltage differencial signalling
LWC	Lightweight Cryptography
MCU	Microcontroller unit
PKC	Public-key cryptography
PKI	Public-key infrastructure
PRNG	Pseudorandom number generator
RNG	Random number generator
SoC	System on chip
SPA	Simple Power Analysis
SSL	Secure Socket Layer
SW	Software
SWD	Serial wire debug
TCP	Transmission Control Protocol
TLD	Top-level Domain
TLS	Transport Layer Security
QSYS	Quartus system integration tool
UART	Universal asynchronous receiver/transmitter

1 Úvod

Táto výskumná správa popisuje výsledky pre

- Projekt: Medzinárodné centrum excelentnosti pre výskum inteligentných a bezpečných informačno-komunikačných technológii a systémov,
- Aktivitu č. 2.1, teda: Základný výskum v oblasti rozvoja inteligentných a bezpečných informačno-komunikačných systémov,
- Tému: Bezpečnosť – Kryptografia, a
- Pracovný balík č. 4: Výskum kybernetickej bezpečnosti symetrická a asymetrická kryptografia s obmedzeniami.

Cieľom aktivity bola realizácia základného výskumu svetovej úrovne v oblasti rozvoja inteligentných a bezpečných informačno-komunikačných systémov (IKT). Zámery a ciele pracovného balíka č. 4 sú popísané v kapitole 1.1. Zhrnutie dosiahnutých cieľov je popísané v kapitole 1.2.

Výskum v IKT v rámci aktivity a pracovného balíka, ktorý je opísaný v tejto správe bol smerovaný na špecifickú oblasť bezpečnej komunikácie s cieľom vytvoriť pevnú vedeckú základňu, na ktorej bude možné stavať následný po projektový aplikovaný výskum.

Táto výskumná správa pokrýva celú dobu realizácie projektu, aktivity a pracovného balíka č. 4, konkrétne 01/2014 – 09/2015.

1.1 Pracovný balík 4: Výskum kybernetickej bezpečnosti - symetrická a asymetrická kryptografia s obmedzeniami

Téma Bezpečnosť je nevyhnutnou súčasťou tak pre tému Inteligentných sietí ako aj pre tému Veľkých dát. Kým súvislosť s prvou témou je o ohrozeniach, súvislosť s poslednou témou je o kvalite dát a ochrane súkromia subjektov, od ktorých sú údaje získavané. Bez spolupráce s témou Bezpečnosť nebude môcť byť ani pre ďalšie témy dostatočne ošetrená kvalita a spoľahlivosť zdrojový údajov v ich samotnom spracovaní, ale aj korektnom uchovávaní v rámci navrhnutých informačných systémov.

Zámerom v rámci tejto aktivity bolo štúdium lightweight kryptografie (LWC). LWC sa pokúša o optimalizáciu pomeru medzi bezpečnosťou a implementačnými požiadavkami na šifru smerom k šetreniu dostupného výpočtového výkonu na úkor niektorých bezpečnostných rizík. V tejto oblasti pokračuje veľmi intenzívny výskum, najmä však v oblasti návrhu nových šifier často bez dôkladnejšej kryptoanalýzy**. Našim cieľom bolo študovať princípy konštrukcie a analýzy LW symetrických šifier.**

Hoci väčšina problémov komunikačnej bezpečnosti je riešiteľná s pomocou symetrickej kryptografie, kryptografia s verejným kľúčom (PKC) dopĺňa mnohé primitíva potrebné pre

pokročilé kryptografické protokoly (špeciálne pre manažment kľúčov). Avšak pri využití PKC spravidla veľmi rastie zložitosť použitých algoritmov. Navyše, väčšina súčasných riešení nebude bezpečná pri nasadení kvantových počítačov**. Našim cieľom bolo adaptovať súčasné postupy a navrhnúť nové algoritmy pre tzv. post-quantum PKC vhodné aj pre zariadenia s obmedzeniami.**

1.2 Pracovný balík 4: Zhodnotenie dosiahnutých výsledkov a naplnenia cieľov

Časť výskumu orientovaná na **postkvantovú kryptografiu** sa zaoberala implementáciou McEliecovho kryptosystému na zariadenia s obmedzeným výpočtovým výkonom. V rámci výskumu sme na našom pracovisku pracovali s 2 rôznymi platformami - vývojovou kartou Altera Cyclone a doskou s mikrokontrólerom STM32F4. Na obe zariadenia sme **implementovali verziu McEliecovho kryptosystému** obsiahnutú v kryptografickej knižnici BitPunch, vyvíjanej na Ústave informatiky a matematiky Fakulte elektrotechniky a informatiky STU v Bratislave. Po implementácii sme obe **zariadenia podrobili kryptoanalýze** z hľadiska postranných kanálov, konkrétne sme sa zamerali na nájdenie kľúča pomocou analýzy **spotreby elektrickej energie**. Ukázalo sa, že dostupná knižnica BitPunch, resp. jej implementácia je **náchylná** na tento typ útoku a podarilo sa nám - najmä pri útoku na zariadenie STM32F4 - **odhaliť analýzou napäťových stôp tajné parametre**. To samozrejme potvrdzuje dnešný svetový trend bezpečných implementácii kryptosystémov, pretože samotná implementácia je minimálne tak dôležitá, ako bezpečnosť implementovaného algoritmu. Žiaľ, nepodarilo sa nám pre krátkosť času lepšie pripraviť merania na platforme Altera a namerať postačujúce napäťové stopy pre ďalšiu analýzu. Taktiež, postkvantové kryptosystémy zväčša obsahujú parametre, ktorých veľkosť je na rozhraní možností dnešných obmedzených zariadení, preto veľkosť parametrov McEliecovho systému implementovaného na zariadení STM32F4 bola menšia, než aká sa bežne odporúča - čo však v tomto prípade nemalo žiadny vplyv na bezpečnostnú analýzu. Avšak aj v tejto oblasti výskum napreduje a už teraz sa na našom pracovisku pracuje na implementácii verzie systému s reálne odporúčanými parametrami. Bližšie informácie k vykonanému výskumu obsahujú kapitoly 2 a 3 tejto záverečnej práce.

V oblasti **symetrických lightweight kryptosystémov** sme sa zamerali na výskum šifier založených na kvázigrupách. Tieto algebraické štruktúry majú vlastnosti vhodné pre dizajn prúdových šifier, ktoré patria k typickým lightweight šifrovacím primitívam. Preto sme skúmali bezpečnosti navrhnutých šifier s kvázigrupovou štruktúrou. Podarilo sa nám odhaliť **viacero slabín v týchto systémoch, či už priamo zlomením šifry, alebo nájdením skupiny slabých kľúčov,** preto odporúčame **opatrnosť** pri nasadzovaní týchto LW systémov v praxi. Bližšie informácie o kvázigrupových systémoch sa nachádzajú v kapitole 4 tejto práce.

Ďalšou analyzovanou oblasťou v symetrickej kryptografii boli generátory náhodných čísiel v sieťovom prostredí a taktiež na zariadeniach s obmedzeným výpočtovým výkonom. Výskumy

dokazujú, že napriek zverejneným bezpečnostným chybám v r. 2012 a r. 2013 ohľadne generovaní prvočísiel v sieťových zariadeniach a v chytrých „smart“ / čipových kartách, stále **neboli tieto bezpečnostné problémy odstránené** a je teda nutné sa zamerať **na správnu implementáciu zberu prvotnej entropie** a na **správnu implementáciou inicializovania náhodných generátorov** slúžiacich pre generovanie prvočísel pre RSA modulus alebo iného kľúčového materiálu. Bližšie informácie k vykonanému výskumu sa nachádzajú v kapitolách 5 a 6 tejto práce.

2 Útoky postrannými kanálmi na šifrovacie systémy implementované na zariadení Altera Cyclone

2.1 Metódy pre získavanie tajných informácií z kryptografických zariadení

Hlavným cieľom útočníka je získanie tajného kľúča alebo inej tajnej informácie, ktorá pomôže dešifrovať zašifrovanú správu. Existuje viacero techník ako zrealizovať tento cieľ. Ak by sme chceli rozdeliť tieto útoky, rozdelili by sme ich podľa nasledovných kritérií.

- **Pasívne útoky:** Útočník meria hodnoty fyzikálnych parametrov (napr. čas vykonávania jednotlivých operácií, spotrebu el. energie). Iným spôsobom nezasahuje do činnosti zariadenia, len pasívne pozoruje správanie zariadenia.
- **Aktívne útoky:** Útočník zámerne manipuluje vstupy zariadenia a/alebo okolité systémy, ktoré sú k nemu pripojené. Tým začne zariadenie generovať abnormálne správanie a vtedy je možné získať tajnú informáciu.

Ďalšie rozdelenie útokov je z pohľadu prístupu k jednotlivým častiam skúmaného zariadenia. Ide o tzv. invazívne, čiastočne invazívne alebo neinvazívne útoky.

- **Invazívne útoky:** Pri získavaní tajných informácií neexistuje žiaden limit, ako pozmeniť časti skúmaného zariadenia. Pri tomto útoky sa väčšinou začína s odňatím vonkajšieho obalu procesora a následné ďalších komponentov. Pomocou sond sa priamo pripojíme na potrebné systémy, ako je dátová zbernica, pamäť atď. Môžeme tu využiť pasívny aj aktívny útok. Zariadenia, ktoré sa využívajú pri tomto spôsobe, sú veľmi drahé. Na odstránenie púzdra integrovaného obvodu potrebujeme laserový nôž. Ďalej je potrebné mať špeciálne meracie vybavenie, ktoré disponuje meracími mikro-sondami. V zvláštnych prípadoch je potrebný aj iónový implantátor, ktorý sa používa pri výrobe integrovaných obvodov. Napriek všetkému je táto metóda najúčinnejšia.
- **Čiastočne invazívny útok:** V tomto útoku je taktiež potrebné odstrániť púzdro integrovaného obvodu. Na rozdiel od invazívneho prístupu už nie je potrebné odstrániť pasivačnú vrstvu a nie je potrebné sa priamo pripojiť (prostredníctvom sondy) na povrch čipu. Ak je použitý pasívny útok, útočník sa zameriava na prečítanie obsahu pamäte. Aktívny prístup využíva röntgenové, elektromagnetické alebo viditeľné žiarenie (svetlo). Osvetlením určitej časti čipu dosiahneme správanie, z ktorého vieme zistiť tajnú informáciu. Čiastočne invazívny útok nevyžaduje vlastniť drahé vybavenie v porovnaní s invazívnym útokom, no napriek tomu, úsilie vynaložené pri tomto útoku je oveľa vyššie ako pri invazívnom. Je to spôsobené tým, že nájdenie správnej časti čipu vhodnej pre útok vyžaduje veľa času a skúseností.
- **Neinvazívne útoky:** Pri tomto útoku už útočník nepotrebuje mechanicky pozmeniť skúmané zariadenie. Podľa druhu útoku mu stačí využiť topológiu zariadenia takú,

aká je a priamo sa k nemu pripojiť pomocou vhodného meracieho zariadenia. Tieto meracie zariadenia sú relatívne lacné. Pri útoku sa najčastejšie používa tzv. časový útok, elektromagnetický útok alebo analýza spotreby el. energie. Fakt, že je možné zistiť tajnú informáciu pomocou bežne dostupných meracích prístrojov a hlavne bez žiadneho náznaku porušenia, predstavuje tento typ útoku vážnu hrozbu pre kryptografické zariadenia. Tieto útoky sa v literatúre popisujú, ako útoky cez postranné kanály. Často ich kategorizujeme do skupiny aktívnych neinvazívnych útokov.

2.1.1 Meranie postranných kanálov

V rámci výskumného projektu sme sa zaoberali bezpečnosťou tzv. embedded aplikácií. V súčasnosti sa na zabezpečenie bezpečnosti v komunikácii, či už prostredníctvom internetu alebo priamo, formou elektronického dokumentu, používajú špecializovane hardvérové (HW) zariadenia, ktoré zabezpečujú šifrovanie/dešifrovanie správy. Výhodou takéhoto riešenia je vysoká bezpečnosť tajného (privátneho) kľúča, ktorý je uložený v pamäti HW zariadenia. Neexistuje teda žiadna možnosť, ako sa dostať k tomuto kľúču, pretože toto zariadenie nepodporuje pripojenie na internet a teda nie je možné na neho vykonať útok, resp. uplatniť taktiky crackerov. Aj keby narušitelia dokonale poznali softvérové a hardvérové špecifiká zariadenia, nie je možné využiť tieto znalosti a získať tajný kľúč. Ani cez ovládanie na úrovni softvérovej (firmware, operačný systém), ani hardvérovej (priame čítanie obsahu pamäte celého obvodu). Jedinou možnosťou, ako sa dá k tajnému kľúču dostať, je reálne zmerať elektrické signály procesora (logiky, vykonávajúcej šifrovanie/dešifrovanie). Tento útok je známy pod menom meranie postranných kanálov. Je viacero typov meraní postranných kanálov [1], [2], [3]:

1. Úzkopásmové meranie akustických signálov
2. Úzkopásmové meranie elektrického potenciálu
3. Jednoduchá analýza elektrickej spotreby
4. Diferenciálna analýza elektrickej spotreby
5. Meranie časových rozdielov v bežiacom algoritme

Pre naše účely je najvýhodnejšia práve diferenciálna analýza elektrickej spotreby.

2.1.1.1 Diferenciálna analýza spotreby

Hlavným cieľom diferenciálnej analýzy je získanie tajného kľúča. Túto informáciu je možné získať z analýzy veľkého množstva nameraných vektorov spotreby el. energie. Vektor spotreby je potrebné merať počas kryptografickej operácie (šifrovanie alebo dešifrovanie). Táto metóda sa dá využiť aj vtedy, ak nemáme k dispozícii detailnú špecifikáciu skúmaného zariadenia (napr. elektrické zapojenie). Cieľom útočníka je analyzovať elektrickú spotrebu v presne danom časovom momente pre všetky vektory spotreby. Analýza spočíva v nájdení závislosti medzi dátami a spotrebou. Celá stratégia diferenciálnej analýzy spotreby sa dá zhrnúť do týchto bodov:

1. Voľba parametra vhodného na analýzu spotreby.
2. Meranie spotreby energie.
3. Vyrátanie očakávaných hodnôt pre zvolený parameter.
4. identifikácia očakávaných hodnôt v nameranej spotrebe energie.
5. Porovnávanie očakávanej a nameranej spotreby energie.

2.1.1.2 Voľba parametra vhodného na analýzu spotreby

Ako prvý krok je potrebné vhodne zvoliť parameter, ktorý je závislý na šifrovacích dátach d a kľúčak. Inými slovami, zvolíme si parameter, ktorý je funkciou$f(d,k)$. Vo väčšine prípadov je hodnota$d$totožná zo zašifrovanou alebo dešifrovanou správou.

2.1.1.3 Meranie spotreby energie

Ako druhý krok, pri diferenciálnom útoku, je samotné meranie elektrickej spotreby čipu počas šifrovania alebo dešifrovania rôznych blokov dát d. Pre každý beh šifrovania resp. dešifrovania musí byť hodnota d známa. Známe hodnoty môžeme zapísať ako dátový vektor $\boldsymbol{d} = (d_1, \dots, d_D)$, kde d_i predstavuje hodnotu dát pre *i*-ty beh šifrovania resp. dešifrovania. Útočník si zaznamená každý beh do vektora spotreby $t_i^{'} = (t_{i,1}, \dots, t_{i,T})$, kde T je dĺžka vektora. Útočník zmeria vektor spotreby pre všetky dáta$D$, čiže vytvorí maticu $\boldsymbol{T}$o dimenzii $D \times T$. Je nevyhnutné, aby všetky namerané vektory spotreby boli správne zarovnané. Presnejšie povedané, aby $\boldsymbol{t}_j$(stĺpce matice $\boldsymbol{T}$), ktoré reprezentujú spotrebu energie, predstavovali tie isté behy šifrovania/dešifrovania. Samozrejme, len pre iné dáta. Pre tieto účely je potrebné mať nejaký pomocný trigger signál, pomocou ktorého by sme vedeli identifikovať začiatok a koniec šifrovania/dešifrovania. V praxi tento signál väčšinou nie je prítomný, v tom prípade sa útok sťažuje, ale je stále uskutočniteľný.

2.1.1.4 Vyrátanie očakávaných hodnôt pre zvolený parameter

V tomto kroku sa vypočíta očakávaná hodnota sledovaného parametra pre všetky možnosti kľúča k. Jednotlivé možnosti zapíšeme do vektora $\boldsymbol{k} = (k_1, \dots, k_K)$, kde K je počet možností kľúča k. Prvky vektora $\boldsymbol{k}$ sa zvyknú nazývať ako kľúčová hypotéza. Útočník môže, pre daný vektor dát $\boldsymbol{d}$ a kľúčovú hypotézu $\boldsymbol{k}$, jednoducho vypočítať funkciu $f(d,k)$ pre všetky D behy šifrovania/dešifrovania a pre všetky K kľúčové hypotézy. Výsledkom tohto kroku je matica $\boldsymbol{V}$ o veľkosti $D \times K$. Celý proces môžeme formálne zapísať takto:

$$v_{i,j} = f(d_i, k_j) \text{ pre } i = 1, \dots, Dj = 1, \dots, K$$

j-ty stĺpec matice $\boldsymbol{V}$obsahuje očakávané hodnoty vypočítané na základe kľúčovej hypotézy k_j. Jeden zo stĺpcov matice $\boldsymbol{V}$ obsahuje očakávanú hodnotu, ktorá bude počítaná počas jedného behu šifrovania resp. dešifrovania. Treba ešte poznamenať, že vektor $\boldsymbol{k}$ obsahuje všetky možnosti pre k, čiže hodnota, ktorú skúmané zariadenie spracujepri svojej činnosti je vlastne prvok tohto vektora. Označme index týchto prvkov ako ck. Kľúč zariadenia potom zapíšeme ako k_{ck}. Cieľom útočníka je teda nájsť, ktorý stĺpec matice $\boldsymbol{V}$ sa spracováva počas D-teho behu šifrovania alebo dešifrovania. Ako náhle vieme, ktorý stĺpec matice $\boldsymbol{V}$sa v zariadení vykonáva, automaticky vieme kľúč k_{ck}.

2.1.1.5 Identifikácia očakávaných hodnôt

Následne, po zostavení matice $\boldsymbol{V}$,spárujeme očakávanú spotrebu energie zapísanú tiež pomocou matice. Označme ju ako $\boldsymbol{H}$. Na výpočet očakávanej spotreby použijeme modely pre spotrebu obvodu. Existuje veľa modelov [2], tie najjednoduchšie vychádzajú napr. z určenia Hammingovej vzdialenosti medzi počtom jednotiek a núl za určitý časový interval.

2.1.1.6 Porovnávanie očakávanej spotreby s nameranou spotrebou

Keď máme spárované prvky matice $\boldsymbol{V}$ a $\boldsymbol{H}$, môžeme pristúpiť k záverečnému kroku. T.j. k porovnaniu všetkých stĺpcov h_i matice $\boldsymbol{H}$ so stĺpcami t_j matice $\boldsymbol{T}$. Inými slovami, útočník porovná očakávané hodnoty spotreby pre každú kľúčovú hypotézu s nameranou spotrebou tiež pre všetky sledované behy. S tohto porovnania vznikne matica $\boldsymbol{R}$ o veľkosti $K \times T$, kde každý prvok $r_{i,j}$ obsahuje jedno porovnanie stĺpcov h_i a t_j. Porovnanie nie je jednoduchá záležitosť, nakoľko získané vzorky prirodzene obsahujú aj šum. Existuje viacero spôsobov ako spoľahlivo porovnať tieto matice. Známe sú prístupy využívajúce korelačnú analýzu alebo Hammingovú vzdialenosť.

2.2 Aplikovanie diferenciálnej analýzy spotreby na extrakciu tajnej informácie

V tejto kapitole detailne popíšeme aplikovanie diferenciálnej analýzy spotreby na konkrétne zariadenie. Náš útok bol prevedený na vývojovú dosku Socrates od firmy EBV (obr. 2.1). Obsahuje množstvo doplnkových obvodov zabezpečujúcich dátovú komunikáciu (USB, Ethernet, SD karta) a programovanie a ladenie zariadenia (3xUSB). Na doske sú integrované aj rôzne senzory, kontrolné LED diódy a prepínače. Jadrom vývojovej dosky je programovateľný obvod SoC (System on Chip) od spoločnosti Altera (5CSEBA6U23C7N), ktorý obsahuje dvoj jadrový procesor s architektúrou ARM Cortex A9 a plne programovateľnú časť na báze programovateľných hradlových polí FPGA. Parametre sú nasledovné:

AlteraCyclone V SoC doska:

- 5CSEBA6U23C7N
- 110K LE (logické elementy)
- 112 DSP blokov
- 5.4 Mbit RAM
- HPS (HardProcessorSystem, 800 MHz, dualcore)

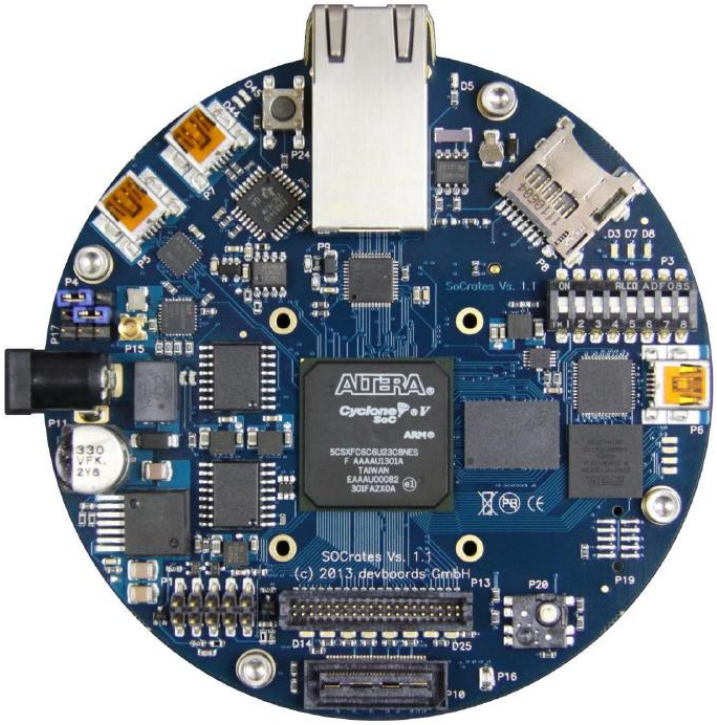
Rozhrania:

- 1Gbit Ethernet
- USB 2.0 OTG
- CAN, SPI, I²C
- UART-USB konvertor

- 3.3V GPIO (62)
- Užívateľské LED diódy (8+3)
- DIP prepínače (8 polôh)
- Joystick
- LVDS konektor
- LCD TFT rozhranie
- Embedded USB Blaster II (programovacie rozhranie)

Pamäť:

- 1Gbyte DDR3 s ECC
- µSDCard Slot (8Gbyte)
- 2x EPCQ256 (konfiguračné zariadenie)



Obr. 2.1 Vývojová doska Socrates na báze FPGA

Ešte pred samotným meraním (útokom) bolo potrebné inicializovať vývojovú dosku. Boli vykonané nasledovné kroky:

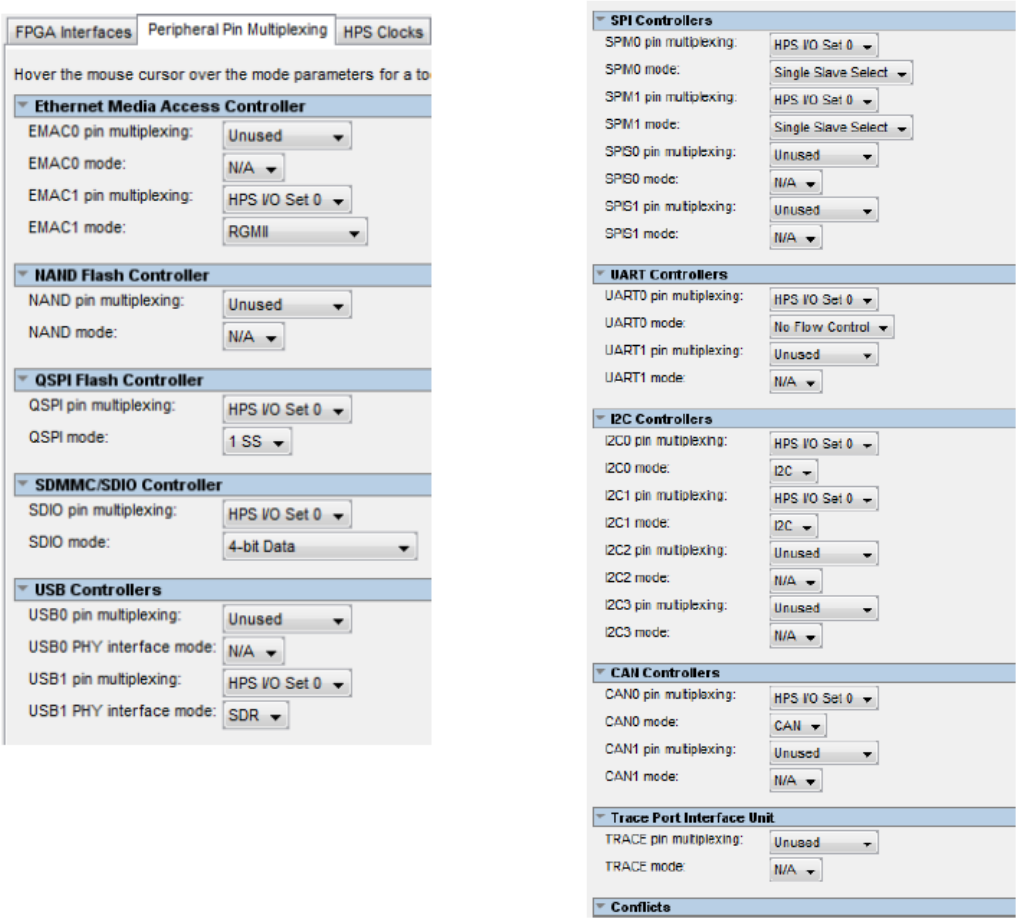
1. Prepojenie vývojového prostredia QUARTUS 13.1 s vývojovou doskou.
2. Nainštalovanie operačného systému Linux socfpga cyclone5 na vývojovú dosku.
3. Implementovanie McEliece kryptosystému a pomocných HW komponentov.
4. Meranie postranných kanálov.
5. Vyhodnocovanie meraní.

2.2.1 Prepojenie vývojového prostredia QUARTUS 13.1 s vývojovou doskou

Vývojové prostredie QUARTUS je softvér, ktorý obsahuje všetky potrebné nástroje na vývoj aplikácií na ľubovoľnom čipe od spoločnosti ALTERA, ďalej umožňuje simuláciu a ladenie aplikácie. Toto prostredie je potrebné na konfiguráciu a prepojenie čipu 5CSEBA6U23C7N so všetkými komponentmi na doske. Ide o externé periférie ako sú pamäť RAM, Ethernet, SD karta a ďalšie komunikačné zbernice (USB, I2C, SPI). Tieto úkony sa vykonávajú pomocou podprogramu QSYS. Potrebné nastavenia sú nasledovné:

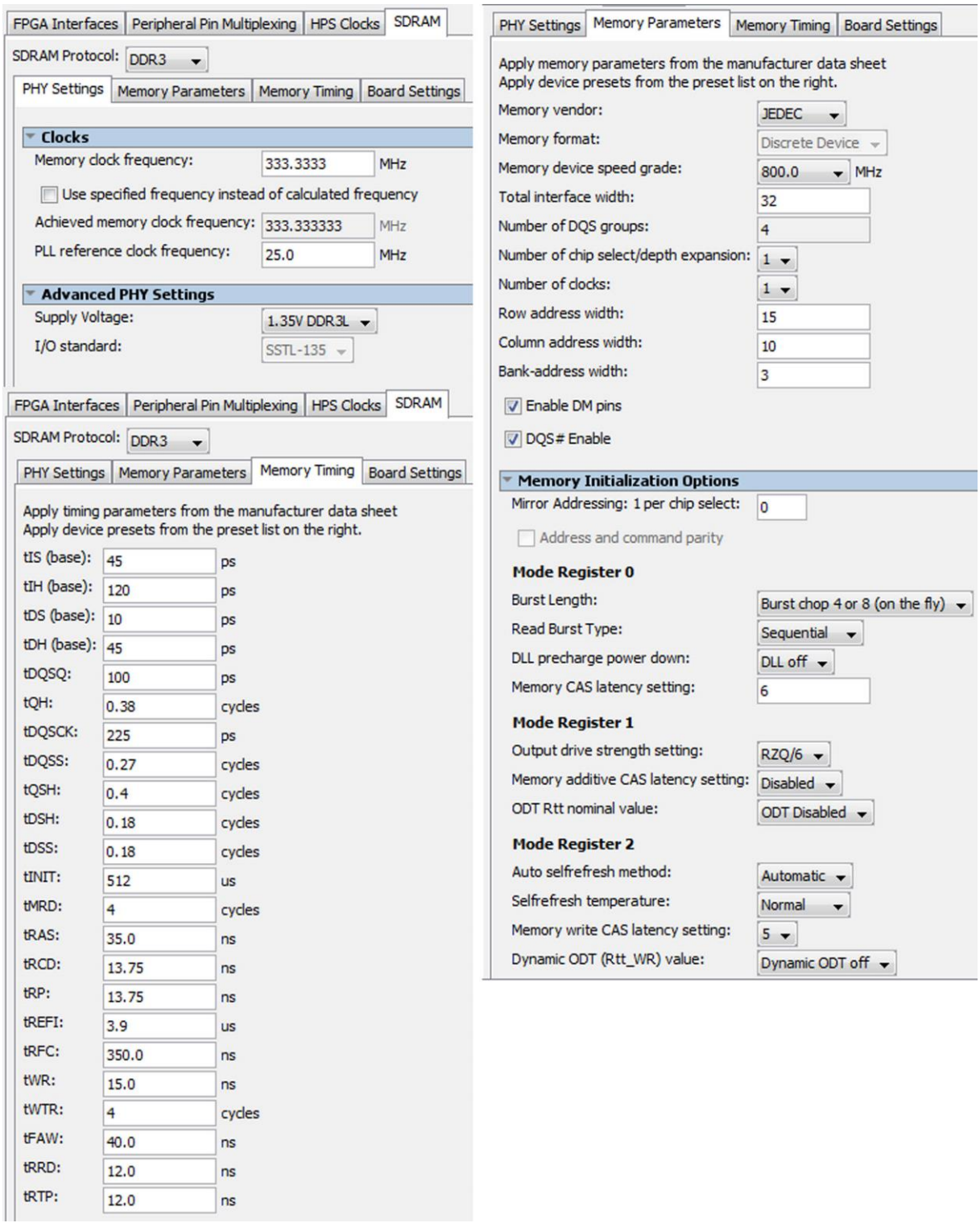
Nastavenie periférií (obr. 2.2):

USB 1 (Set0) , SDR, SPIM0 (Set0, single slaveselect), SPIM1 (Set0, single slaveselect), UART0 (Set0, No FlowControl), I2C0, I2C1 (Set0), CAN0 (Set0), GPIO (GPIO00, GPIO09, GPIO28, GPIO35, GPIO37, GPIO40, GPIO41, GPIO42, GPIO43, GPIO44, GPIO48, GPIO53 and GPIO54)



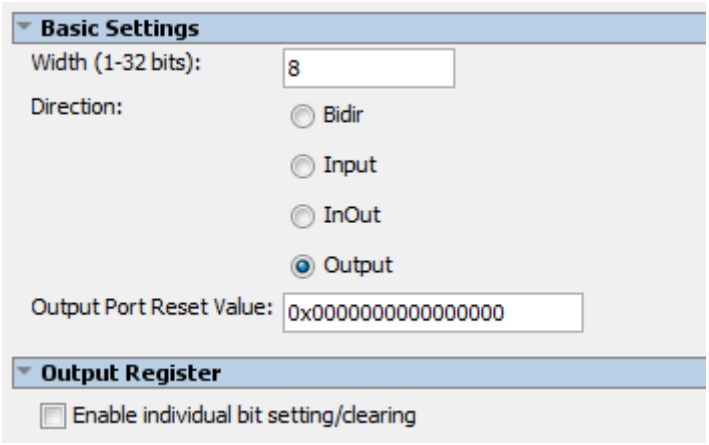
Obr. 2.2 Nastavenie periférií a I/O vývodov v QSYS-e

Nastavenie pamäte RAM (obr. 2.3):



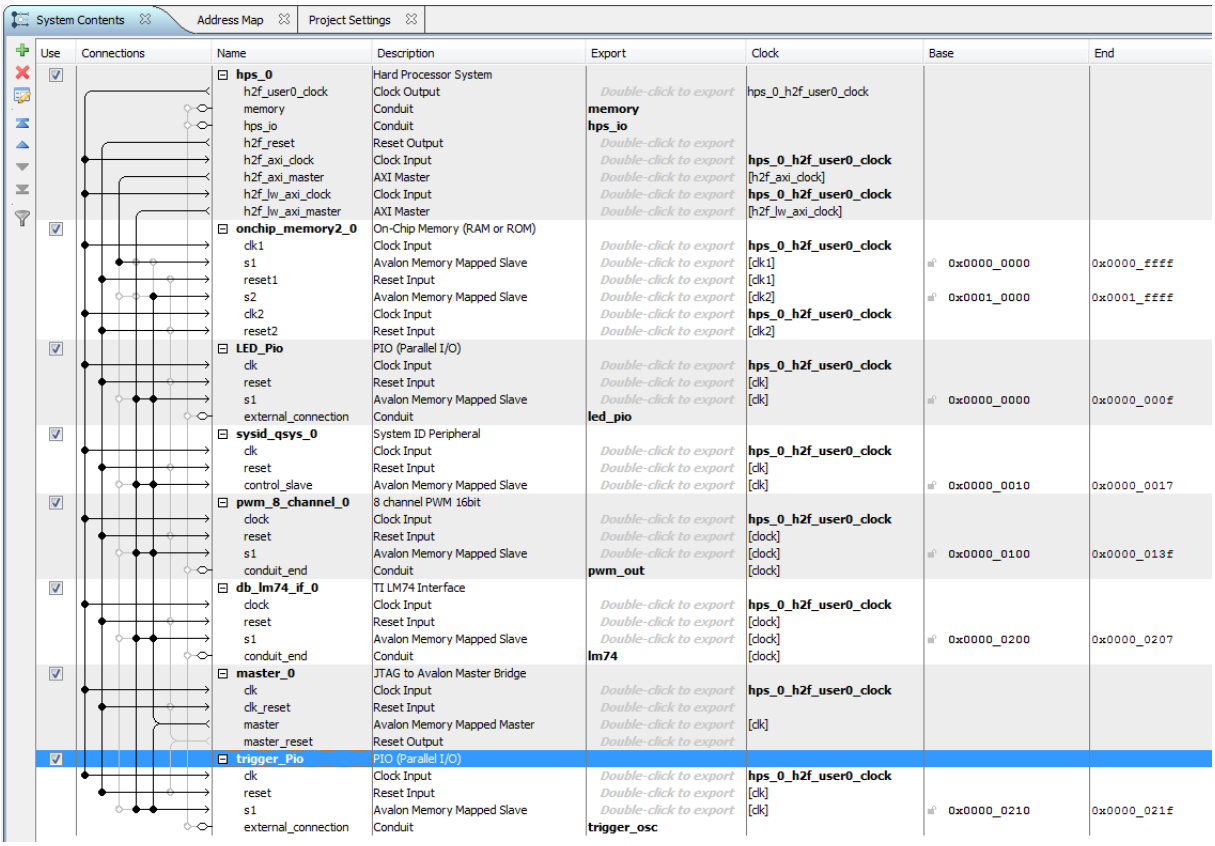
Obr. 2.3 Nastavenie parametrov pamäte RAM v QSYS-e

Nastavenie kontrolných LED diód a pomocného „trigger“ signálu (obr. 2.4):



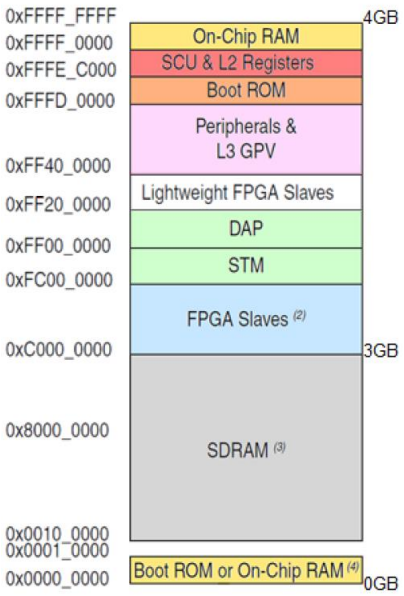
Obr. 2.4 Nastavenie parametrov pre LED a PIO (Parallel I/O) v QSYS-e

Podprogram QSYS umožňuje prepojiť ľubovoľné nakonfigurované komponenty. Pomocou grafického rozhrania sme nakonfigurovali SoC podľa tohto plánu (obr. 2.5):



Obr. 2.5 Celková konfigurácia SoC čipu v QSYS-e

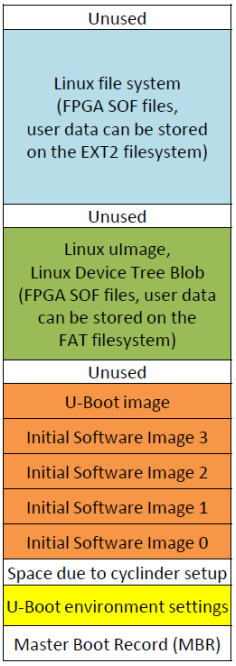
Pri prepojovaní komponentov bolo nevyhnutné nastaviť správnu adresu každého jedného komponentu. Adresa slúži pre korektné komunikovanie ARM procesora. Hodnoty adries nie sú vybrané náhodne. Rešpektovali sme odporúčania výrobcu a zadali adresy z povoleného rozsahu (obr. 2.6):



Obr. 2.6 Adresný priestor ARM procesora v SoC systéme

2.2.2 Nainštalovanie operačného systému Linux socfpga_cyclone5

Prvým krokom je príprava miesta, kde bude nainštalovaný Linux distribúcia. Doska podporuje zavádzanie systému z SD karty, preto sme využili túto možnosť. SD kartu bolo potrebné pred samotným inštalovaním správne naformátovať. Formátovanie bolo potrebné presne dodržať podľa daného adresného priestoru definovaného výrobcom (obr. 2.7).

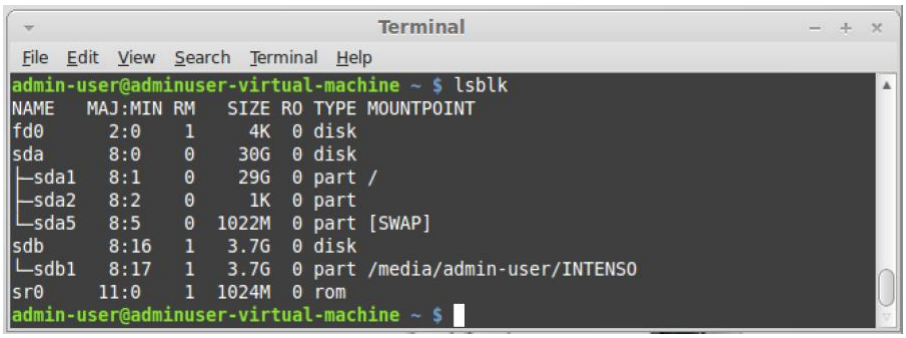


Obr. 2.7 Rozloženie diskových partícií na SD karte

Postup tvorby partícií bol nasledovný:

Krok 1 - Formátovanie SD karty (4GB) v prostredí Linux Ubuntu 12.04LTS:

1. **Zavoláme fdisk z terminálu:**
   % sudofdisk /dev/devb
2. **Vytvoríme 3 partície:**
   Partícia1: n, p, 1, 2048, 4096, t, a2
   Partícia2: n, p, 2, 4097, +20M, t, 2, c
   Partícia3: n, p, 3, 45057, , t, 83
   Pre kontrolu stlačíme: p
   Zápis partícií vykonáme stlačením: w
3. **Naformátujeme FAT32 partíciu:**
   % sudomkdosfs /dev/sdb2
   Výsledok by mal vyzerať nasledovne (obr. 2.8)



Obr. 2.8 Výsledok formátovania SD karty

Krok 2 - Kopírovanie súborov na vytvorené partície SD karty:

1. **Kopírovanie dodaného diskového obrazu Linuxu na partíciu sdb3**
   % sudodd if=rootfs.ext3 of=/dev/sdb3
2. **Kopírovanie preloadera a bootloadera na partíciu sdb1**
   % sudoddif=preloader-mkpimage.bin of=/dev/sdb1 bs=64k seek=0
   % sudoddif=u-boot.img of=/dev/sdb1 bs=64k seek=4

2.2.3 Implementácia McEliece kryptosystému a pomocných komponentov

McEliecov kryptosystém je systém s verejným kľúčom, ktorý využíva lineárny samoopravný kód. Pre neho existujú rýchle dekódovacie algoritmy, tzv. Goppove kódy. Bezpečnosť tohto systému zabezpečuje špecifická vlastnosť matematických operácií, ktoré sa pri dešifrovaní vykonávajú. Ide totižto o NP úplný problém, t.j. výsledok je vypočítateľný v nedeterministickom polynomiálnom čase. Pri dekódovaní sa uplatňujú samoopravné kódy. Ich vlastnosťou je, že dokážu opraviť určité množstvo chýb, ktoré vznikli, či už pri prenose v silne zašumenom kanály, alebo úmyselne. McEliecov kryptosystém nie je v praxi často používaný, kvôli väčšej výpočtovej náročnosti a väčšiemu dátovému prenosu. Napriek tomu je vhodným kandidátom na výskum, pretože je odolný voči najviac obávanej hrozbe pre šifrovanú komunikáciu – kvantovými počítačmi. Mechanizmus šifrovania a dešifrovania prebieha podľa nasledovného scenáru. Predstavme si komunikáciu medzi 2 účastníkmi Alicou a Bobom. Alica si zvolí binárny lineárny kód $C\ (n, k)$, ktorý je schopný opraviť t chýb.

Následne, pre daný kód C, vygeneruje maticu $\boldsymbol{G}$. Ďalej si zvolí náhodnú binárnu nejednotkovú maticu $\boldsymbol{S}$ rozmeru $k \times k$ a permutačnú maticu $\boldsymbol{P}$ rozmeru $n \times n$. Ďalším krokom je vypočítanie matice $\boldsymbol{G}$'

$$\boldsymbol{G}' = \boldsymbol{S}^{-1} \cdot \boldsymbol{G} \cdot \boldsymbol{Q}^{-1}.$$

Rozmer $\boldsymbol{G}$ je $k \times n$. Verejným kľúčom je $\boldsymbol{G}$', t a tajným $\boldsymbol{S}$, $\boldsymbol{G}$, $\boldsymbol{P}$. Ak chce Bob poslať správu pre Alicu, zašifruje správu m ako reťazec dĺžky k podľa nasledujúceho vzorca:

$$c' = m \cdot \boldsymbol{G}'.$$

Ďalej si vygeneruje náhodný vektor z s dĺžkou n a váhy t. Bob vytvorí zašifrovaný text podľa vzťahu:

$$c = c' + z.$$

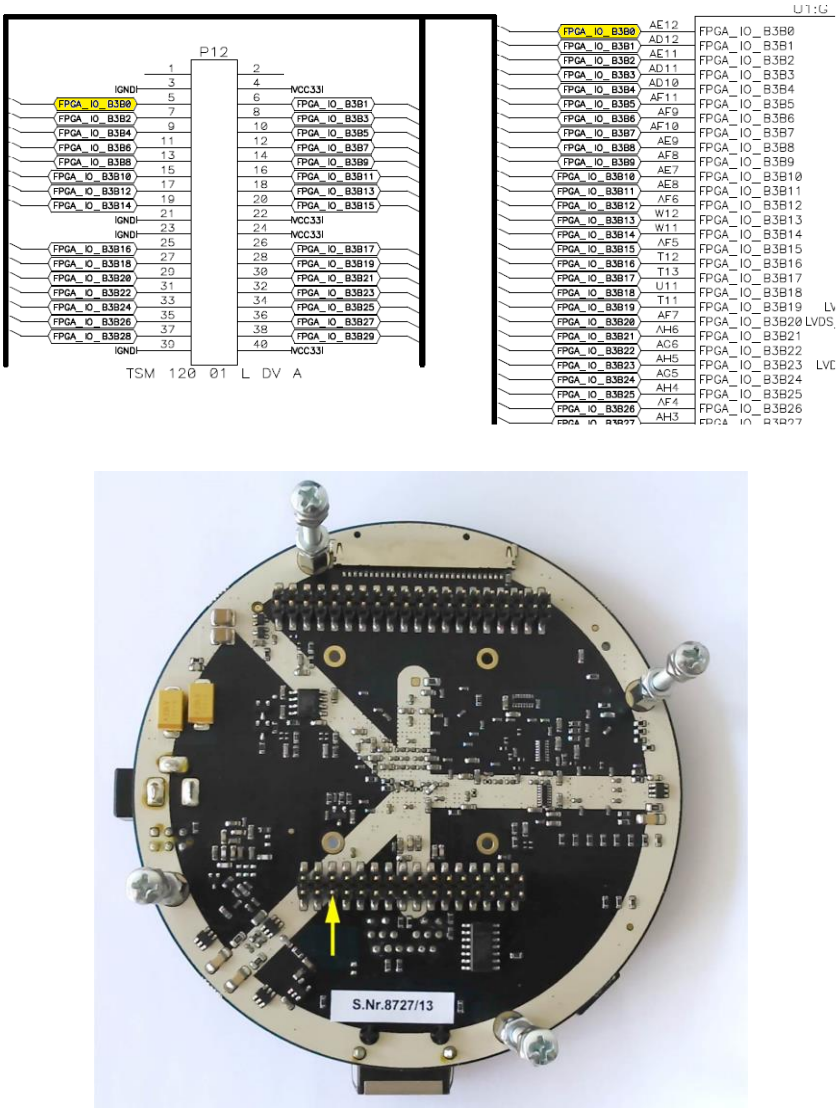
Dešifrovanie na Alicinej strane prebehne tak, že si najprv vypočíta inverznú maticu $\boldsymbol{P}^{-1}$a následne vektor c':

$$c' = c \cdot \boldsymbol{P}^{-1}.$$

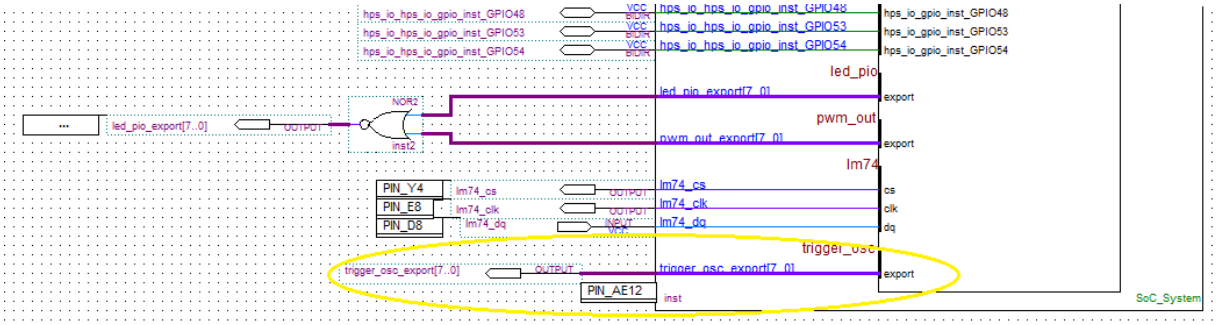
Alica použije dekódovací algoritmus pre zašifrovanú správu c. Vektor c' dekóduje na vektor m' a ten podľa vzorca vypočíta dešifrovanú správu ako:

$$m = m' \cdot \boldsymbol{S}^{-1}.$$

Tieto algoritmy sú implementované v open-source knižnici BitPunch v.2 [5]. Bohužiaľ, implementácia bola napísaná pre architektúru procesorov *x86* a teda nebola vhodná pre architektúru procesora ARM. Boli preto potrebné úpravy v tomto kóde. V hlavnej *main.c* funkcii, ako aj v súbore *goppa.c*. Úpravy spočívali v odstránení nepodporovaných knižníc pre ARM architektúru. Ďalšie úpravy spočívali v prispôsobení dešifrovacieho procesu na meranie postranných kanálov. Išlo o generovanie tzv. trigger signálu, ktorý má za úlohu synchronizovať osciloskop s práve prebiehajúcim dešifrovaním správy. Aj keď sa na prvý pohľad zdalo, že to bude jednoduchá úloha, nebolo tomu tak. Bolo potrebné urobiť zásah do hardvérového nastavenia vnútri SoC obvodu tak, aby mohol procesor ARM fyzicky nastaviť príslušný vývod SoC na požadovanú logickú hodnotu. Vývod sme zvolili tak, aby sme sa vedeli na neho pripojiť meracou sondou osciloskopu (obr. 2.9). Spomínané hardvérové nastavenia sa týkali doprogramovania FPGA entity v jazyku Verilog. V QSYS sme túto entitu (trigger_PIO,obr. 2.5) pripojili k procesoru (HPS entita) a pridelili bázovú adresu 0x0000 0210 (fyzická adresa 0xFF20 0210). Hodnota adresy môže byť v rozsahu 0xFF20 000 až 0xFF40 0000, podľa manuálov z [4]. Pripojenie entity samozrejme ešte nestačilo na to, aby s ňou procesor vedel pracovať. Bolo potrebné editovať hlavnú entitu, tzv. top-level entitu celého SoC obvodu. To sme vykonali v grafickom móde podľa obr. 2.10.

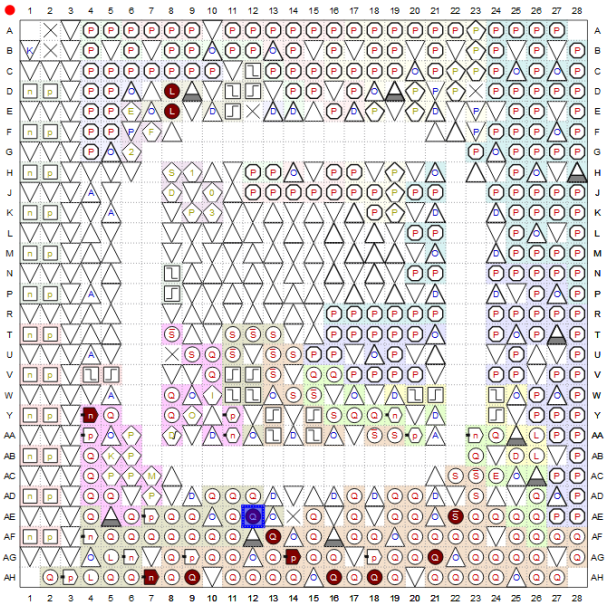


Obr. 2.9 Umiestnenie trigger signálu na konektore P12 (spodná strana dosky)



Obr. 2.10 Pripojenie entity trigger_PIO do top-level entity SoC_System

Následne sme mohli priradiť číslo vývodu, ktorý prislúcha konektoru P12 (PIN AE12 obr. 2.11).



Obr. 2.11 Priradenie umiestnenie vývodu AE12 na entitu trigger_PIO v aplikácii PinPlanner (QUARTUS)

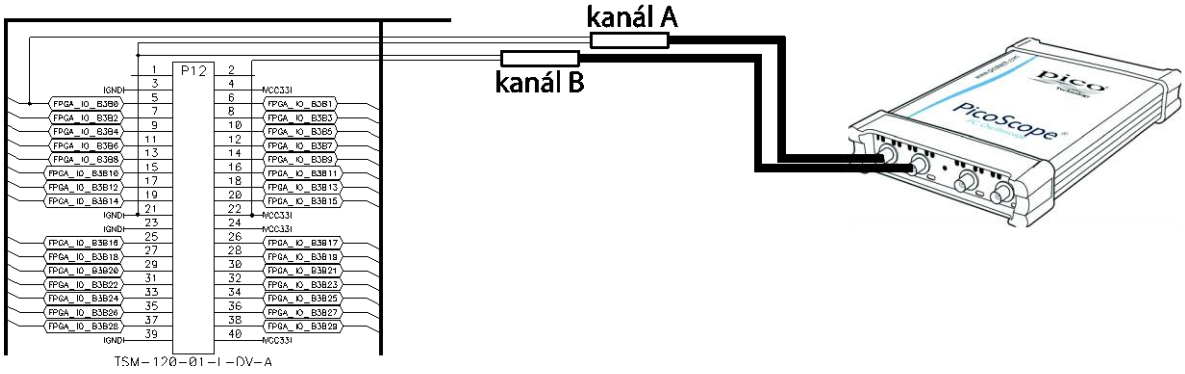
2.2.4 Meranie postranných kanálov

Na meranie postranných kanálov sme použili digitálny USB osciloskop PicoScope 6403A [6] spolu s pasívnymi sondami TA101 x10 350 MHz. Hlavné parametre osciloskopu sú v tab. 2.1.

Tab. 2.1 Základné parametre osciloskopu PicoScope 6403A

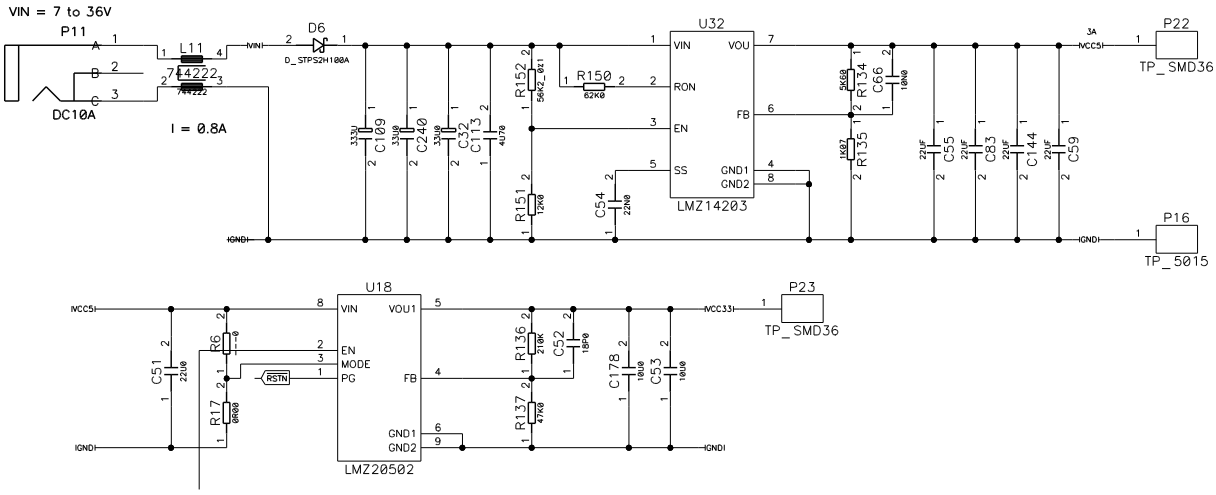
Počet kanálov	4
Vertikálne rozlíšenie	8 bit
Šírka pásma	350 MHz
Maximálna vzorkovacia frekvencia	5 GS/s
Časové základne	10ns/div – 200s/div
Pamäť	256 MS

Schéma meracieho pracoviska je zobrazená na obr. 2.12. Všetky meracie body sú umiestnené na konektore P12. Kanál A meria trigger signál, pomocou ktorého vieme identifikovať čas dešifrovacieho procesu, a vieme cielene analyzovať daný časový úsek. Kanál B meria fluktuácie v napájacej vetve SoC čipu v ktorom sa dešifrovací proces cyklicky opakuje. Na obr. 2.13 môžeme vidieť detail napájacej časti vývojovej dosky. Napájanie je zabezpečené prostredníctvom sieťového adaptéra (12V DC/1.25A) pracujúcom na princípe impulzového meniča. Nie je to teda klasicky sieťový adaptér s transformátorom. Stabilizačný obvod U32 je napäťový stabilizátor pracujúci na impulznom princípe. Jeho pracovná frekvencia je 620 kHz a výstupné napätie 5V. Obvod sa vyznačuje malým šumom a vysokou stabilitou výstupného napätia. Toto napätie je pripojené na rôzne zariadenia, ale aj na ďalšiu stabilizačnú vetvu, ktorá generuje 3.3V pre SoC čip. Tento čip pracuje aj s inými napäťovými úrovňami, nás ale zaujíma len 3.3V vetva, pretože tá napája zabudovaný ARM procesor.



Obr. 2.12 Naznačenie meracích bodov v schéme (konektor P12)

A keďže nás zaujímajú procesy, ktoré sa odohrávajú len v procesore, logicky, musíme sledovať práve toto napätie. Napätie 3.3V sa získava prostredníctvom stabilizátora U18. Je to tiež impulzný menič s fixnou frekvenciou meniča 3MHz. Vedomosť, s akou frekvenciou meniče pracujú, nám môže byť nápomocná pri filtrovaní/odstránení týchto frekvenčných zložiek. Na schéme z obr. 2.13 nie je zrejmé prepojenie medzi konektorom P22 a vstupom pre stabilizátor U18. Reálne tam samozrejme existuje. Z konektora P23 potom vychádzajú vodivé cesty do SoC obvodu, ale aj iných zariadení na doske, ktoré potrebujú 3.3V. Táto skutočnosť nám sťažuje meranie postranných kanálov, pretože nám zanáša do užitočného signálu aj nechcené frekvenčné zložky, ktoré pôsobia ako nadbytočný šum.



Obr. 2.13 Časť napájania vývojovej dosky

2.2.5 Metodika a vyhodnocovanie merania postranných kanálov

Metodika merania je založená na analýze vektorov spotreby. Každý vektor spotreby reprezentuje špecifickú operáciu v procese dešifrovania. Náš útok sa zameral na odhalenie tzv. permutačnej matice $\boldsymbol{P}$. Algoritmus McEliece sme upravili pre potreby tohto útoku tak, aby sme dopredu vedeli hodnotu tejto permutačnej matice. Pre každú iteráciu algoritmu sme každému riadku permutačnej matice nastavili špeciálnu hodnotu. Riadok teda obsahoval jednu jednotku a všetky ostatné hodnoty (stĺpce) boli nulové. Počet riadkov permutačnej matice bol 2048 (dané McEliece algoritmom). V každom i-tom riadku ($i \in < 1; 2048 >$) je zmenená pozícia jednotkového bitu. Cieľom útoku je zmerať všetky vektory spotreby dané i-

tou permutačnou maticou (2048 vektorov spotreby). Kvôli spoľahlivosti merania potrebujeme zmerať aspoň 10 vektorov spotreby pre i-tu permutačnú maticu, t.j. potrebujeme zmerať 10 × 2048 vektorov spotreby. Tých desať rovnakých vektorov spriemerujeme a tak dostaneme jeden vektor spotreby pre jednu kombináciu permutačnej matice, čiže 2048 vektorov, v ktorých sme čiastočne eliminovali šum. Zo získaných vektorov dokážeme zrekonštruovať hodnoty permutačnej matice. To je možné pomocou korelačnej analýzy. Korelačná analýza zisťuje podobnosť skúmaných signálov vzhľadom na jeden referenčný signál. Vo vektore spotreby (pre ľubovoľné nastavenie matice $\boldsymbol{P}$) je vždy iba jedna oblasť (v čase), ktorá má zvýšenú spotrebu (nárast napätia). McEliece algoritmus ale túto „jednotku" náhodne posunie. Úlohou korelačnej analýzy je nájsť jednotlivé časové posunutia oproti referenčnému vektoru spotreby (ľubovoľný z 2048 vektorov) a dať do súvisu tento časový posun so skutočnou hodnotou permutačnej matice. Takto budeme v budúcnosti vedieť rozpoznať hocijaký vektor spotreby a priradiť k nej hodnotu permutačnej matice. Ako náhle budeme vedieť všetky riadky dokážeme zrekonštruovať aj zašifrovaný text, resp. výrazne urýchliť proces dešifrovania správy.

Na vyhodnocovanie meraní postranných kanálov sme použili program MATLAB. Analyzovanie merania prebiehalo v niekoľkých krokoch:

1. Načítanie nameraných priebehov.
2. Preprocessing.
3. Analýza jednotlivých vektorov spotreby.

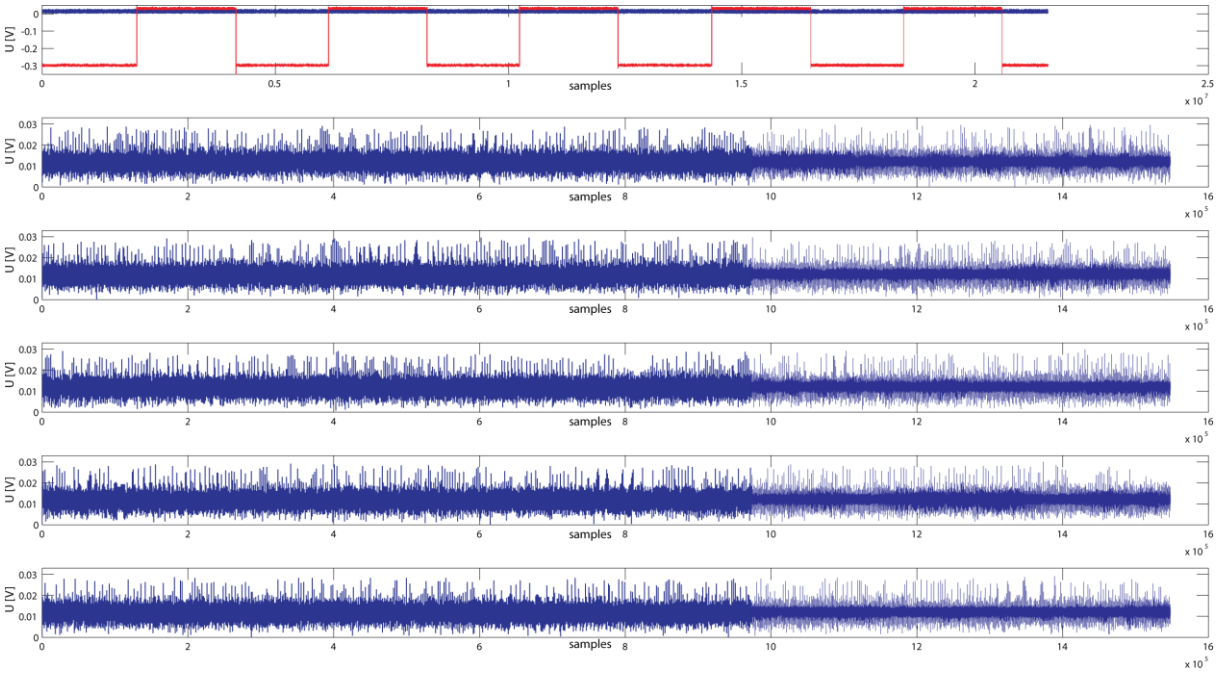
2.2.5.1 Načítanie nameraných vektorov spotreby

Táto operácia je pomerne jednoduchá, pomocou MATLAB funkcie načítame súbor uložený osciloskopom (*.mat). Implementované je to tak, že stačí zvoliť priečinok v ktorom sa tieto súbory nachádzajú. Treba spomenúť, že tieto súbory musia byť z toho istého merania, t.j. s rovnakým nastavením ofsetu, významom meracích kanálov a hlavne rovnakým nastavením časovej základne osciloskopu a vzorkovacej frekvencie.

2.2.5.2 Preprocessing

Tento proces je druhý najzdĺhavejší. Vykonáva sa tu selekcia vektorov spotreby pre jednotlivé iterácie, jednoduché filtrovanie a automatické odstraňovanie chyb v meraní (prechodové javy). Selekcia jednotlivých vektorov je potrebná kvôli zvolenej metodike analýzy postranných kanálov. Zaujímajú nás iba tie časové úseky v nameranom signáli (obr. 2.14– modré priebehy), v ktorých prebieha dešifrovanie. Tieto úseky sú dané trigger signálom (obr. 2.14 – červený priebeh). Prvá fáza preprocesingu teda pozostáva zo získania modrých úsekov s rovnakým nastavením permutačnej matice. V našom konkrétnom príklade sme merali 10 rovnakých vektorov spotreby, ktoré sme priemerovali a dostali sme 1 vektor pre jednu kombináciu permutačnej matice. Na obr. 2.14 vidíme iba 5. Je to dané pamäťou osciloskopu, ktorý už nemal miesto na uloženie ďalších piatich vektorov. Museli sme teda priebežne zastavovať algoritmus dešifrovania a uložiť namerané dáta na meracom počítači. Tento proces sa dá v budúcnosti ľahko automatizovať, nakoľko je možné pomocou MATLABu

riadiť skoro všetky funkcie pripojeného osciloskopu. Je potrebné uviesť, že meranie bolo vykonané pri vzorkovacej frekvencii 625 MS/s.

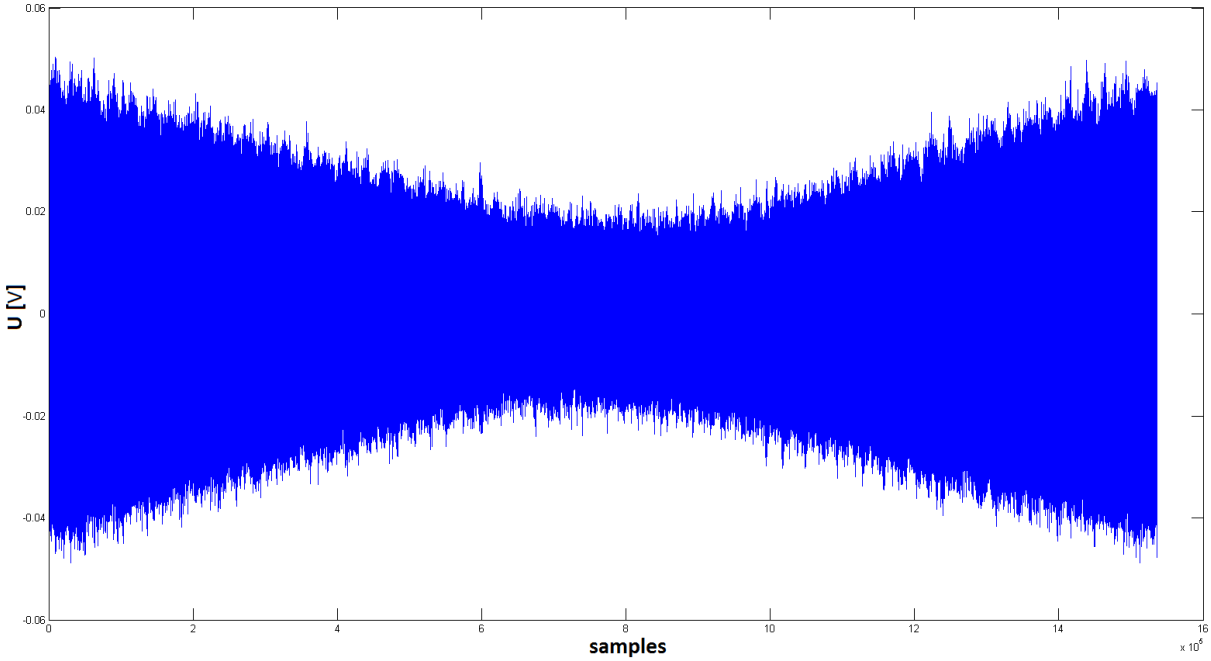


Obr. 2.14 Záznam z merania piatich vektorov spotreby spolu s trigger signálom

Odstraňovanie chybne nameraných signálov bolo pomerne nenáročné. Odstraňovali sa hlavne zákmity, ktoré boli spôsobené jednak trigger signálom, alebo aj iným zariadením na vývojovej doske, ktoré bolo napájané z tej istej napájacej vetvy ako SoC čip. Ďalej sa odstraňovali hodnoty, ktoré presiahli nastavený merací rozsah osciloskopu. Takéto hodnoty osciloskop vyhodnotil ako „inf“ (infinity). Odstraňovanie bolo vykonané nahradením tejto vzorky priemernou hodnotou vektoru spotreby počas piatich períód trigger signálu. Toto číslo je určené dynamicky podľa veľkosti pamäte osciloskopu. V našom prípade (pre danú hodnotu vzorkovacej frekvencie) to bolo 5. Eliminovanie zákmitov bolo nastavované empiricky na základe empiricky zvolenej miere odvodenej zo štandardnej odchýlky. Akonáhle bola zaznamenaná vzorka s väčšou/menšou hodnotou, ako je empiricky zvolený násobok štandardnej odchýlky, bola táto vzorka prepísaná priemernou hodnotou (tak ako pri odstraňovaní presahov). Odstraňovanie takýchto chýb je veľmi dôležité pretože presne takéto chyby by mohli spôsobiť nesprávne zistený časový posun v korelačnej analýze.

2.2.5.3 Analýza jednotlivých vektorov spotreby

Ako už bolo spomenuté vyššie, všetky vektory spotreby (2048 vektorov) by mali byť podrobené korelačnej analýze. V tomto dokumente uvedieme (pre názornosť) výsledok len pre dva namerané vektory spotreby podľa dvoch riadkov permutačnej matice. Prvý riadok má jednotku na prvej pozícii (ostatné sú nuly) a druhý má jednotku na druhej pozícii. Výsledok korelácie je možné vidieť na obr. 2.15.



Obr. 2.15 Výsledok korelačnej analýzy medzi dvoma vektormi spotreby

Z korelácie, podľa tohto obrázka, nie sme schopní identifikovať relatívnu pozíciu jednotkového bitu medzi týmito dvoma nameranými vektormi spotreby. Príčin môže byť veľa, no v tomto prípade sú príčinou: nedostatočná vzorkovacia frekvencia osciloskopu, malá šírka pásma osciloskopu a meracích sond, nevhodný merací bod a pomalý nábeh trigger signálu generovaný FPGA časťou SoC čipu. Preto bolo potrebné modifikovať zapojenie aj implementáciu McEliece kryptosystému.

2.2.6 Modifikácia merania postranných kanálov

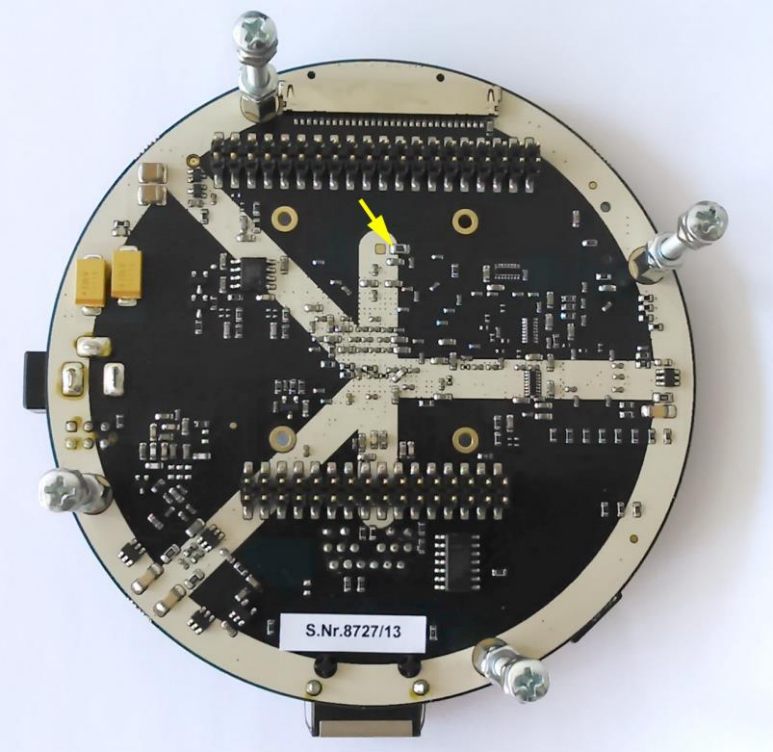
Prvá modifikácia (najdôležitejšia) bola zmena meracieho prístroja, konkrétne zmena osciloskopu PicoScope 6403A za osciloskop s väčšou šírkou pásma. Keďže procesor, ktorý vykonáva dešifrovanie, je taktovaný na frekvencii 800 MHz, osciloskop by mal mať minimálne 2-krát väčšiu vzorkovaciu frekvenciu. Vyplýva to zo vzorkovacej Shannon-Kotelnikovej teorémy [7], ktorá definuje vzťah medzi vzorkovacou frekvenciou a maximálnou frekvenciou signálu, ktorú dokážeme správne zrekonštruovať:

$$f_v > 2f_{max},$$

kde f_v je vzorkovacia frekvencia a f_{max} je maximálna frekvencia, ktorá sa vyskytuje v signáli. Dôsledok tohto teorému je, že ak by sme použili nižšiu vzorkovaciu frekvenciu ako je $2f_{max}$, zrekonštruovaný signál by sa výrazne líšil od pôvodného signálu. To je pre naše účely nevyhovujúce, pretože vtedy strácame informáciu o amplitúde práve na frekvenciách (vyšších harmonických), kde sa vykonávajú jednotlivé inštrukcie. Použitý osciloskop PicoScope 6403A ma šírku pásma 350 MHz, t.j. jeho vnútorné obvody frekvenčne obmedzia meraný signál na 350 MHz. Preto nie sme schopný merať signáli s frekvenciou nad 350 MHz aj keď nastavíme vysokú vzorkovaciu frekvenciu, napr. 2.5 GS/s.Pre naše potreby by sme

potrebovali osciloskop so šírkou pásma minimálne 2 × 800 MHz, t.j. 1.6GHz a vzorkovacou frekvenciou minimálne 1.6 GS/s.

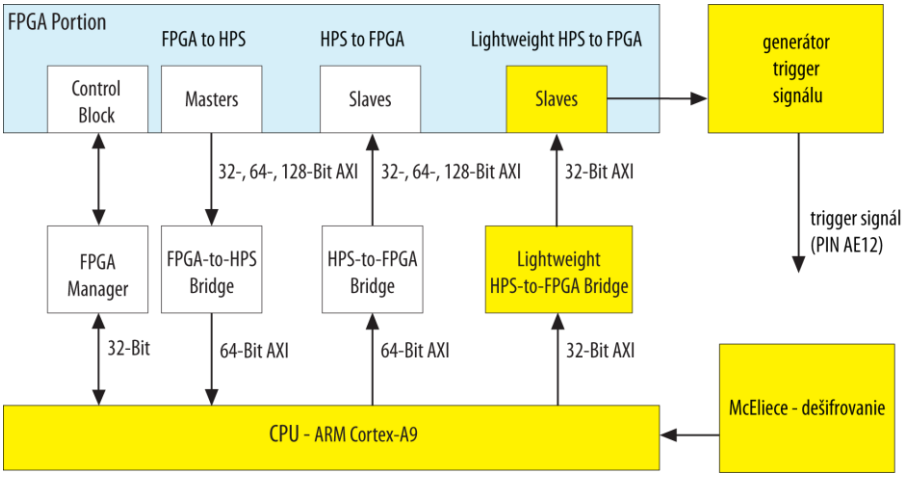
Ďalším faktorom, ktorý má veľký vplyv na úspešné zrekonštruovanie tajného parametra, je poloha meracej sondy osciloskopu. Najideálnejším miestom je mať meraciu sondu, pokiaľ je to možné, čo najbližšie k púzdru procesora a elektricky sa napojiť na jeho napájaciu vetvu. Niekedy je možné upraviť zariadenie tak, aby sme zvýšili kvalitu meraného signálu. Najjednoduchšia úprava spočíva v zaradení predradného rezistora do napájacej vetvy skúmaného čipu. Hodnota tohto rezistora sa pohybuje rádovo v jednotkách ohmov. Prúd, ktorý prechádza z napájacieho zdroja do čipu vytvorí na rezistore pomerne veľký úbytok napätia (radovo 10-ky až 100-ky mV). Spomínaný úbytok napätia ešte nespôsobí zmenu funkcionality (alebo úplné znefunkčnenie) zariadenia, no zároveň zvýši zisk signálu vhodného na vykonanie diferenciálnej analýzy spotreby. Vtedy je dokonca výhodné použiť na meranie diferenciálnu osciloskopickú sondu. Sú rôzne kombinácie, ako zapojiť tento predradný rezistor. V tomto prípade nebolo možné uplatniť túto modifikáciu, pretože hustota osadenia súčiastok bola príliš veľká. Navyše, bola doska plošných spojov viacvrstvová (viac ako 2 vrstvy), čo extrémne komplikovalo umiestnenie meracieho rezistora. Preto sme vôbec neuvažovali o tomto vylepšení a snažili sme sa umiestniť meraciu sondu do napájacej vetvy SoC čipu, čo najbližšie k puzdru (obr. 2.16). Tak sme minimalizovali vplyv filtračných kondenzátorov pripojených do napájacej vetvy čipu.



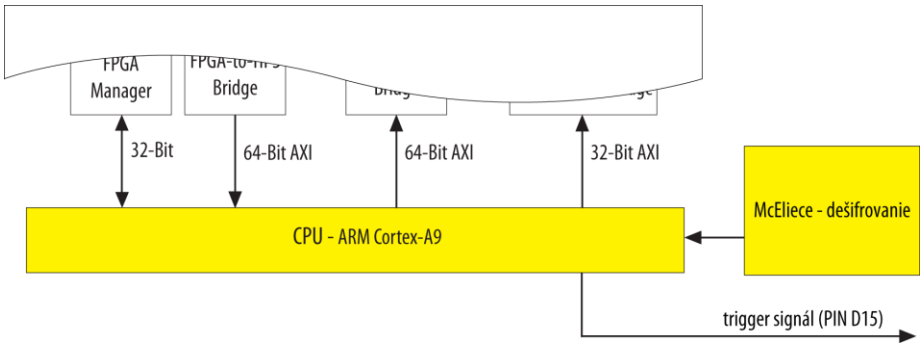
Obr. 2.16 Poloha výhodnejšieho meracieho bodu na meracej doske Socrates

Poslednou modifikáciou bol spôsob generovania synchronizačného „trigger“ signálu. Ako sme už uviedli, tento signál je potrebný na určenie presného časového okamihu vykonávania operácie dešifrovania a správneho zarovnania do vektora spotreby. To je pre

diferenciálnu analýzu spotreby tiež veľmi dôležité. Pôvodne sme generovali tento signál pomerne komplikovaným spôsobom (obr. 2.17 – zvýraznené žltou farbou). Procesor (ARM CortexA9) softvérovo spustil proces, ktorý vyslal kontrolný signál do FPGA časti SoC čipu. Ten následne nastavil výstupný port AE12 (konektor P12) na logickú jednotku (3.3 V). Medzi tým sa procesor vrátil do procesu vykonávania samotného dešifrovania textu. Keď sa dešifrovanie skončilo, spustil sa proces, ktorý nastavil na výstupný port AE12 logickú nulu (0 V). A tento cyklus sa opakoval 10×2048-krát. Problematické bolo, príliš veľké oneskorenie medzi spustením procesu na nastavenie/vynulovanie trigger signálu a samotným nastavením resp. vynulovaním vývodu AE12. Toto oneskorenie bolo rádovo 8 ms. Preto sme zamietli tento spôsob generovania trigger signálu. Na generovanie synchronizačného impulzu sme použili priamo I/O výstupy procesora ARM CortexA9 (obr. 2.18). Dosiahli sme tak veľmi nízkeho oneskorenia a presnej synchronizácie medzi jednotlivými dešifrovaniami. Čas nastavenia bol približne 300ns.



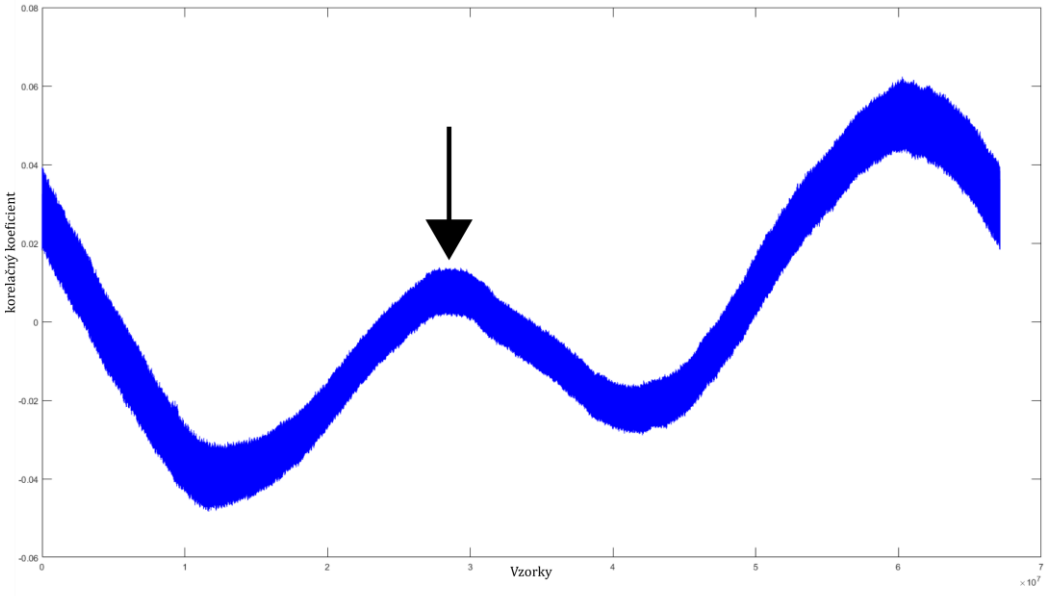
Obr. 2.17 Generovanie trigger signálu pomocou FPGA oddielu SoC čipu (pôvodný spôsob)



Obr. 2.18 Generovanie trigger signálu pomocou CPU v SoC čipe (nový spôsob)

Kvôli tejto zmene bolo samozrejme potrebné opätovne pregenerovať celý HW design SoC čipu a implementovať generovanie trigger signálu priamo prostredníctvom CPU, t.j. rozšíriť pôvodnú implementáciu McEliece kryptosystému o nízkoúrovňové príkazy pracujúce

s I/O vývodmi procesora. Na toto sme použili knižnicu *hwlib*, v ktorej sú tieto operácie implementované. Knižnica je určená len pre architektúru ARM cortex A9. Na obr. 2.19 je zobrazená korelačná analýza pre tie isté hodnoty permutačnej matice $\boldsymbol{P}$,ako v pôvodnej metóde, ale s vykonanými HW zmenami. Šípka identifikuje polohu jednotkového bitu v čase.



Obr. 2.19 Korelácia dvoch vektorov spotreby podľa nových HW a SW modifikácií

Ako merací osciloskop sme použili Rigol DS6104, ktorého parametre sú uvedené v tab. 2.2. Meranie prebehlo pri vzorkovacej frekvencii 2.5 GS/s. Vďaka výrazne väčšej šírke pásma osciloskopu DS6104 a správne zvolenej vzorkovacej frekvencii dokážeme identifikovať polohu jednotkového bitu v procese dešifrovania. Ak porovnáme všetky výsledky korelačnej analýzy (2048 korelácií) s očakávanými polohami jednotkových bitov, dostaneme všetky bity tajného parametra. Ten ešte nepredstavuje priamo tajný (privátny) kľúč, ale pomocou neho by sme už dokázali prelomiť bezpečnosť McEliece šifry s 2048-bitovým privátnym kľúčom.

Tab. 2.2 Použitý merací osciloskop Rigol DS6104 s väčšou šírkou pásma

Počet kanálov	4
Vertikálne rozlíšenie	8 bit
Šírka pásma	1 GHz
Maximálna vzorkovacia frekvencia	5 GS/s (single channel), 2.5 GS/s (dualchannel)
Pamäť	140 MS

2.3 Záver

V rámci výskumného projektu sa nám podarilo vytvoriť pracovisko pre meranie postranných kanálov. Tak isto sa nám podarilo vypracovať metodiku merania postranných kanálov pre McEliece kryptosystém. Pracovisko pozostáva z dvoch embedded zariadení na báze ARM procesora, ktorý je v dvoch prevedeniach. Prvým typom je jednoduchší, ARM

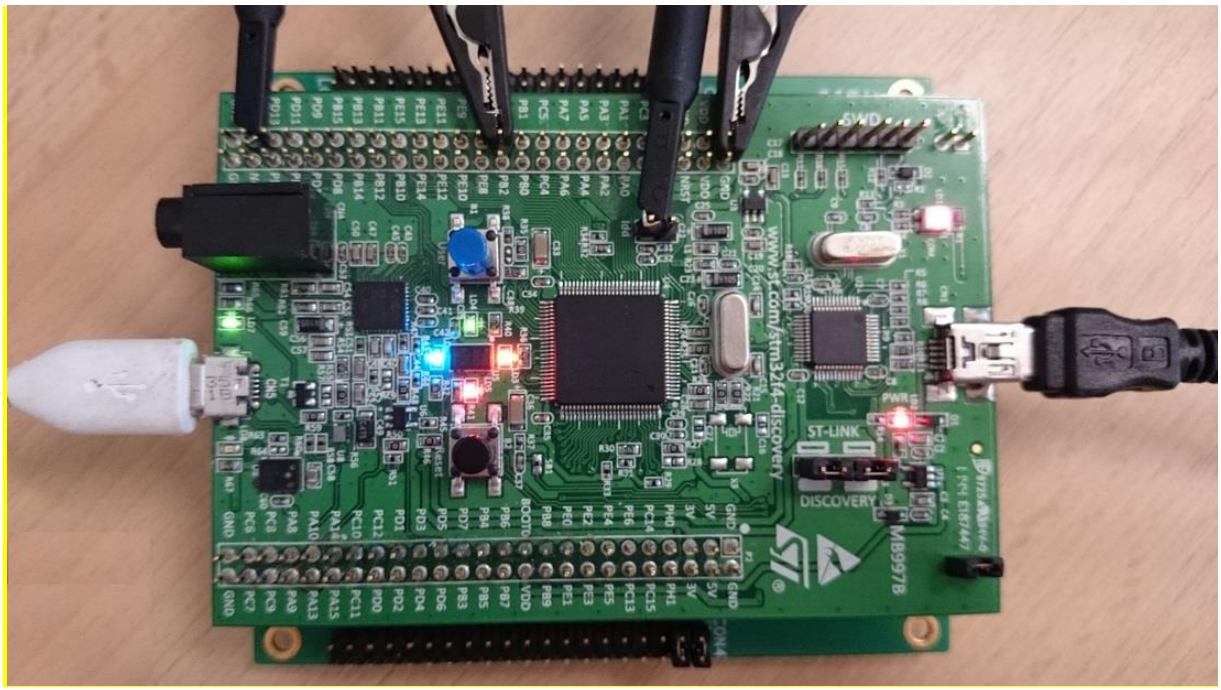
Cortex-M4 (STM32F4, bližšie popísaný v kapitole 3) a druhý je podstatne výkonnejší ARM Cortex-A9. Oba procesory dokážu pracovať s operačným systémom Linux. Pre potreby porovnania úspešnosti útokov sme, v prvom prípade, testovali implementáciu McEliece kryptosystému bez operačného systému a v druhom sme nasadili operačný systém Linux socfpga cyclone5. V prvom prípade sme dokázali upraviť vývojovú dosku v takej miere, aby sme spoľahlivo vedeli odmerať vektory spotreby a tak zistiť časť tajnej informácie (permutačnú maticu). Úprava spočívala vo vložení meracieho rezistora do napájacej vetvy procesora. Úprava bola pomerne ľahko realizovateľná, pretože doska plošných spojov bola len 2-vrstvová. V druhom prípade bola podobná úprava nerealizovateľná. Aj napriek tomu, že sme mali k dispozícii elektrickú schému celej vývojovej dosky, nedokázali sme nájsť vhodné miesto pre merací rezistor. Spomínaná doska mala totižto viac ako 2-vrstvovú dosku plošných spojov. Preto sme merali bez neho a tým sme simulovali reálny útok, keď by nebola možná mechanická úprava. Merací bod bol umiestnený na konektore pre všeobecné využitie, kde sa nachádzali okrem dátových vývodov aj napájacie. Pri takejto konfigurácii nie je možné namerať akúkoľvek informáciu cez postranný kanál. Toto zistenie platí za podmienky, keď použijeme tzv. jednoduchú alebo diferenciálnu analýzu elektrickej spotreby a lacnejšie meracie prístroje. Pre reálny útok by sme potrebovali osciloskop s oveľa vyššou šírkou pásma, pre ARM Cortex-A9,najideálnejšie 2.4 GHz. Prvý osciloskop, so šírkou pásma 350 MHz, orezával vyššie frekvenčné zložky signálu, v ktorom bola prítomná informácia o postrannom kanáli. Preto sme nedokázali z korelačnej analýzy zistiť potrebné informácie. Nezanedbateľný vplyv na správnosť meraní mal aj samotný operačný systém Linux. Pokiaľ beží operačný systém, nemáme kontrolu nad spúšťaním procesov. Tie sú výhradne v jeho réžii. Aplikácia, zabezpečujúca šifrovanie/dešifrovanie môže byť kedykoľvek prerušená (softvérové prerušenie iným procesom) a tak narušiť dôveryhodnosť merania. Ak by malo zariadenie zabezpečovať bezpečnosť a využívalo by na svoju činnosť operačný systém, určite by bol útok prostredníctvom diferenciálnej analýzy sťažený.

3 Využitie mikroprocesora STM32F4 pre kryptografické účely

3.1 Mikroprocesor STM32F4

Mikroprocesor STM32F4 je Discovery kit, ktorý obsahuje ARM MCU(microcontroller unit) s jadrom Cortex M4. Jedná sa o klasický mikroprocesor, ktorý je na jednej nerozoberateľnej doske. Na tejto doske sa nachádza JTAG+SWD rozhranie a aj vlastný čip na testovanie. JTAG rozhranie je ST-LINK/V2, čo znamená, že sa jedná o druhú verziu, preto je potrebné mať aktuálny ovládač, aby sa úspešne podarilo pripojiť toto zariadenie cez USB rozhranie a aby nevznikali žiadne problémy pri prenose. Čip, ktorý vykonáva funkciu rozhrania je STM32F103. V kite je SWD konektor, takže sa dá programovať aj externé MCU.

Základom tohto kitu STM32F4 je mikroprocesor STM32F407VGT6, čo je Cortex M4 MCU s 1 MB Flash, 192 kB SRAM a je zapuzdrený do LQFP100. To znamená, že sa dá použiť aj na pripojenie externej pamäte. Napájanie je riešené z USB, ale dá sa pripojiť aj externé 5V napájanie.



Obr 3.1 Karta STM32F4 s mikroprocesorom STM32F407VGT s rozsvietenými LED diódami, pripojenými sondami na meranie napätia a pripojenými USB káblami zabezpečujúcimi napájanie (čierne miniUSB) a I/O komunikáciu s PC (biele mikroUSB).

3.2 Programovanie mikroprocesora STM32F4

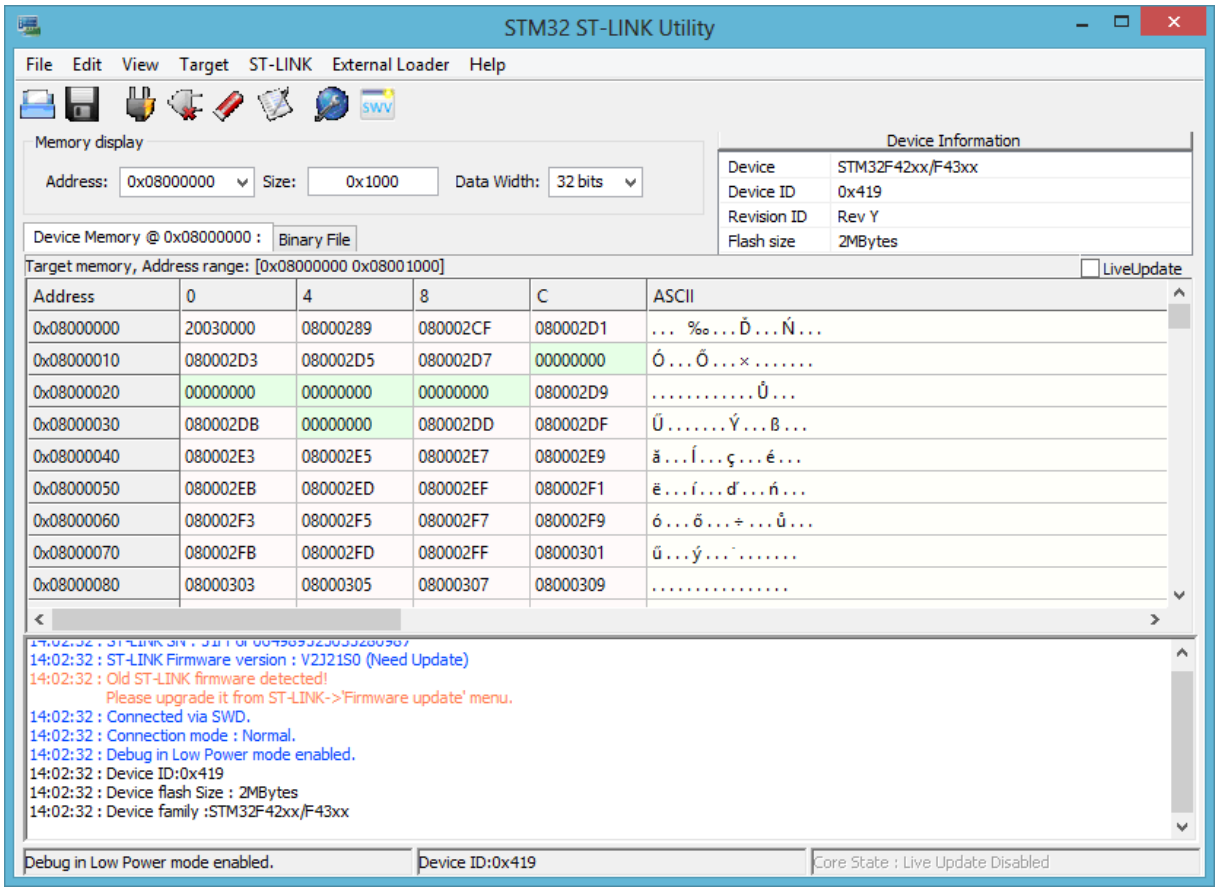
Pre prácu/programovanie zariadenia potrebujeme nasledovné softvérové vybavenie:

1) Vývojové prostredie (IDE – Integrated Development Environment) – my sme zvolili voľne dostupné prostredie EmBlocks, ktoré podporuje debuggovanie zariadenia cez

DBG server s výpismi na konzolu, link: http://www.emblocks.org a navyše obsahuje vlastný ARM GCC kompilátor.

2) Utilitu STM32ST-LINK, ktorá obsahuje drivere, potrebné pre komunikáciu so zariadením, link: http://www.st.com/web/en/catalog/tools/PF258168
3) (voliteľné) Program STM32Cube, slúžiaci pre grafickú konfiguráciu zariadenia – dajú sa v ňom jednoducho nastaviť hodnoty registrov zariadenia, prípadne preddefinované hodnoty rôznych pinov – program vygeneruje zdrojový kód v jazyku C, ktorý sa priloží ku kompilovanému programu, link: http://www.st.com/web/catalog/tools/FM146/CL2167/SC2004/PF259243
4) (voliteľné) Clock Configuration Tool utilita, slúžiaca pre jednoduché nastavenie vnútorných hodín zariadenia – jedná sa o excelovský súbor, v ktorom sa jednoducho nastaví taktovanie zariadenia. Následne súbor vygeneruje zdrojový kód v jazyku C, ktorý sa priloží ku kompilovanému programu, link: http://www.st.com/web/en/catalog/tools/PF257927

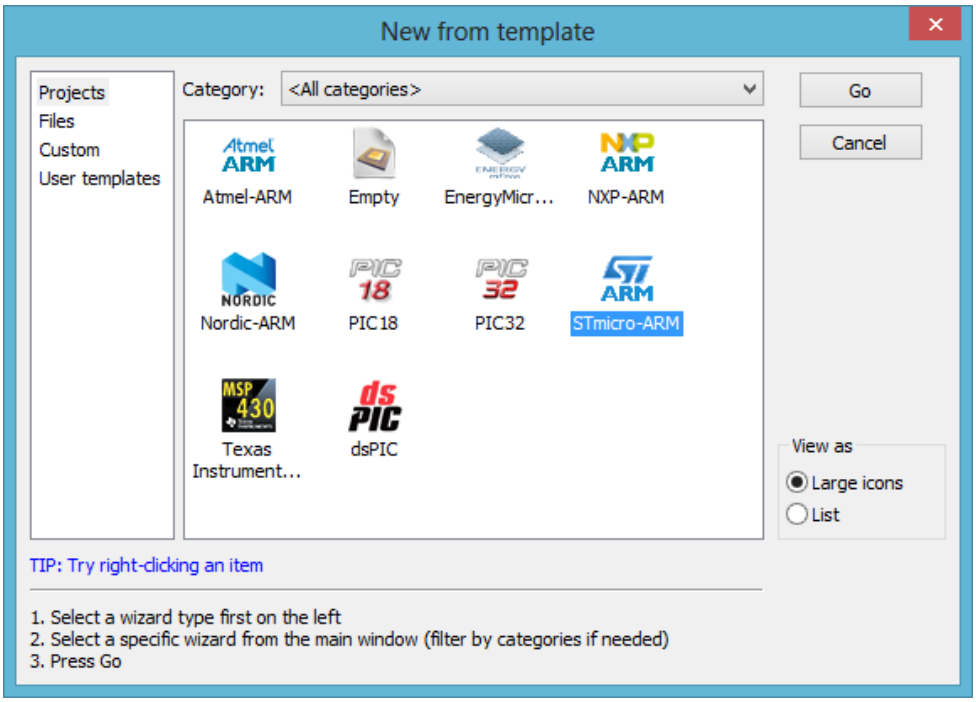
Na prvotné pripojenie k zariadeniu, zistenie informácii o ňom, vymazanie FLASH pamäte, modifikácia a zobrazenie pamäte, či samotné programovanie zariadenia nahratím skompilovaného kódu, slúži spomínaná utilita od výrobcu, s názvom „STM32 ST- LINK Utility“.



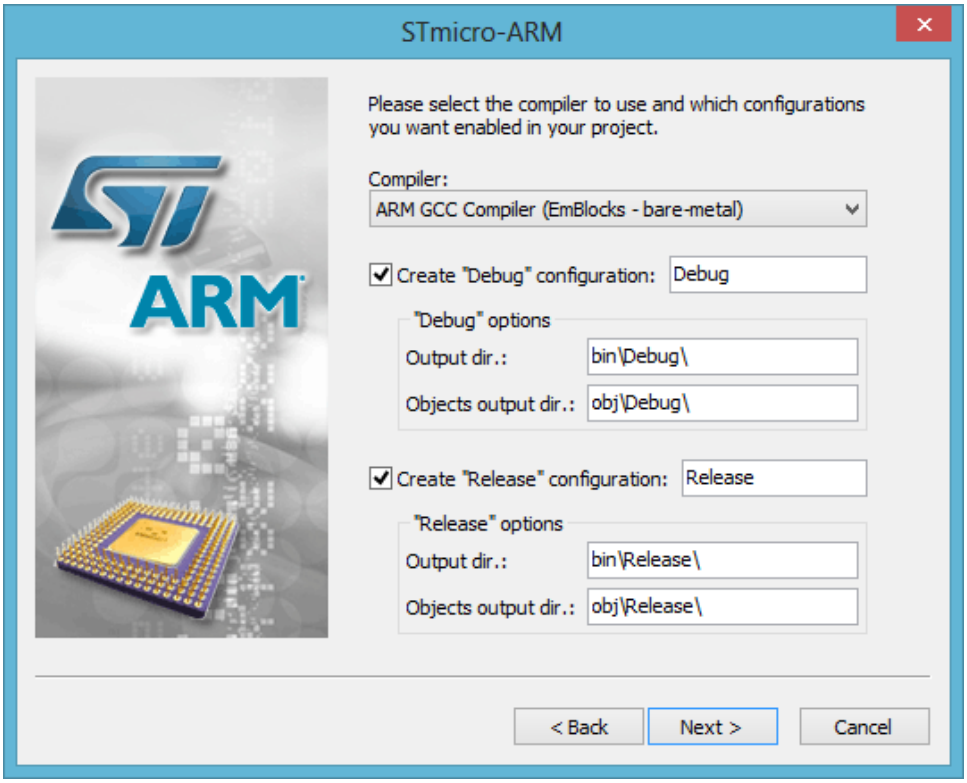
Na obrázku vyššie môžeme vidieť užívateľské rozhranie utility s pripojeným zariadením, to sme jednoducho pripojili do USB a po stlačení tlačidla „Connect to the target“ program automaticky rozpoznal zariadenie. Teraz môžeme využívať funkcie utility a pracovať so zariadením.

Samotný program píšeme v EmBlocks IDE (Integrated Development Environment), ktorý je voľne dostupný. (link: http://www.emblocks.org/web/downloads-main). Ten vo svojom inštalačnom balíčku obsahuje GDB server, ktorý slúži na vzdialené ladenie programu priamo na našom zariadení.

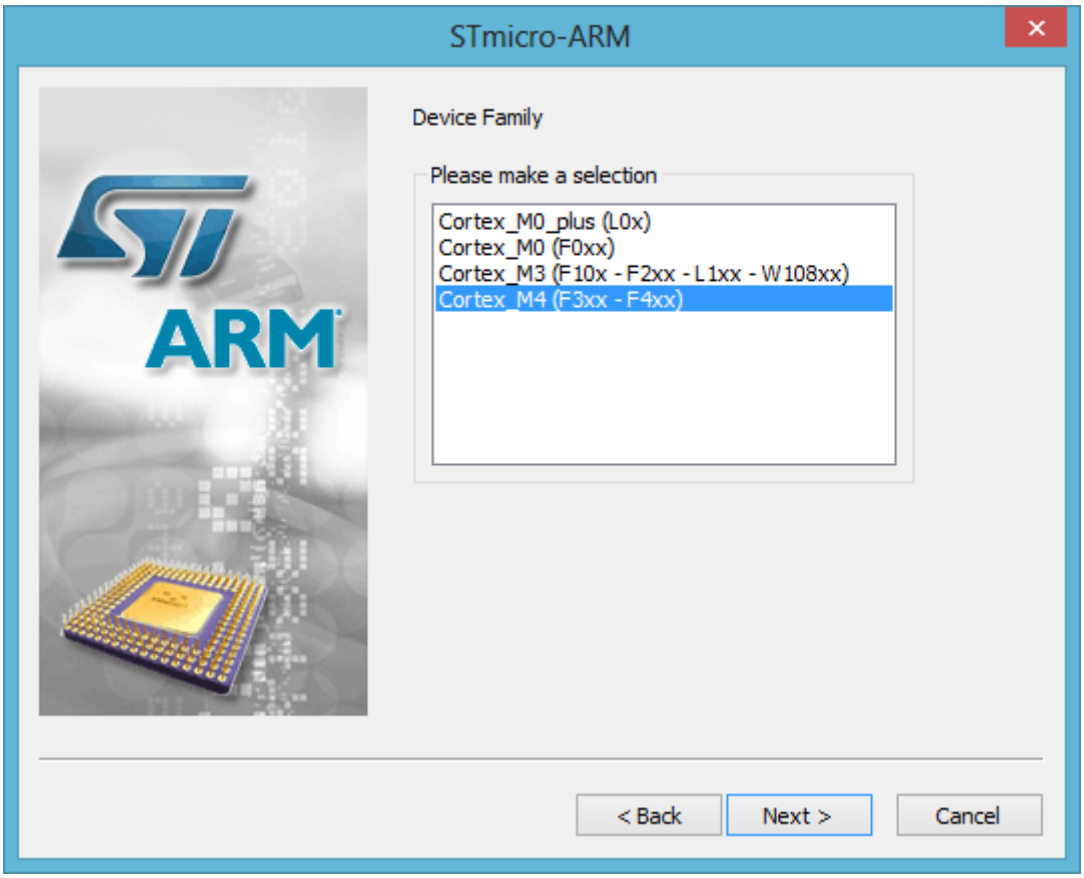
Vytvorenie projektu je veľmi intuitívne a všetky potrebné nastavenia nám uľahčí „wizard“, ktorý vyberáme v prvom kroku po kliknutí na File->New->Project.

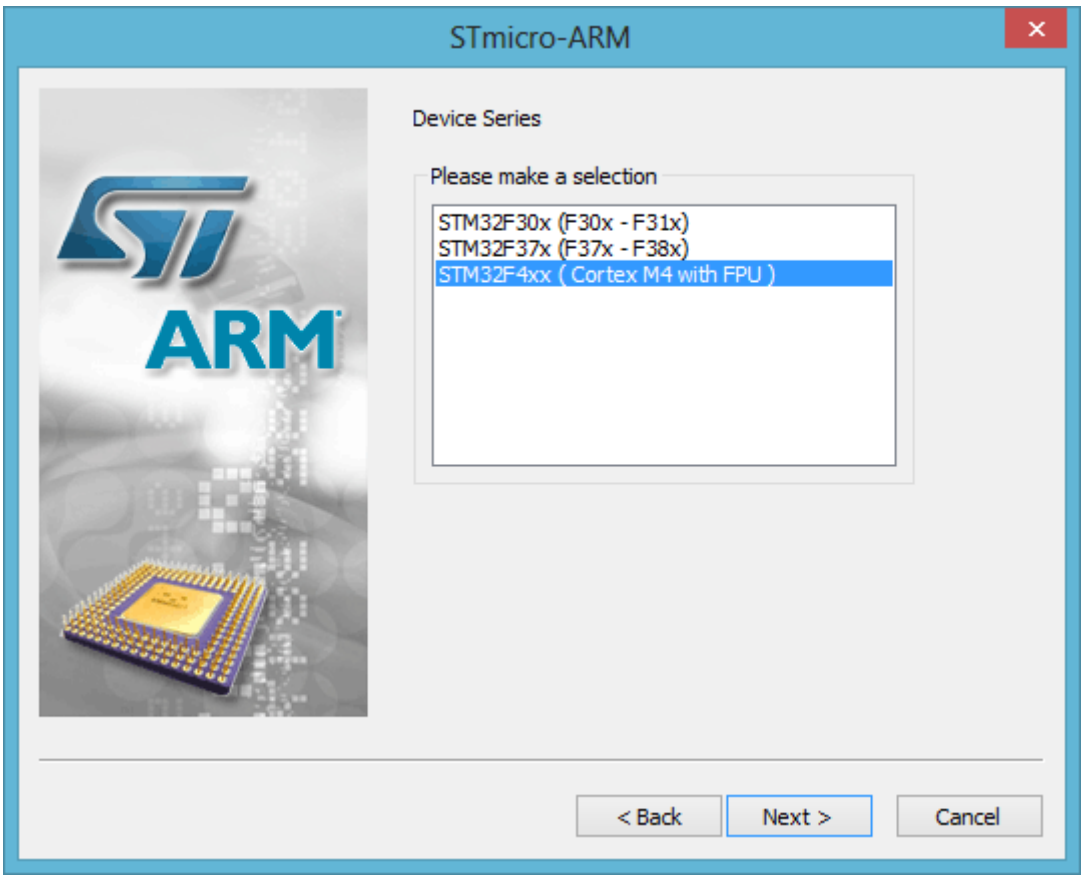


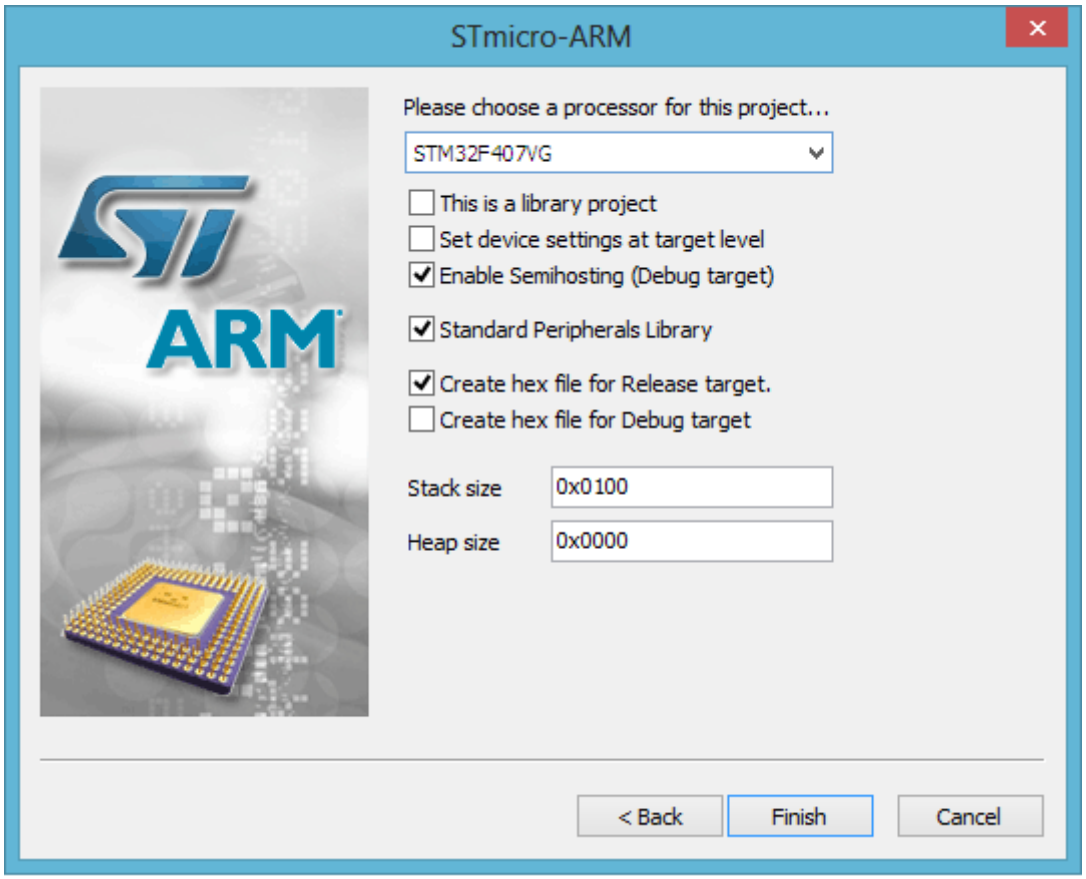
V našom prípade sme vybrali STmicro-ARM wizard. Po výbere nás privítal zvolený sprievodca a ďalším krokom je výber názvu projektu a jeho umiestnenia. Nasleduje voľba kompilátora (ponecháme základný kompilátor, ktorý využíva EmBlocks – ARM GCC Compiler EmBlocks – bare-metal) a možnosť zvoliť, ktorý typ konfigurácie sa má po zbuildovaní projektu vytvoriť a na aké umiestnenie. Je potrebné nastaviť semi-hosting, aby sme umožnili v debug móde vypisovanie rôznych užitočných informácií na konzolu.



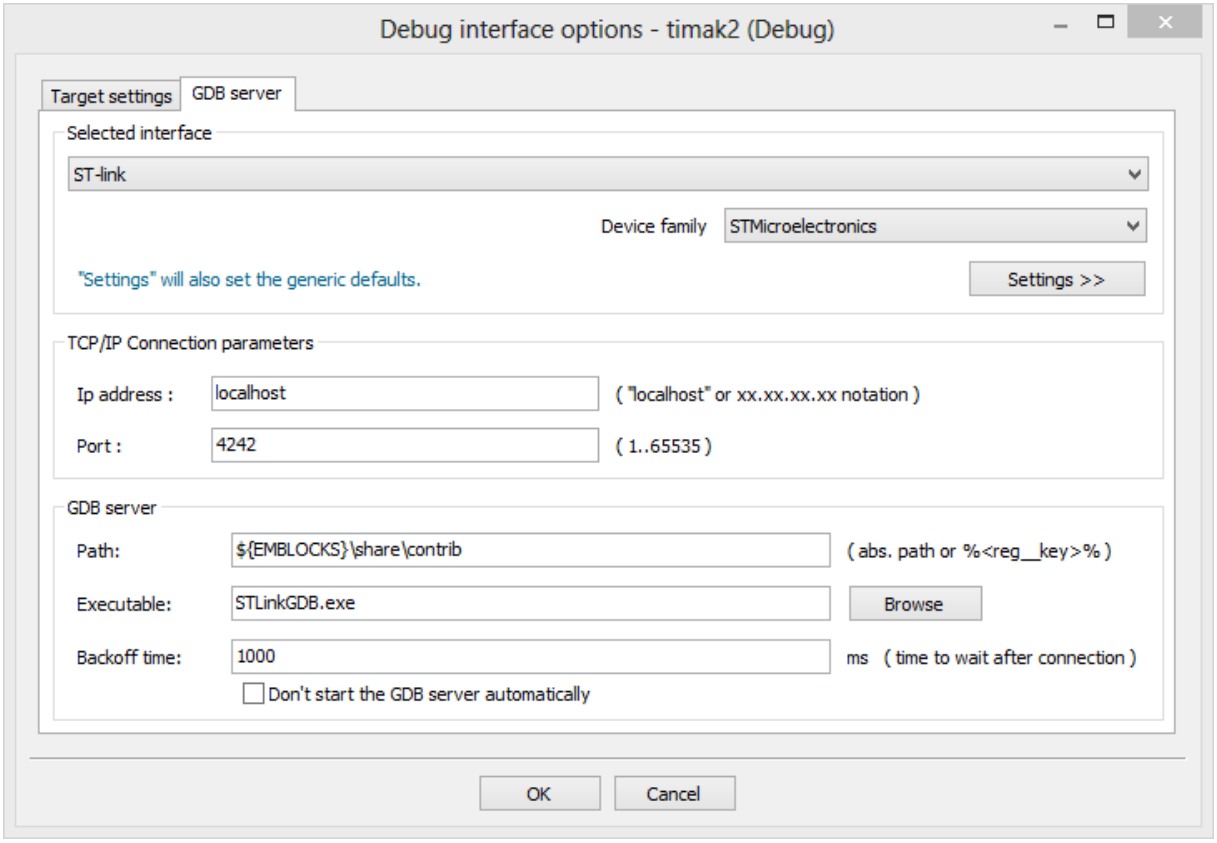
Potom nasledujú kroky, v ktorých vyberáme typ nášho zariadenia.

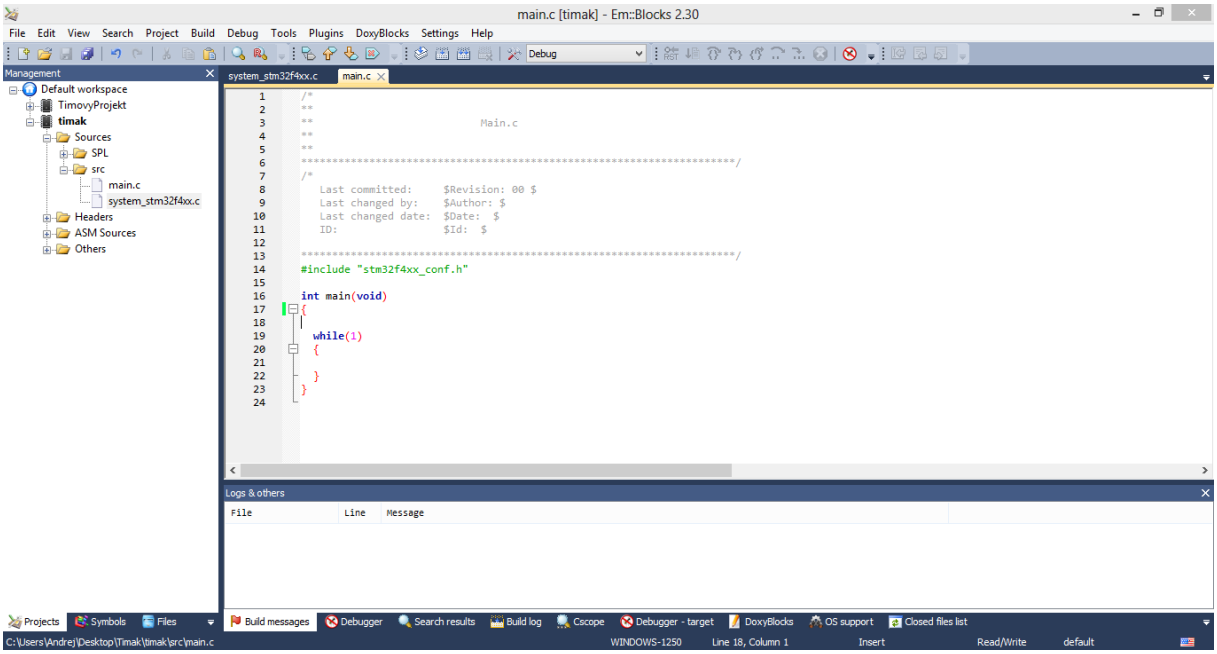

STmicro-ARM
Device Family
Please make a selection
Cortex_M0_plus (L0x)
Cortex_M0 (F0xx)
Cortex_M3 (F10x - F2xx - L1xx - W108xx)
Cortex_M4 (F3xx - F4xx)
< Back
Next >
Cancel


STmicro-ARM
Device Series
Please make a selection
STM32F30x (F30x - F31x)
STM32F37x (F37x - F38x)
STM32F4xx (Cortex M4 with FPU)
< Back
Next >
Cancel

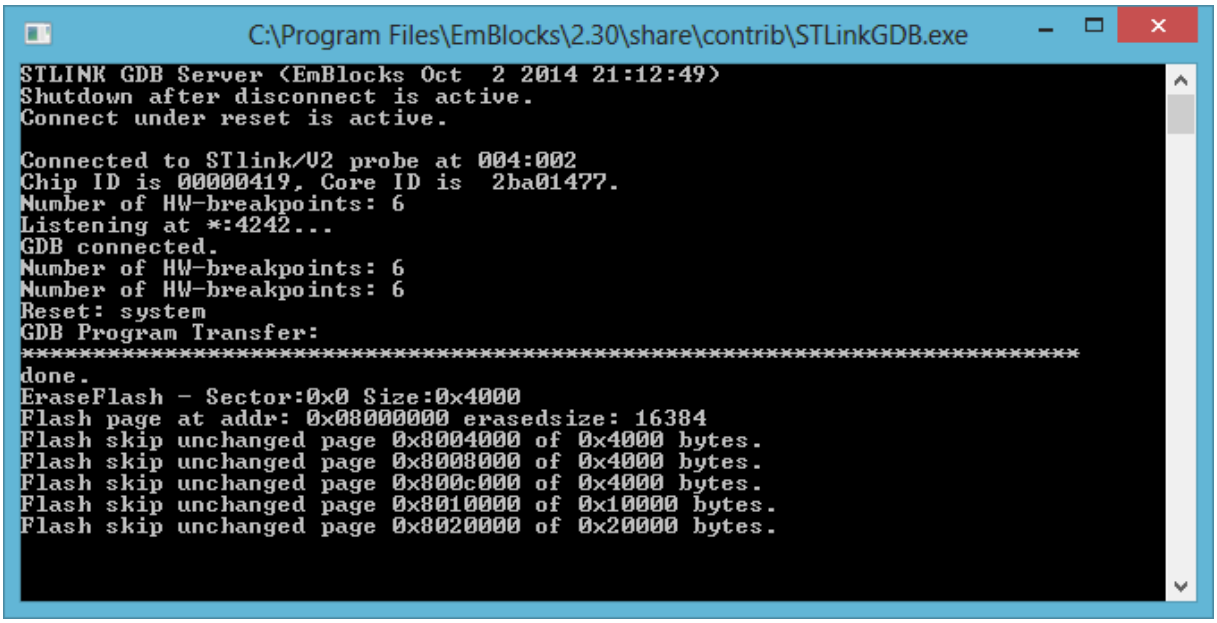


Po týchto krokoch už máme vytvorený prvý projekt, ale ešte musíme nastaviť hore spomínaný GDB server, resp. potvrdiť nastavenie, keďže EmBlocks ho nastaví automaticky.





Teraz môžeme písať samotný program. Ak chceme program nahrať na zariadenie a ladiť ho, stlačíme tlačidlo Start/Stop Debug Session. Program sa začne buildovať a keď je bez chyby začne sa spúšťať GDB server, ktorý rozpozná pripojené zariadenie a začne doň nahrávať program.



Po celom tomto procese, sme schopní ladiť prvý program. Môžeme pridávať breakpointy tak, ako sme zvyknutí z iných vývojových prostredí. Program môžeme krokovať, prípadne rovno spustiť jeho vykonávanie (F5).

3.3 Testovanie výkonnostných parametrov prúdových šifier na zariadení STM32F4

Rozhodli sme sa otestovať výpočtové možnosti zariadenia implementáciou prúdových šifier z projektu eSTREAM. Projekt eSTREAM bol organizovaný sieťou EU ECRYPT a jeho úlohou bolo schválenie nových bezpečných prúdových šifier pre verejné použitie.

Výsledkom projektu bolo schválenie 7 prúdových šifier pre verejné použitie. Šifry sú rozdelené do 2 portfólií, podľa toho, či sú určené pre softvérovú implementáciu, alebo hardvérovú implementáciu (FPGA, ASIC).

Tab. 3.1 Prehľad implementovaných prúdových šifier

Hardvérovo-orientované prúdové šifry	Softvérovo-orientované prúdové šifry
Grain v1	HC-128
Mickey 2.0	Rabbit
Trivium	Salsa20/12
	SOSEMANUK

Zo stránky projektu eSTREAM (http://www.ecrypt.eu.org/stream/) sme stiahli referenčné zdrojové kódy hore uvedených prúdových šifier, implementovali sme ich na zariadenie

STM32F407VG a merali sme ich výkonnostné parametre v 4 rôznych scenároch, ktoré sme prevzali z projektu eSTREAM.

Sledované scenáre boli nasledovné:

- Šifrovanie dlhého prúdu bitov s 1 kľúčom a 1 inicializačným vektorom (22 * 4096 bajtov)
- Šifrovanie kratších paketov o veľkosti 40 bajtov (celkovo 350 paketov, každý paket bol šifrovaný s iným inicializačným vektorom, na 35 paketov pripadal 1 kľúč).
- Šifrovanie stredne dlhých paketov o veľkosti 576 bajtov (celkovo 120 paketov, každý paket bol šifrovaný s iným inicializačným vektorom, na 12 paketov pripadal 1 kľúč).
- Šifrovanie dlhých paketov o veľkosti 1500 bajtov (celkovo 50 paketov, každý paket bol šifrovaný s iným inicializačným vektorom, bol použitý len 1 šifrovací kľúč).
- Vo všetkých prípadoch sme sledovali výkonnostné parametre len z hľadiska šifrovania, t.j. inicializačnú fázu šifry (nahrávanie kľúča) sme nebrali do úvahy.

Sledované výkonnostné parametre boli:

- Počet cyklov/taktov procesora na 1 bajt: cycles/B.
- Dátová priepustnosť (rýchlosť) šifrovania/generovania prúdového kľúča v megabitoch za sekundu: Mbps.

Výsledky uvádzame v tabuľkách.

Tab. 3.2 Šifrovanie dlhého prúdu bitov

Meno šifry	Počet bitov kľúča	Počet bitov IV	Cycles/B	Mbps
HW				
Grain v1	80	64	33466	0.04
Mickey 2.0	80	80	5036	0.265
Trivium	80	80	124	10.81
SW				
HC-128	128	128	58	22.78
Rabbit	128	64	96	13.72
Salsa20	256	64	136	9.81
SOSEMANUK	256	128	114	11.69

Tab. 3.3 Šifrovanie krátkych paketov

Meno šifry	Počet bitov kľúča	Počet bitov IV	Cycles/B	Mbps
HW				
Grain v1	80	64	50404	0.02
Mickey 2.0	80	80	9166	0.15
Trivium	80	80	386	3.46
SW				
HC-128	128	128	8964	0.15
Rabbit	128	64	286	4.69
Salsa20	256	64	206	6.52
SOSEMANUK	256	128	246	5.43

Tab.3.4 Šifrovanie stredne dlhých paketov

Meno šifry	Počet bitov kľúča	Počet bitov IV	Cycles/B	Mbps
HW				
Grain v1	80	64	34634	0.03
Mickey 2.0	80	80	5340	0.25
Trivium	80	80	142	9.44
SW				
HC-128	128	128	670	2.00
Rabbit	128	64	108	12.32
Salsa20	256	64	138	9.76
SOSEMANUK	256	128	124	10.67

Tab.3.5 Šifrovanie dlhých paketov

Meno šifry	Počet bitov kľúča	Počet bitov IV	Cycles/B	Mbps
HW				
Grain v1	80	64	33930	0.04
Mickey 2.0	80	80	5164	0.26
Trivium	80	80	130	10.25
SW				
HC-128	128	128	294	4.55
Rabbit	128	64	102	13.07
Salsa20	256	64	140	9.63
SOSEMANUK	256	128	116	11.52

Z vykonaných experimentov vyplýva, že na zariadení STM32F4 s procesorom STM32F407VG dosahuje najlepšie výsledky pri šifrovaní dlhého prúdu bitov prúdová šifra HC-128 (22.8 Mbps, 58 cyklov/bajt).

Pre šifrovanie kratších paketov sa ako najrýchlejšia šifra ukázala šifra Salsa20 (6.5 Mbps, 206 cyklov/bajt). S ich rastúcou veľkosťou sa však na prvé miesto dostala šifra Rabbit (12 – 13 Mpbs, 102 – 108 cyklov/bajt).

Je zaujímavé, že hoci šifra Trivium je primárne určená pre implementáciu v hárdvéri (čo vyplýva aj z jej dizajnu, keďže sa skladá z „HW-friendly“ prvkov, ako nelineárny spätnoväzobný posuvný register, ktorého výstupom je len XOR 3 bitov, dosahuje veľmi dobré výsledky aj v softvérovej implementácii.

Naopak, šifra Grain v1 dosiahla pomerne zlé výsledky aj z hľadiska softvérovej implementácie. To mohlo byť spôsobené však aj tým, že jej zdrojový kód dostupný na web stránke projektu nemusel byť optimalizovaný pre vyššiu rýchlosť (zmienku o tomto fakte sme však nenašli).

3.4 Implementácia McEliecovho kryptosystému na zariadenie STM32F4 a meranie postranných kanálov

Samotný McEliecov kryptosystém je popísaný v kapitole 2.2.3 tejto záverečnej vedeckej práce. V tejto časti preto uvádzame len stručný popis vykonaného útoku a príslušné výsledky.

3.4.1 Úvod do útoku analýzy spotreby elektrickej energie

Útok odberovej analýzy využíva skutočnosť, že spustenie kryptografického algoritmu na fyzickom zariadení nevracia informácie o spracovaných dátach a spustených operáciách na základe okamžitej spotreby energie. Meranie a vyhodnotenie spotreby energie na kryptografickom zariadení umožňuje využitie informácií v kombinácii so znalosťami o otvorenom, alebo zašifrovanom texte, za účelom získania tajného kľúča. Vzhľadom na to že výsledky výpočtov sú sériovo spracované (špeciálne v 8, 16 a 32bit architektúrach), dá sa použiť metóda rozdeľuj a panuj. Napr. tajný kľuč môže byť obnovený bajt po bajte.

Útok jednoduchou spotrebou analýzy (SPA) sa spolieha na vizuálnu kontrolu priebehov napätia, ktoré môžu byť napríklad namerané z vloženého mikroprocesora čipovej karty. Cieľom SPA je odhaliť podrobnosti o spustení programu softvérovej implementácie, ako aj detekciu podmienených vetiev, v závislosti na tajnej informácii. Obnovenie privátneho kľúča RSA bit po bite pomocou SPA square and multiply algoritmu a odhalenie KeeLog tajného kľúča pomocou SPA na softvérovej implementácii dešifrovacím algoritmom, patria medzi najsilnejšie praktické príklady využitia SPA v reálnom svete. Na rozdiel od SPA, zložitá odberová analýza DPA využíva štatistické metódy a vyhodnocuje niekoľko odberových stôp. DPA nevyžaduje žiadnu znalosť o konkrétnej implementácii šifry a môže byť použitá na väčšinu nechránených blackbox implementácií. Vzhľadom na stredné hodnoty, ktoré sú závislé na kľúčovej hypotéze, stopy sú korelované k odhadovaným odberovým hodnotám a potom korelačné koeficienty naznačujú najviac pravdepodobnú hypotézu medzi všetkými možnými hypotézami. Aby bolo možné vykonať koreláciu na DPA, spotreba energie zariadenia počas útoku musí byť odhadnutá. Napájací model by mal byť definovaný podľa charakteristík atakovaného zariadenia napríklad Hammigovou váhou spracovaných dát pre mikroprocesor, vzhľadom na existenciu prednapájanej zbernice v mikroprocesore. V prípade zlej kvality získanej spotreby, napríklad kvôli zlému nastaveniu, zlému nastaveniu merania, alebo lacnému vybaveniu, môže byť priemerovanie aplikované na dešifrovanie rovnakého šifrovaného textu.

3.4.2 Model útočníka

V útoku je uvažovaný model protivníka:

Útočník pozná verejné G^*,t. Taktiež pozná implementačnú platformu (napr. typ použitého mikroprocesora), implementačný profil napr. zdrojový kód dešifrovanej schémy (samozrejme s výnimkou obsahu pamäte, vopred vypočítaných hodnôt a informácie o kľúči). Taktiež je schopný zvoliť iný zašifrovaný text a merať spotrebu energie počas dešifrovania.

3.4.3 Prípadné zraniteľnosti v analýze spotreby

Za účelom skúmania zraniteľnosti cieľovej implementácie platformy k útokom odberovej analýzy sme na meranie použili mikroprocesor STM32F4. Spotreba energie cieľového zariadenia je meraná pomocou osciloskopu Picoscope. Náš útok je **reprodukciou** útoku prezentovaného v článku [8] od *Heysa, Moradiho* a *Paara* z roku 2010, vedeného na kryptosystém McEliece postavený na Goppovych kódoch implementovaného na zariadení STM32F4.

SPA na permutačnej matici. Prvá tajná informácia, ktorá je použitá v dešifrovacom procese je permutačná matica *P*. Po permutácii je zašifrovaný text vynásobený maticou H^T. Násobenie *c** a H^T môže byť efektívne realizované sčítaním týchto riadkov *H* pre každý zodpovedajúci *byt c* = „1“* a preskočenie všetkých „0“, doba trvania násobenia závisí na počte 1 v c*. Ako bolo spomenuté predtým, útočník na postranný kanál by mal poznať poradie vykonaných operácií. Ak áno, môže obnoviť obsah *c** bit po bite a to skúmaním, či sa súčet vykoná alebo nie. Avšak HW *c** je rovnaká ako HW *c*, ale s permutovaným umiestnením bitov. K obnoveniu permutačnej matice používa útočník zašifrovaný text s HW = 1 (v prípade 2048 rôznych zašifrovaných textov) a pre každý zašifrovaný text nájde okamih, kedy sa vykonáva sčítanie. Roztriedenie časových úsekov umožňuje obnoviť celú permutačnú maticu. Obrázok nižšie znázorňuje 2 odberové stopy na začiatku dešifrovania, pre dva rôzne zašifrované texty. Podľa vizuálnej kontroly je začiatok ľahko rozoznateľný a hlavná schéma je podobná schéme uvedenej vyššie. To znamená, že ľubovoľná časť stopy môže byť považovaná za referenčnú vzorku a výpočet cross korelácie referenčnej vzorky. Ďalšie odberové stopy odhaľujú pozície v čaše, keď prebieha výpočet. Nasledujúci obrázok predstavuje dva korelačné vektory pre zodpovedajúce odberové stopy. Všimnite si, že na odstránenie šumu sa opakovali merania a priemerovali 10 stôp, pre každý zašifrovaný text. Použitím tejto schémy pre všetky zašifrované texty s HW = 1, bola permutačná matica kompletne obnovená. Zrealizovanie tohto typu útoku bolo našim cieľom.

3.4.4 Vykonané merania

Po dôkladnom naštudovaní spôsobu útoku a možnostiach vykonania útoku, sme prešli k meraniam. Našou úlohou bolo správne nainštalovať kartu STM32F4, aby bola schopná vykonávať merania na našich počítačoch. Pri tejto fáze bolo potrebné posťahovať správne ovládače. Celý spôsob správneho spustenia meraní na karte je popísaný v kapitole Programovanie mikroprocesora STM32F4.

Pre úspešnosť útoku je dôležité, aby sa nám podarilo určiť permutácie, ktoré vzniknú po vynásobení vstupného vektora a permutačnej matice *P*. Táto permutácia sa prejaví na osciloskope tak, že zašifrovaná jednotka spôsobí rýchly nárast na grafe osciloskopu a zašifrovaná nula zase spôsobuje jemné vlnenie okolo nuly. Každé ďalšie vstupné vektory a násobenia maticou P spôsobia iné posuny zašifrovaných jednotiek, následným porovnávaním stôp týchto „záznamov“ zašifrovaných permutácií koreláciami v prostredí MATLAB získame hodnoty posunov od zafixovanej permutácie a týmto budeme vedieť

približne určiť hodnoty vektorov. V prípade, že by sme mali fixovanú permutáciu so zašifrovanou jednotkou hneď na začiatku, a zároveň by sme mali k dispozícii všetkých 64 nameraných hodnôt, vedeli by sme presne určiť poradie jednotiek nachádzajúcich sa v permutovaných vektoroch. Keďže poznáme permutačnú maticu, vedeli by sme si následne overiť správnosť našich predpokladov. Pri reálnom útoku, však permutačnú maticu nepoznáme a snažíme sa ju zistiť.

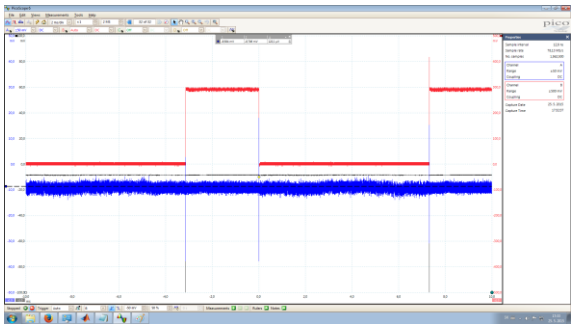
Museli sme zaručiť, že vždy pri šifrovaní budeme používať rovnakú permutačnú maticu, aby McEliece šifroval s rovnakým kľúčom. Pre každú nastavenú hodnotu (nastavená hodnota = číslo pozície, na ktorej sa nachádza jednotka vo vektore) odmeriame 32 vzoriek a z nich urobíme priemer, ktorý potom použijeme na výpočet korelácie v MATLABe.

Nameranie týchto 32 vzoriek v programe Picoscope trvalo približne 8 sekúnd.

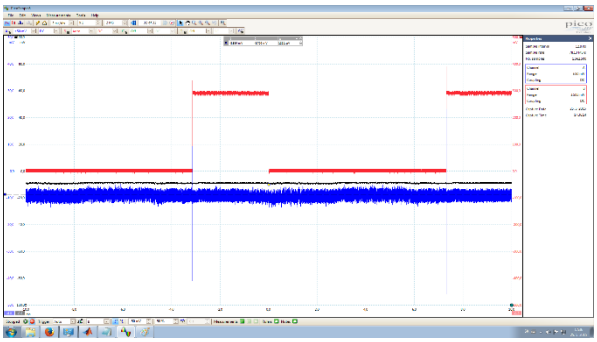
Na nasledujúcich obrázkoch môžete vidieť, ako vyzerá spriemerovaných 32 vzoriek. Červená stopa v meraní bude kanál B – trigger a modrá stopa zodpovedá kanálu A, čiže napätiu. Tmavomodrá farba označuje priemerované hodnoty nameraných vzoriek. Ešte pred sériou obrázkov uvádzame tabuľku, v ktorej môžete vidieť pozície jednotiek v jednotlivých nastaveniach.

Tab.3.6 Pozície jednotiek pre jednotlivé nastavenia

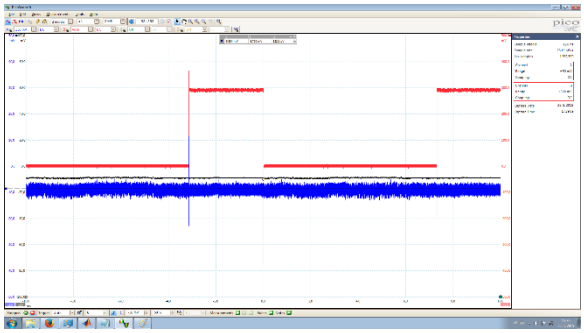
Číslo nastavenia	Pozícia jednotky v zašifrovanom texte
2	6.
3	8.
4	12.
5	16.
6	17.
7	18.
8	19.
9	20.
10	27.



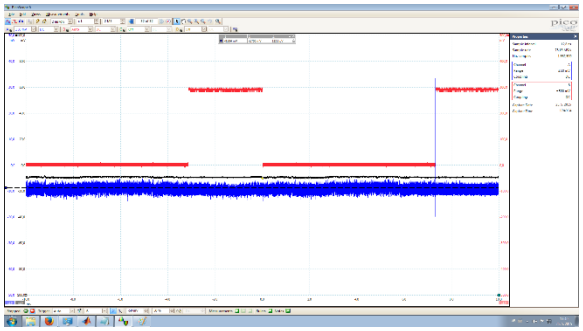
Obr. 3.2 Meranie pre nastavenie č. 2



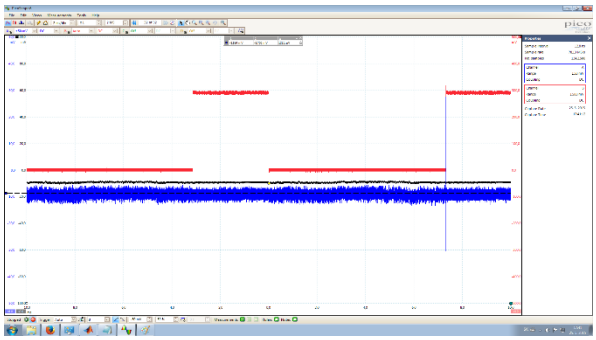
Obr. 3.3 Meranie pre nastavenie č. 3



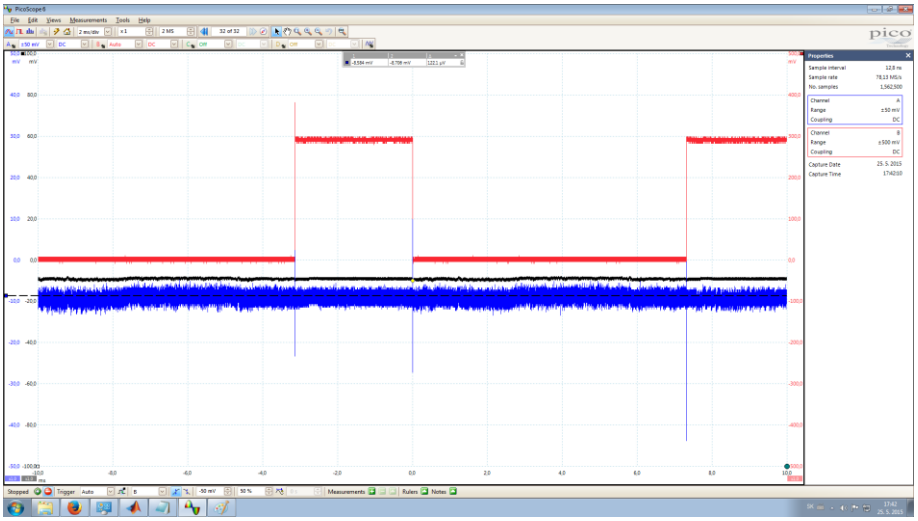
Obr. 3.4 Meranie pre nastavenie č. 6



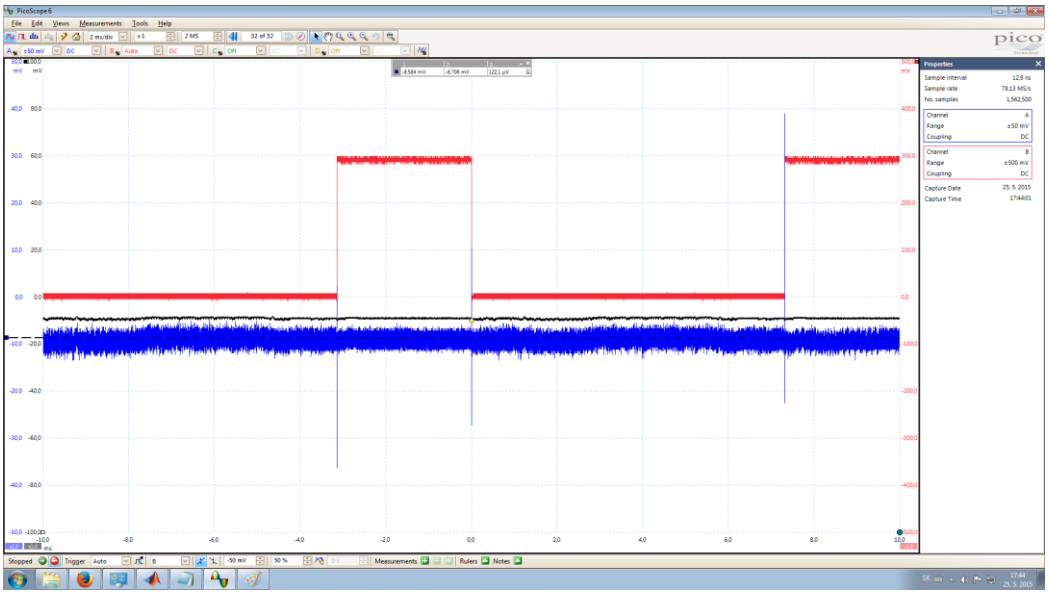
Obr. 3.5 Meranie pre nastavenie č. 7



Obr. 3.6 Meranie pre nastavenie č. 8



Obr. 3.7 Meranie pre nastavenie č. 9



Obr. 3.8 Meranie pre nastavenie č. 10

Permutačná matica, ktorú sme pri šifrovaní používali sa počas desiatich meraní nemenila a vyzerala nasledovne:

Permutačná matica:
Size: 64

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

3.4.5 Výsledky meraní

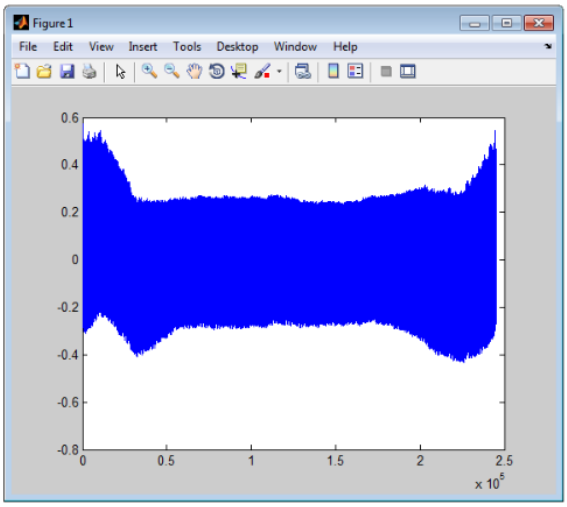
Počítali sme vzájomnú koreláciu medzi jednotlivými vzorkami, aby sme odhalili vzájomný časový posun jednotlivých nameraných vzoriek.

Merania sme vykonali dvakrát, nakoľko sme na prvýkrát vybrali nevhodnú ako referenčnú vzorku č. 2. Na základe výsledkov, ktoré sme dostali pri tejto vzorke, sme pri meraniach za referenčnú vzorku zobrali vzorku číslo 9, s ktorou sme dávali do korelácie ostatné vzorky. Pretože za referenčnú vzorku treba vybrať takú vzorku, ktorá po násobení permutačnou maticou bude mať pozíciu jednotky úplne vľavo v porovnaní s ostatnými vzorkami. Táto vzorka má jednotku na 20. pozícii. V tabuľke č. 4 môžete vidieť jednotlivé hodnoty korelácie a posunov priliehajúcich ku týmto koreláciám. Hodnota korelácia je najvyššie dosiahnutá korelácia v danom časovom posune I.

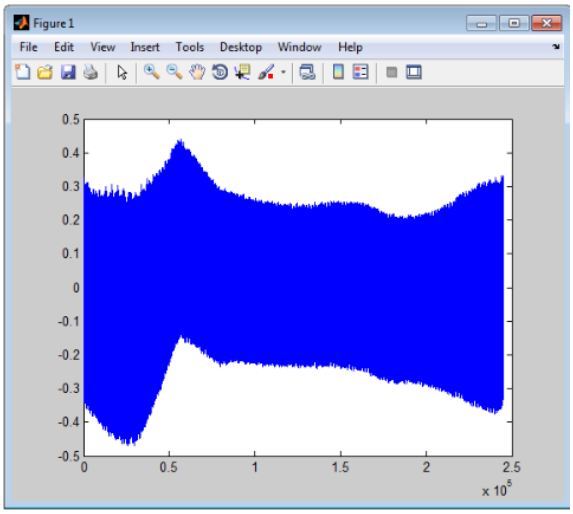
Tab.3.7 Cross korelácia c a posun l pri porovnaní jednotlivých vzoriek

Porovnanie vzoriek	Cross korelácia *c*	Posun *l*	Vhodnosť vzorky
9. vzorka – 3. vzorka	0,5718	54	**Nie**
9. vzorka – 4. vzorka	0,4397	56564	**Áno**
9. vzorka – 5. vzorka	0,5428	84	**Nie**
9. vzorka – 6. vzorka	0,4341	196049	**Áno**
9. vzorka – 7. vzorka	0,5064	67063	**Áno**
9. vzorka – 8. vzorka	0,5545	244805	**Nie**
9. vzorka – 10. vzorka	0,8268	13802	**Nie**
9. vzorka – 2. vzorka	0,6155	243909	**Nie**

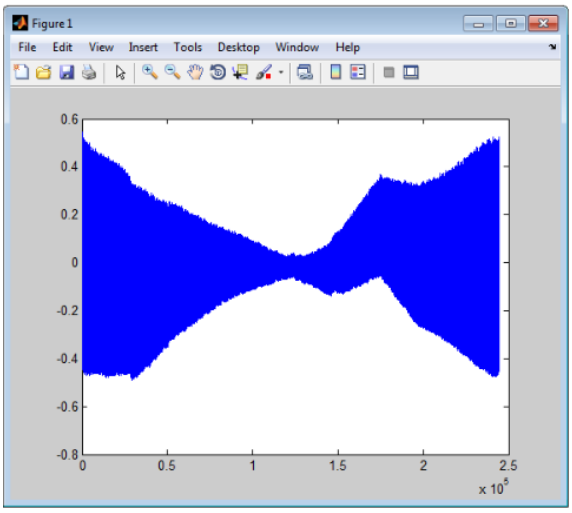
Nasledujúce obrázky ukazujú grafy jednotlivých korelácií vzorky č. 9 s ostatnými vzorkami.



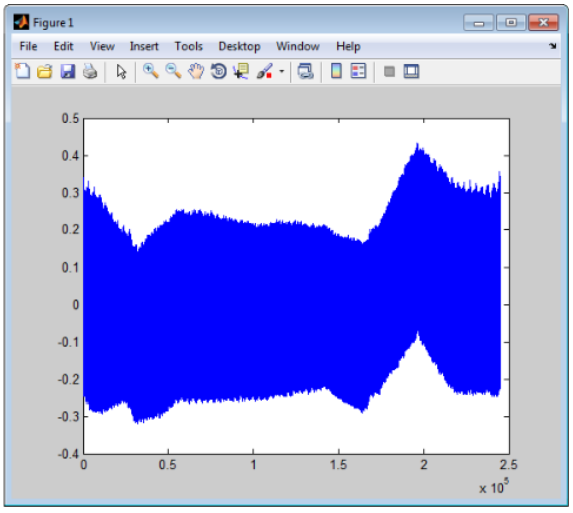
Obr. 3.9 Vzorka č. 9 vs vzorka č. 3



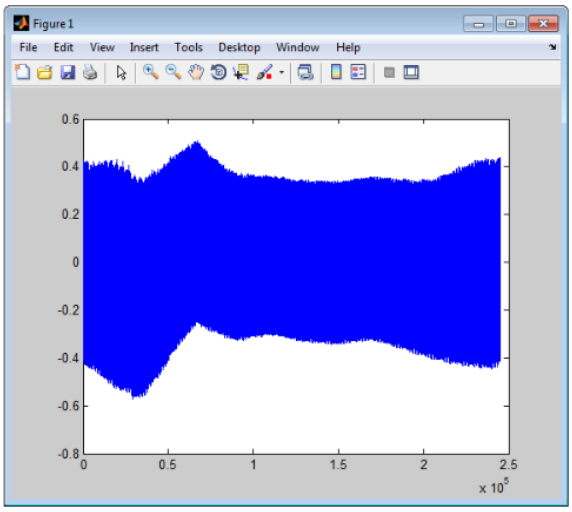
Obr. č. 3.10 Vzorka č. 9 vs vzorka č. 4



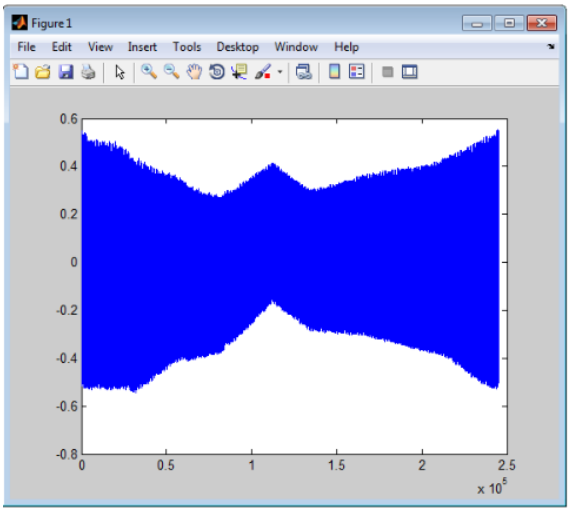
Obr. 3.11 Vzorka č. 9 vs vzorka č. 5



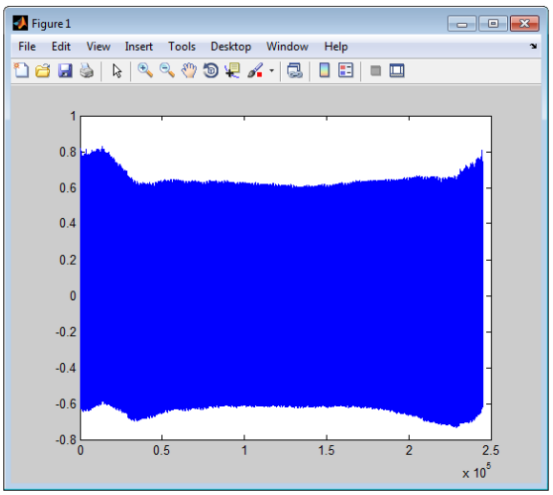
Obr. 3.12 Vzorka č. 9 vs vzorka č. 6



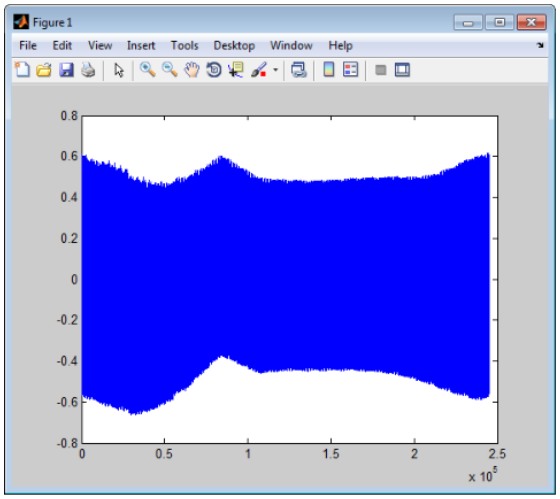
Obr. 3.13 Vzorka č. 9 vs. vzorka č. 7



Obr. 3.14 Vzorka č. 9 vs. vzorka č. 8



Obr. 3.15 Vzorka č. 9 vs. vzorka č. 10



Obr. 3.16 Vzorka č. 9 vs. vzorka č. 2

Pre korektné vyhodnotenie nameraných výsledkov budeme brať do úvahy len vzorky, ktorých posun I sa pohyboval v intervale *<20 000 – 220 000>.* Vhodnosť týchto vzoriek sme zobrazili už v tabuľke č. 4. Zoradíme vybrané vzorky podľa hodnoty I od najmenšej po najväčšiu. Ďalším krokom bude otočiť zoradenie, a teda vzorka, ktorá mala najnižšiu hodnotu posunu I bude mať akoby najväčšiu váhu. Pokračujeme zoradením vzoriek podľa jednotlivých pozícii jednotky vo vektore po prenásobení permutačnou maticou. V prípade, že zoradenia (otočenie poradia a prenásobenie vektora permutačnou rovnicou) sa rovnajú, znamená to, že pomocou tohto útoku vieme odhaliť permutačnú maticu na zariadení, na ktorom je McEliece implementovaný.

Podľa nameraných a dostupných výsledkov sme vybrali vzorku č. 4, vzorku č. 6 a vzorku č. 7. Podľa posunu I sme zoradili vzorky od najmenšej po najväčšiu. Podľa tohto sú vzorky zoradené nasledovne*: 4<7<6.* V ďalšom kroku otočíme vzorky nasledovne: *4>7>6*. Následne vynásobíme vektory jednotlivých vzoriek permutačnou maticou. Pre štvorku sa jednotka nachádza na 12. pozícii, a keďže sa jedná o vektor s Hammingovou váhou 1, tak vynásobenie s permutačnou maticou vykonáme na 12. pozícii, a teda pre štvrtú vzorku dostaneme hodnotu 61. Pre 6. vzorku je vektor s jednotkou na 17. pozícii, a teda po vynásobení s permutačnou maticou, dostaneme pre šiestu vzorku hodnotu 21. Pre vzorku č. 7, pre ktorú sa jednotka vo vektore nachádza na 18. pozícii, a teda je hodnota po vynásobení permutačnou maticou rovná číslu 58. Teraz jednotlivé vzorky podľa týchto hodnôt zoradíme a dostávame: *4>7>6*. Z toho vyplýva, že *4>7>6 = 4>7>6*. Preto môžeme skonštatovať, že útok na permutačnú maticu bol úspešný. Útočník by bol schopný na základe napäťovej analýzy napadnúť McEliecov kryptosystém aj na našom mikroprocesorovom zariadení STM32F4 cez postranné kanály.

3.5 Záver

Experimentálne sme zistili, že implementácia McEliecovho kryptosystému dostupná v projekte BitPunch [5] je náchylná na útok postranným kanálom - meraním spotreby elektrickej energie, pri ktorom by útočník teoreticky vedel zistiť štruktúru permutačnej matice, ktorá tvorí súčasť kľúča tohto kryptosystému. Preto je potrebné venovať zvýšenú pozornosť návrhu samotnej implementácie, aby nebolo možné dešifrovaním rôznych textov zistiť štruktúru permutačnej matice.

Žiaľ, keďže McEliecov kryptosystém je náročný na zdroje (najmä operačnú pamäť), nepodarilo sa nám implementovať plnú verziu systému na zariadenie STM32F4. Verzia, na ktorú sme útočili pracovala len so 64-bitovými kódovými slovami (odporúčané parametre sú 2048-bitové kódové slová). To však neznamená, že by prezentovaný útok nebolo možné vykonať, keďže ten je aplikovateľný na všetky veľkosti kódových slov.

4 Kryptoanalýza vybraných šifier pre lightweight a postkvantovú kryptografiu

V súčasnej dobe sa otázka bezpečnosti informácií a komunikácie stáva stále dôležitejšou. Jednou z možností dosiahnutia bezpečnosti v komunikácii je použitie rôznych šifier. Preto sa hľadajú nové spôsoby a postupy ako súčasné šifrovacie metódy vylepšiť alebo celkom nahradiť novými, kvalitnejšími. Špeciálnym typom šifrovacích metód v komunikácii sú prúdové šifry ktoré majú v danej oblasti nezastupiteľné miesto.

Využitie kvázigrúp je pomerne novým a stále skúmaným prístupom. Vzhľadom na jednoduché operácie, ktoré tieto kryptografické primitíva používajú, sú vhodným kandidátom pre tzv. lightweight kryptografické aplikácie. Viaceré návrhy sú uvažované aj v oblasti tzv. post-kvantovej kryptografie. Ukazuje, že rôzne blokové [9], prúdové šifry [9], [10], [11], [12] alebo hašovacie funkcie [9], [13], [14] využívajúce kvázigrupy prinášajú pozoruhodné výsledky. Viac o rôznych aplikáciách kvázigrúp si je možné naštudovať v nasledujúcich zdrojoch [9], [15], [16], [17]. To je jedným z dôvodov, prečo sme sa rozhodli preskúmať jeden zo spôsobov využitia kvázigrúp a to v prúdových šifrách. Cieľom tejto práce je preskúmať niektoré návrhy prúdových šifier založených na 3-kvázigrupách [10], [11] a posúdiť ich vlastnosti.

4.1 Použité pojmy a označenia

Kvázigrupa

Definícia: Štruktúra (*A*, *) sa nazýva kvázigrupa, ak * je binárna operácia na množine *A* a pre ktorékoľvek dva dané prvky *a*, *b* z *A* majú rovnice *a***x*=*b* a *y***a*=*b* práve jedno riešenie *x*, *y* z *A*.

Izotopia

Definícia: Nech (*G*, .) a (*H*, *) sú dve kvázigrupy. Usporiadaná trojica (θ, φ, ψ) bijektívnych zobrazení θ, φ, ψ z množiny *G* na množinu *H* sa nazýva izotopia (*G*, .) na (*H*, *), ak θ(*x*)*φ(*y*)=ψ(*x*.*y*) pre všetky *x*, *y* patriace *G*. Kvázigrupy (*G*, .) a (*H*, *) sa potom nazývajú izotopné.

n-kvázigrupa

Definícia: *n*-kvázigrupa (*A*, α) môže byť definovaná ako algebra (*A*, α_1, α_2, ..., α_n) s (*n*+1) *n*-árnymi operáciami, spĺňajúcimi nasledovné identity:

$$\alpha(x_1^{i-1}, \alpha_i(x_1^n), x_{i+1}^n) = x_i,$$
$$\alpha_i(x_1^{i-1}, \alpha(x_1^n), x_{i+1}^n) = x_i,$$

pričom $x_b^c = x_b, x_{b+1}, \ldots, x_{c-1}, x_c \ a \ i = 1, 2, \ldots, n.$

Nech $[f_1^n; f]$ je usporiadaný systém permutácií prvkov množiny *A*. Definujeme *n*-árnu

operáciu β na množine *A* nasledovne:

$$\beta(x_1^n) = f(\alpha(f_1^{-1}(x_1), \ldots, f_n^{-1}(x_n)))$$

Potom (*A*, β) je tiež *n*-kvázigrupa a $[f_1^n; f]$ nazývame izotopiou (*A*, α) na (*A*, β).

4.2 Šifrovacia schéma

V tejto časti sa venujeme šifre prezentovanej v [10]. Nech *A* je konečná množina. *M* je množina všetkých konečných, neprázdnych reťazcov zložených z prvkov množiny *A*. *M* je priestor správy (OT). Prvok $m=m_1m_2...m_p$ z *M* sa nazýva vstupný text. Množina *C* je priestor zašifrovaného textu (ZT). V našom prípade *C*=*M*.

Pre každý prvok *a* z *A* nech f_a je permutácia prvkov z *A*. Nech (*A*, α, α_1, α_2, α_3) je 3-kvázigrupa s nosičom *A* (nazýva sa koreňová kvázigrupa).

$K=A^4\times\{1, 2, 3\}$ je priestor šifrovacieho kľúča. Prvok $k=a_1a_2a_3a_4i$ unikátne definuje bijekciu $E_k:M\rightarrow C$ definovanú nasledovne:

Nech (*A*, β) je 3-kvázigrupa s

$$\beta(x_1, x_2, x_3) = f_4(\alpha(f_1^{-1}(x_1), f_2^{-1}(x_2), f_3^{-1}(x_3))$$

kde $f_j = fa_j, j = 1, 2, 3, 4.$

Definujeme $E_k(m_1m_2...m_p)= c_1c_2...c_p$ nasledovne:

- pre $i = 1$

$$\begin{aligned} c_1 &= \beta(m_1, a_1, a_2), \\ c_2 &= \beta(m_2, a_3, a_4), \\ c_j &= \beta(m_j, c_{j-2}, c_{j-1}), \quad j > 2; \end{aligned}$$

- pre $i = 2$

$$\begin{aligned} c_1 &= \beta(a_1, m_1, a_2), \\ c_2 &= \beta(a_3, m_2, a_4), \\ c_j &= \beta(c_{j-2}, m_j, c_{j-1}), \quad j > 2; \end{aligned}$$

- pre $i = 3$

$$\begin{aligned} c_1 &= \beta(a_1, a_2, m_1), \\ c_2 &= \beta(a_3, a_4, m_2), \\ c_j &= \beta(c_{j-2}, c_{j-1}, m_j), \quad j > 2. \end{aligned}$$

E_k je šifrovacia funkcia alebo šifrovacia transformácia. Pre každý kľúč $k=a_1a_2a_3a_4i$, D_k je bijekcia z *C* do *M* a je to dešifrovacia funkcia alebo dešifrovacia transformácia.

Definujeme $D_k(c_1c_2...c_p)= m_1m_2...m_p$ nasledovne:

- pre i = 1

$$m_1 = \beta_1(c_1, a_1, a_2),$$
$$m_2 = \beta_1(c_2, a_3, a_4),$$
$$m_j = \beta_1(c_j, c_{j-2}, c_{j-1}), \quad j > 2;$$

- pre i = 2

$$m_1 = \beta_2(a_1, c_1, a_2),$$
$$m_2 = \beta_2(a_3, c_2, a_4),$$
$$m_j = \beta_2(c_{j-2}, c_j, c_{j-1}), \quad j > 2;$$

- pre i = 3

$$m_1 = \beta_3(a_1, a_2, c_1),$$
$$m_2 = \beta_3(a_3, a_4, c_2),$$
$$m_j = \beta_3(c_{j-2}, c_{j-1}, c_j), \quad j > 2.$$

4.3 Bezpečnosť šifrovacej schémy

Bezpečnosť tejto šifrovacej schémy je založená na fakte, že útočník, ktorý pozná ZT $c=c_1c_2...c_p= E_k\ (m_1m_2...m_p)$ musí vyriešiť systémy vyššie uvedených rovníc, aby zostavil 3-kvázigrupu (A, β). Ale počet 3-kvázigrup s nosičom s n-prvkami je obrovský.

4.4 Kryptosystém s kvázigrupou zadanou algebraicky

Táto šifrovacia schéma môže byť použitá v programe na online digitálnu komunikáciu. Bola uvedená v [10]. Dáta sú reprezentované 8-bitovými prvkami z množiny A={0, ..., 255}. V tom prípade existuje najmenej $256!255!...2!1! > 10^{58000}$ binárnych kvázigrúp s nosičom A. Z každej binárnej kvázigrupy (A, *) môžeme odvodiť dve 3-kvázigrupy :

$$\alpha(x_1, x_2, x_3) = (x_1 * x_2) * x_3,$$
$$\alpha(x_1, x_2, x_3) = x_1 * (x_2 * x_3).$$

Takáto 3-kvázigrupa sa nazýva reducibilná. Ale existujú aj ireducibilné 3-kvázigrupy s nosičom A.

Priestor kľúča K má $(256^4-1)*3$ prvkov. Triviálne kľúče (0, 0, 0, 0, i), i=1, 2, 3 sú zakázané.

Je dôležité mať jednoducho spočítateľnú šifrovaciu a dešifrovaciu funkciu. Ako koreňová kvázigrupa je použitá (A, α) s

$$\alpha(x_1, x_2, x_3) = x_1 - x_2 + x_3 \qquad (mod\ 256)$$

a pre každé a z A

$$f_a(x) = x + a \qquad (mod\ 256).$$

Potom

$$\begin{aligned}
\beta(x_1, x_2, x_3) &= x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \quad (mod\, 256),\\
\beta_1(x_1, x_2, x_3) &= x_1 + x_2 - x_3 + a_1 - a_2 + a_3 - a_4 \quad (mod\, 256),\\
\beta_2(x_1, x_2, x_3) &= x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \quad (mod\, 256),\\
\beta_3(x_1, x_2, x_3) &= -x_1 + x_2 + x_3 + a_1 - a_2 + a_3 - a_4 \quad (mod\, 256).
\end{aligned}$$

V tomto systéme je dôležité zachovať utajený kľúč a koreňovú kvázigrupu, z ktorej sa odvodzujú šifrovacie a dešifrovacie transformácie.

Príklad

Dáta sú reprezentované 3-bitovými prvkami z množiny *A*={0, 1, 2, 3, 4, 5, 6, 7}. Nech OT $m=m_1m_2m_3m_4m_5m_6m_7m_8$=23410576 a kľúč $k=a_1a_2a_3a_4i$=76542, z čoho *i*=2.

Pri šifrovaní využijeme šifrovaciu funkciu

$$\begin{aligned}
c_1 &= \beta(a_1, m_1, a_2),\\
c_2 &= \beta(a_3, m_2, a_4),\\
c_j &= \beta(c_{j-2}, m_j, c_{j-1}), \quad j > 2,
\end{aligned}$$

pričom

$$\beta(x_1, x_2, x_3) = x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \quad (mod\, 8).$$

Šifrovanie:

$$\begin{aligned}
c_1 &= \beta(7, 2, 6) = 7 - 2 + 6 - 7 + 6 - 5 + 4 = 1,\\
c_2 &= \beta(5, 3, 4) = 5 - 3 + 4 - 7 + 6 - 5 + 4 = 4,\\
c_3 &= \beta(1, 4, 4) = 1 - 4 + 4 - 7 + 6 - 5 + 4 = 7,\\
c_4 &= \beta(4, 1, 7) = 4 - 1 + 7 - 7 + 6 - 5 + 4 = 0,\\
c_5 &= \beta(7, 0, 0) = 7 - 0 + 0 - 7 + 6 - 5 + 4 = 5,\\
c_6 &= \beta(0, 5, 5) = 0 - 5 + 5 - 7 + 6 - 5 + 4 = 6,\\
c_7 &= \beta(5, 7, 6) = 5 - 7 + 6 - 7 + 6 - 5 + 4 = 2,\\
c_8 &= \beta(6, 6, 2) = 6 - 6 + 2 - 7 + 6 - 5 + 4 = 0,
\end{aligned}$$

čiže ZT *c*=14705620.

Pri dešifrovaní využijeme dešifrovaciu funkciu

$$\begin{aligned}
m_1 &= \beta_2(a_1, c_1, a_2),\\
m_2 &= \beta_2(a_3, c_2, a_4),\\
m_j &= \beta_2(c_{j-2}, c_j, c_{j-1}), \quad j > 2,
\end{aligned}$$

pričom

$$\beta_2(x_1, x_2, x_3) = x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \quad (mod\, 8).$$

Dešifrovanie:

$$
\begin{aligned}
m_1 &= \beta_2(7,1,6) = 7-1+6-7+6-5+4 = 2,\\
m_2 &= \beta_2(5,4,4) = 5-4+4-7+6-5+4 = 3,\\
m_3 &= \beta_2(1,7,4) = 1-7+4-7+6-5+4 = 4,\\
m_4 &= \beta_2(4,0,7) = 4-0+7-7+6-5+4 = 1,\\
m_5 &= \beta_2(7,5,0) = 7-5+0-7+6-5+4 = 0,\\
m_6 &= \beta_2(0,6,5) = 0-6+5-7+6-5+4 = 5,\\
m_7 &= \beta_2(5,2,6) = 5-2+6-7+6-5+4 = 7,\\
m_8 &= \beta_2(6,0,2) = 0-0+0-7+6-5+4 = 6.
\end{aligned}
$$

Príklad s použitím nevhodného kľúča

Sú použité rovnaké vstupy ako v predošlom príklade okrem kľúča. Je vybraný jeden zo zakázaných kľúčov, ktorý má všetky znaky nulové okrem posledného.

OT $m=m_1m_2m_3m_4m_5m_6m_7m_8$=23410576 a kľúč $k=a_1a_2a_3a_4i$=00002, z čoho i=2. Pri šifrovaní využijeme šifrovaciu funkciu

$$
\begin{aligned}
c_1 &= \beta(a_1, m_1, a_2),\\
c_2 &= \beta(a_3, m_2, a_4),\\
c_j &= \beta(c_{j-2}, m_j, c_{j-1}), \quad j > 2,
\end{aligned}
$$

pričom

$$\beta(x_1,x_2,x_3) = x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \pmod 8.$$

Šifrovanie:

$$
\begin{aligned}
c_1 &= \beta(0,2,0) = 0-2+0-0+0-0+0 = 6,\\
c_2 &= \beta(0,3,0) = 0-3+0-0+0-0+0 = 5,\\
c_3 &= \beta(6,4,5) = 6-4+5-0+0-0+0 = 7,\\
c_4 &= \beta(5,1,7) = 5-1+7-0+0-0+0 = 3,\\
c_5 &= \beta(7,0,3) = 7-0+3-0+0-0+0 = 2,\\
c_6 &= \beta(3,5,2) = 3-5+2-0+0-0+0 = 0,\\
c_7 &= \beta(2,7,0) = 2-7+0-0+0-0+0 = 3,\\
c_8 &= \beta(0,6,3) = 0-6+3-0+0-0+0 = 5,
\end{aligned}
$$

čiže ZT c=65732035.

Pri dešifrovaní využijeme dešifrovaciu funkciu

$$m_1 = \beta_2(a_1, c_1, a_2),$$
$$m_2 = \beta_2(a_3, c_2, a_4),$$
$$m_j = \beta_2(c_{j-2}, c_j, c_{j-1}), \quad j > 2,$$

pričom

$$\beta_2(x_1, x_2, x_3) = x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \pmod 8.$$

Dešifrovanie:

$$m_1 = \beta_2(0, 6, 0) = 0 - 6 + 0 - 0 + 0 - 0 + 0 = 2,$$
$$m_2 = \beta_2(0, 5, 0) = 0 - 5 + 0 - 0 + 0 - 0 + 0 = 3,$$
$$m_3 = \beta_2(6, 7, 5) = 6 - 7 + 5 - 0 + 0 - 0 + 0 = 4,$$
$$m_4 = \beta_2(5, 3, 7) = 5 - 3 + 7 - 0 + 0 - 0 + 0 = 1,$$
$$m_5 = \beta_2(7, 2, 3) = 7 - 2 + 3 - 0 + 0 - 0 + 0 = 0,$$
$$m_6 = \beta_2(3, 0, 2) = 3 - 0 + 2 - 0 + 0 - 0 + 0 = 5,$$
$$m_7 = \beta_2(2, 3, 0) = 2 - 3 + 0 - 0 + 0 - 0 + 0 = 7,$$
$$m_8 = \beta_2(0, 5, 3) = 0 - 5 + 3 - 0 + 0 - 0 + 0 = 6.$$

4.4.1 Útok na kryptosystém so znalosťou páru OT, ZT

V prípade, že útočník pozná dešifrovacie transformácie kryptosystému a má pár OT, ZT zašifrovaný daným kryptosystémom, tak vie ľahko získať dostatok informácii na rozlúštenie akéhokoľvek ZT zašifrovaného rovnakým kľúčom s použitím tých istých transformácií.

Poznáme:

- OT, ZT,
- β_1, β_2, β_3.

Keďže v dešifrovacích transformáciách β_1, β_2, β_3 pre znaky ZT c_3, c_4, ... využívame ako parametre iba znaky ZT, tak je jednoduché zostrojiť sústavu rovníc pomocou ktorých vypočítame informáciu o kľúči.

Príklad

Nech OT $m=m_1m_2m_3m_4m_5m_6m_7m_8$=23410576 a kľúč $k=a_1a_2a_3a_4i$=76542, z čoho i=2 a ZT c=14705620. Útočník nepozná kľúč, ale pozná dešifrovacie transformácie:

Ak i = 1

$$m_1 = \beta_1(c_1, a_1, a_2),$$
$$m_2 = \beta_1(c_2, a_3, a_4),$$
$$m_j = \beta_1(c_j, c_{j-2}, c_{j-1}), \quad j > 2.$$

Ak i = 2

$$m_1 = \beta_2(a_1, c_1, a_2),$$
$$m_2 = \beta_2(a_3, c_2, a_4),$$
$$m_j = \beta_2(c_{j-2}, c_j, c_{j-1}), \quad j > 2.$$

Ak i = 3

$$m_1 = \beta_3(a_1, a_2, c_1),$$
$$m_2 = \beta_3(a_3, a_4, c_2),$$
$$m_j = \beta_3(c_{j-2}, c_{j-1}, c_j), \quad j > 2.$$

$$\beta_1(x_1, x_2, x_3) = x_1 + x_2 - x_3 + a_1 - a_2 + a_3 - a_4 \pmod 8,$$
$$\beta_2(x_1, x_2, x_3) = x_1 - x_2 + x_3 - a_1 + a_2 - a_3 + a_4 \pmod 8,$$
$$\beta_3(x_1, x_2, x_3) = -x_1 + x_2 + x_3 + a_1 - a_2 + a_3 - a_4 \pmod 8.$$

Z rovníc je jasné, že pre dešifrovanie potrebuje zistiť informáciu o kľúči

$y_k = \pm a_1 \pm a_2 \pm a_3 \pm a_4 = \pm 7 \pm 6 \pm 5 \pm 4$.

Na vypočítanie tohto parametra je potrebné zostaviť sústavy rovníc pre každú kvázigrupu, pretože nevieme, ktorá bola použitá.

Ak i = 1

$$m_3 = \beta_1(7, 1, 4) = 7 + 1 - 4 + y_k = 4 \rightarrow y_k = 0 \pmod 8,$$
$$m_4 = \beta_1(0, 4, 7) = 0 + 4 - 7 + y_k = 1 \rightarrow y_k = 4 \pmod 8,$$
$$m_5 = \beta_1(5, 7, 0) = 5 + 7 - 0 + y_k = 0 \rightarrow y_k = 4 \pmod 8,$$
$$\vdots \qquad\qquad \vdots$$

Ak i = 2

$$m_3 = \beta_2(1, 7, 4) = 1 - 7 + 4 + y_k = 4 \rightarrow y_k = 6 \pmod 8,$$
$$m_4 = \beta_2(4, 0, 7) = 4 - 0 + 7 + y_k = 1 \rightarrow y_k = 6 \pmod 8,$$
$$m_5 = \beta_2(7, 5, 0) = 7 - 5 + 0 + y_k = 0 \rightarrow y_k = 6 \pmod 8,$$
$$\vdots \qquad\qquad \vdots$$

Ak i = 3

$$m_3 = \beta_3(1, 4, 7) = -1 + 4 + 7 + y_k = 4 \rightarrow y_k = 2 \pmod 8,$$
$$m_4 = \beta_3(4, 7, 0) = -4 + 7 + 0 + y_k = 1 \rightarrow y_k = 6 \pmod 8,$$
$$m_5 = \beta_3(7, 0, 5) = -7 + 0 + 5 + y_k = 0 \rightarrow y_k = 2 \pmod 8,$$
$$\vdots \qquad\qquad \vdots$$

Zo sústav vyplýva, že iba v prípade, keď je využitá dešifrovacia transformácia β_2 sa pre všetky znaky ZT parameter y_k zhoduje.

Jediný problém je, že nevieme dešifrovať prvé dva znaky ZT, pretože pri ich dešifrovaní sú v transformáciách využité namiesto znakov ZT znaky kľúča, ktorý nepoznáme.

Vzhľadom na to, že bezpečnosť šifrovacej schémy je do veľkej miery závislá od utajenia dešifrovacích transformácií, čiže použitej 3-kvázigrupy, je zložitosť tohto útoku relatívne malá.

V prípade, že dáta sú reprezentované prvkami množiny A={0, 1, 2, 3, 4, 5, 6, 7} ako v našom prípade, potom existuje 8!7!6!5!4!3!2!1!=5056584744960000 binárnych kvázigrúp s nosičom *A*.

Tým, že poznáme dešifrovacie transformácie, nemusíme zostavovať použitú 3-kvázigrupu, ale bezpečnosť je založená iba na znalosti kľúča, čiže stačí vyriešiť tri systémy rovníc. V najhoršom prípade teda treba vyriešiť 2(*p*-2) rovníc, kde *p* je počet znakov zašifrovaného textu.

4.5 Kryptosystém s kvázigrupou zadanou tabuľkou

Tento kryptosystém bol uverejnený v článku [11], ktorý bol inšpirovaný článkom [10]. Podstata kryptosystému je zachovaná až na spôsob použitia 3-kvázigrupy. V pôvodnom článku bola zadaná algebricky. Tento kryptosystém naopak využíva 3-kvázigrupu zadanú tabuľkou.

Kryptosystém využíva dve rôzne metódy generovania 3-kvázigrupy. Prvý spôsob sa používa v prípade, že množina *A* obsahuje menej ako 17 prvkov. Druhý spôsob je použitý pri väčších množinách a využíva pseudonáhodný generátor, ktorý však nebol ďalej špecifikovaný. V tejto práci sme sa venovali prvej zo spomínaných metód.

4.5.1 Pseudonáhodné generovanie kvázigrupy na základe kľúča

Nech *A*={1, 2, ..., *n*} je množina *n* prvkov. Na vygenerovanie pseudonáhodnej kvázigrupy rádu *n*, potrebná dĺžka kľúča je 2*n*. Priestor kľúča je teda $K=\{a_1...a_{2n}\}$, kde $a_j \in A$, $1 \leq j \leq 2n$. Nech kľúč $k= a_1...a_{2n}$. Najprv si zoberieme základnú kvázigrupu (A, α) reprezentovanú maticou M_s, ktorej 1.riadok je identická permutácia prvkov *A* a ostatných *n*-1 riadkov vznikne rotáciou predchádzajúceho riadka o jeden znak doľava.

$$M_s = \begin{bmatrix} 1 & 2 & 3 & \dots & n \\ 2 & 3 & \dots & n & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ n & 1 & \dots & \dots & n-1 \end{bmatrix}$$

Aplikovaním nasledovného algoritmu založeného na znakoch $a_1...a_n$ kľúča *k* na riadky matice M_s získame maticu M_{sr}. Nech x mod m ≡ n ak n|x. r_1,r_2 -riadky matice *temp*. s_1, s_2 - stĺpce matice *temp*.

Algoritmus 1: Algoritmus na riadkov

$temp = M_s$;
for $j = 1 : n$ **do**
  $r_1 = (j + a_j) \mod n$;
  $r_2 = (\lceil \frac{n}{2} \rceil * j + 1 + a_j) \mod n$;
  Swap r_1 and r_2 of temp;
$M_{sr} = temp$;

Podobne sú *n* krát vymenené aj stĺpce matice M_{sr} na základe znakov $a_{n+1}...a_{2n}$ hesla *k*. Týmto procesom získame kvázigrupu (A, γ), ktorú nazývame koreňová kvázigrupa. Táto kvázigrupa je reprezentovaná maticou M_f.

Algoritmus 2: Algoritmus na vymieňanie stĺpcov

$temp = M_{sr}$;
for $j = n + 1 : 2n$ **do**
  $s_1 = (j + a_j) \mod n$;
  $s_2 = (\lceil \frac{n}{2} \rceil * j + 1 + a_j) \mod n$;
  Swap s_1 and s_2 of temp;
$M_f = temp$;

Príklad

Nech rád $|A| = 4$ a nech kľúč $k=a_1...a_8=41312143$. V tomto prípade $\lceil n/2 \rceil = 2$ a

$$M_s = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{bmatrix}.$$

Po vymieňaní riadkov 4 krát podľa algoritmu

$$M_{sr} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \\ 2 & 3 & 4 & 1 \\ 4 & 1 & 2 & 3 \end{bmatrix}.$$

Teraz aplikujeme vymieňanie stĺpcov podľa algoritmu a vytvoríme koreňovú kvázigrupu

$$M_f = \begin{bmatrix} 3 & 1 & 4 & 2 \\ 1 & 3 & 2 & 4 \\ 4 & 2 & 1 & 3 \\ 2 & 4 & 3 & 1 \end{bmatrix}.$$

4.5.2 Analýza

V tejto časti analyzujeme danú metódu pre generovanie kvázigrúp, ktorá bola zverejnená v článku [11].

V prvom rade sme testovali aké množstvo rôznych kvázigrúp daného rádu je táto metóda schopná vygenerovať v porovnaní so známym počtom kvázigrúp daného rádu. Ukázalo sa, že získaný počet kvázigrúp je v porovnaní s celkovým počtom kvázigrúp daného rádu značne nižší. Z tohto poznatku je možné vyvodiť nasledovnú hypotézu:

Hypotéza: Množina hesiel sa pri danej metóde rozdelí na skupiny, ktoré generujú rovnakú kvázigrupu, čiže aj 3-kvázigrupu.

Experimenty pre rády vyššie ako 5 nebolo možné vzhľadom na časovú náročnosť výpočtu realizovať exhaustívnym spôsobom. V nasledujúcej tabuľke je možné vidieť zistené výsledky. „Počet kvázigrúp“ označuje celkový známy počet kvázigrúp daného rádu. „Počet unikátnych kvázigrúp” označuje počet rôznych kvázigrúp daného rádu, ktoré boli vygenerované príslušnými heslami. 6.stĺpec so znakom „%” označuje akú veľkú časť zo všetkých kvázigrúp predstavujú unikátne kvázigrupy vyjadrenú percentuálne.

Tab.4.1 Vzťah medzi heslami a kvázigrupami

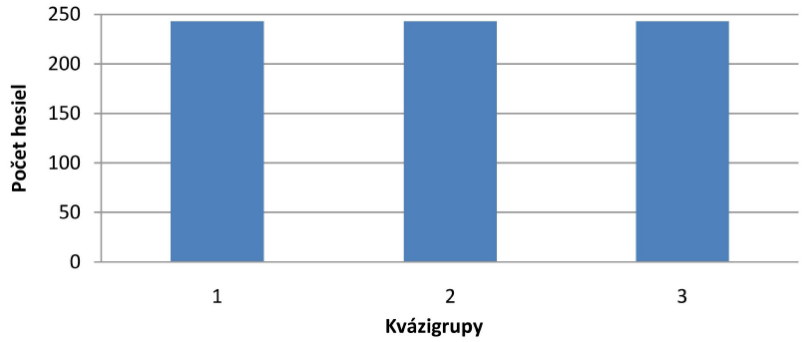
Rád	Počet hesiel		Počet kvázigrúp	Počet unikátnych kvázigrúp	%
3	729	>	12	3	25
4	65536	>	576	72	12,5
5	9765625	>	161280	720	0,446
6	2176782336	>	812851200	-	-
7	678223072849	<	61479419904000	-	-
8	281474976710656	<	108776032459082956800	-	-

Vzhľadom na to, že algoritmus 1 modifikuje základnú kvázigrupu M_s iba výmenou riadkov a výmenou stĺpcov, je možné vyvodiť nasledovné tvrdenie:

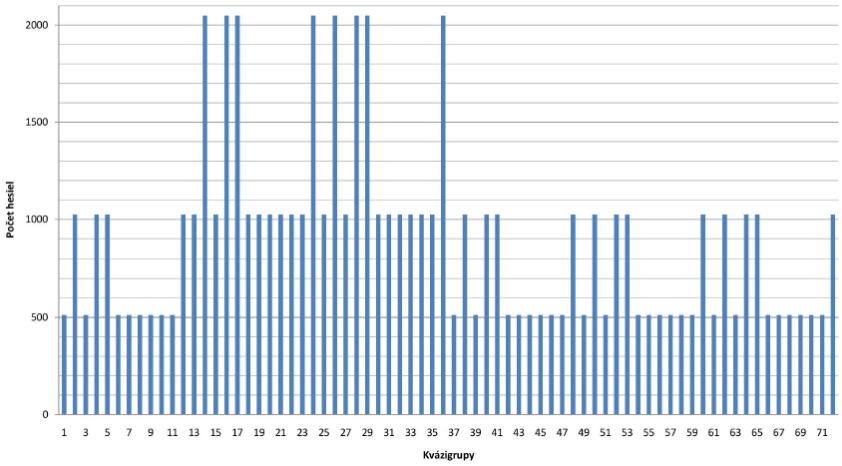
Tvrdenie: Všetky kvázigrupy vygenerované algoritmom 1 sú vzájomne izotopné.

Dá sa predpokladať, že toto je dôvod nízkeho počtu unikátnych kvázigrúp, ktoré je tento algoritmus schopný vygenerovať.

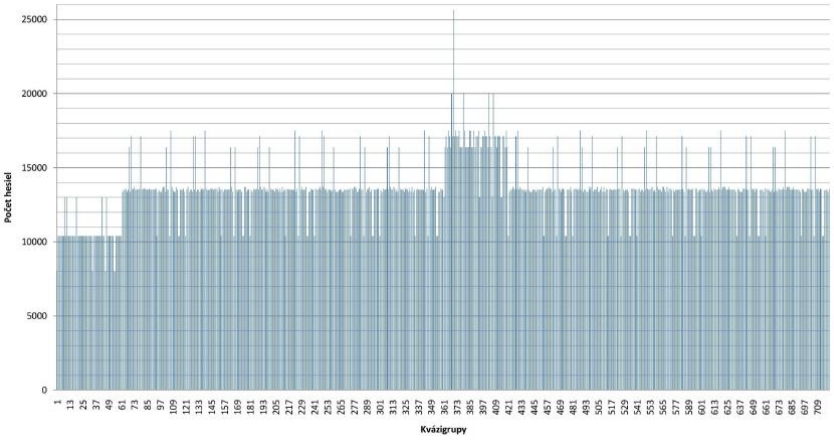
Zaujímavým faktom je, že jednotlivé kvázigrupy neboli vygenerované rovnakým počtom hesiel, ale boli medzi nimi značné rozdiely, ako je možné vidieť v grafoch na nasledovných obrázkoch.



Obr. 4.1 Rozloženie početnosti hesiel pre kvázigrupy rádu 3



Obr. 4.2 Rozloženie početnosti hesiel pre kvázigrupy rádu 4



Obr. 4.3 Rozloženie početnosti hesiel pre kvázigrupy rádu 5

Ďalšou vlastnosťou danej schémy je, že je možné odhadnúť, ktoré špeciálne zvolené heslá vygenerujú rovnakú kvázigrupu. Napríklad pre nižšie rády kvázigrúp ako napríklad 4 a 5 platí, že prvých n^2 hesiel je v rovnakých skupinách. Tento fakt je naznačený v nasledujúcej tabuľke pre rád n=4.

Tab.4.2 Niektoré skupiny hesiel generujúcich rovnakú kvázigrupu pre rád 4

1.	2.	...	*n*	...
11111111	11111112	...	1111111*n*	...
11111121	11111122	...	1111112*n*	...
...	...	...	...	...
111111*n*1	111111*n*2	...	...	...
...	...	...	...	...

Tab.4.3 Niektoré skupiny hesiel generujúcich rovnakú kvázigrupu pre rád 4

1.	2.	...	*n*	...
11111111	11121111	...	111*n*1111	...
11211111	11221111	...	112*n*1111	...
...	...	...	...	...
11*n*11111	11*n*21111	...	11*nn*1111	...
...	...	...	...	...

Po krátkej analýze algoritmu 1 je možné predpokladať, že ak budú dve rôzne heslá generovať tú istú kvázigrupu *A*, čiže budú v jednej skupine, tak aj heslá, ktoré vzniknú výmenou polovíc týchto hesiel budú generovať tú istú kvázigrupu *B*. Táto vlastnosť je dosiahnutá tým, že algoritmus využíva prvú polovicu hesla na vymieňanie riadkov a druhú polovicu na vymieňanie stĺpcov. Túto vlastnosť sme testovali na kvázigrupách rádu 4, ale je jasné, že je všeobecne platná pre všetky rády.

Z faktu, že prvých n^2 hesiel je pri rádoch 4 a 5 v rovnakých skupinách sme predpokladali, že to tak platí aj pri vyšších rádoch. Tento predpoklad sa však po vykonaných experimentoch nepotvrdil. Aj v heslách pre vyššie rády sa dá nájsť spojitosť, z ktorej vyplýva, že budú generovať rovnakú kvázigrupu (budú v rovnakej skupine). Tieto skupiny však nie sú také mohutné ako pri spomínaných rádoch 4 a 5. Tieto skupiny sú viac roztrúsené v množine

hesiel, ale sú dostatočne veľké, aby ich skúmanie a hľadanie malo praktický význam pri voľbe hesiel. Nasleduje analýza typu hesiel, ktoré generujú rovnakú kvázigrupu:

Rád 4 Heslá pre rád 4 sme si už z časti rozobrali, ale ešte sa v ňom vyskytujú určité ďalšie typy hesiel.
Napr.: $111111x1, 111211x1, 111311x1, 111411x1$ a podobne len znovu vymenené polovice $11x11111, 11x11112, \ldots x \in \{1,2,3,4\}$

Rád 5 Pri ráde 5 platia pre heslá úplne rovnaké zákonitosti ako pri ráde 4, len sú tieto heslá pochopiteľne dlhšie (10 znakové).

Rád 6 V ráde 6 sa vyskytuje iný typ vzťahu medzi heslami, ktorý sme nedokázali roztriediť do jednotlivých skupín. Vo všeobecnosti by sa dalo povedať, že je dosť veľká pravdepodobnosť že ak budú mať dve heslá tvar $1\ldots1xy, 1111xy1\ldots1$, $1\ldots1x1111y$ alebo $x1111y1\ldots1$ pričom $x, y \in \{1,2,3,4,5,6\}$, tak tieto heslá vygenerujú identickú kvázigrupu. Napríklad heslá 111111311113, 111111611113 sú takéhoto typu.
Ďalším typom hesiel, ktoré sú v rovnakej skupine, sú heslá $1\ldots1x111151$, $1\ldots1x211111$ a $1\ldots1x511111$ pričom $x, y \in \{1,2,3,4,5,6\}$.

Rád 7 V ráde 7 sa vyskytuje viac typov.
Napr.: 1. typ: $1\ldots1x1111, 1\ldots1x1112, 1\ldots1x1113, \ldots$ a podobne pri výmene polovíc hesiel $11x11111\ldots1, 11x11121\ldots1, \ldots$
2. typ: $11x1\ldots11, 11x1\ldots12, 11x1\ldots13, \ldots$ a opäť podobne aj pri výmene strán $1\ldots1111x1111, 1\ldots1211x1111, 1\ldots1311x1111, \ldots$ pričom $x \in \{1,2,3,4,5,6,7\}$.

Rád 8 1. typ: $1\ldots1x111, 1\ldots1x112, \ldots$ a naopak $1111x1111\ldots1, 1111x1121\ldots1, \ldots$
2. typ: $1\ldots111111x111, 1\ldots121111x111, 1\ldots131111x111, \ldots$ a po výmene $1111x1\ldots11, 1111x1\ldots12, 1111x1\ldots13, \ldots$ pričom $x \in \{1,2,3,4,5,6,7,8\}$.

Rád 9 Pri ráde 9 nastáva podobná situácia ako pri ráde 6. Tento typ vzťahu by sa dal zapísať ako $1\ldots1x1111y$ a $111x1111y1\ldots1$ pričom $x, y \in \{1,2,3,4,5,6,7,8,9\}$.

Rád 10 Pri ráde kvázigrupy 10 je zložitejšie zapísať jednotlivé skupiny hesiel, pretože ich je viac a sú menšie. 3 hlavné skupiny sú: $1\ldots1x11y, 1\ldots1x1111y$ a $1\ldots1x11111111y$ a podobne pri výmene strán pričom $x, y \in \{1,2,\ldots,10\}$.

Rád 11 Skupiny pre rád 11 sú opäť kompaktnejšie ako v predchádzajúcich dvoch prípadoch. Obmedzím sa na vypísanie len niektorých z nich, pretože ostatné je jednoduché odvodiť. Pre prehľadnosť budem v tomto a nasledujúcich rádoch písať znak bodky, pretože by mohlo k dôjsť k neprehľadnosti jednotlivých pozícií. Veľkosť hesiel je 22 znakov.

1. typ: $1\ldots1.x.1.1.1, 1\ldots1.x.1.1.2, \ldots, 1\ldots1.x.1.1.11$ a podobne pri výmene polovíc: $1.1.1.1.1.1.1.x.1.1.1.1.\ldots1, 1.1.1.1.1.1.1.x.1.1.2.1.\ldots1, \ldots, 1.1.1.1.1.1.1.x.1.1.11.1.\ldots1.$

2. typ: $1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.x.1.1.1, \ldots, \underbrace{1.\ldots1}_{10}.11.1.1.1.1.1.1.1.x.1.1.1$ podobne po výmene pričom $x \in \{1,2,3,4,5,6,7,8,9,10,11\}$.

Rád 12 1. typ: $\underbrace{1.\ldots1}_{18}.x.1.1.1.1.1, \ldots, \underbrace{1.\ldots1}_{18}.x.1.1.1.1.12$ a podobne po výmene

$\underbrace{1.\ldots1}_{6}.x.1.1.1.1.1.\underbrace{1.\ldots1}_{12}, \ldots, \underbrace{1.\ldots1}_{6}.x.1.1.1.1.12.\underbrace{1.\ldots1}_{12}$

2. typ: $\underbrace{1.\ldots1}_{11}.1.1.1.1.1.1.1.x.\underbrace{1.\ldots1}_{5}, \ldots, \underbrace{1.\ldots1}_{11}.12.1.1.1.1.1.1.x.\underbrace{1.\ldots1}_{5}$ a podobne po výmene pričom $x \in \{1,2,3,\ldots,12\}$.

Rád 13 1. typ: $\underbrace{1.\ldots1}_{17}.x.1.1.1.1.1.1.1.1, \ldots, \underbrace{1.\ldots1}_{17}.x.1.1.1.1.1.1.1.12$ a podobne po výmene polovíc.

2.typ: $\underbrace{1.\ldots1}_{12}.1.1.1.1.1.x, \ldots, \underbrace{1.\ldots1}_{12}.12.1.1.1.1.x$ a podobne po výmene polovíc pričom $x \in \{1,2,3,\ldots,13\}$.

Rád 14 1.typ: $\underbrace{1.\ldots1}_{20}.x.1.1.1.1.1.1.1.y$ a po výmene $\underbrace{1.\ldots1}_{6}.x.1.1.1.1.1.1.1.y.\underbrace{1.\ldots1}_{14}$

2. typ: $\underbrace{1.\ldots1}_{22}.x.1.1.1.1.y$ a po výmene $\underbrace{1.\ldots1}_{8}.x.1.1.1.1.y.\underbrace{1.\ldots1}_{14}$ pričom $x, y \in \{1,2,3,\ldots,14\}$.

Rád 15 $\underbrace{1.\ldots1}_{24}.x.1.1.1.1.y$ a po výmene $\underbrace{1.\ldots1}_{9}.x.1.1.1.1.y.\underbrace{1.\ldots1}_{15}$ pričom $x, y \in \{1,2,3,\ldots,15\}$.

Rád 16 1. typ: $\underbrace{1.\ldots1}_{24}.x.1.1.1.1.1.1.1.1, \ldots, \underbrace{1.\ldots1}_{24}.x.1.1.1.1.1.1.1.16$ a podobne po výmene.

2. typ: $\underbrace{1.\ldots1}_{8}.x.\underbrace{1.\ldots1}_{22}.1, \ldots, \underbrace{1.\ldots1}_{8}.x.\underbrace{1.\ldots1}_{22}.16$ a podobne po výmene polovíc pričom $x \in \{1,2,3,\ldots,16\}$.

Pri skúmaní jednotlivých skupín hesiel a kvázigrúp, ktoré generujú, sa objavila myšlienka, či je skupina hesiel, ktorá po použití algoritmu 1 na počiatočnú kvázigrupu opäť vygeneruje tú istú kvázigrupu. Ukázalo sa, že takáto skupina existuje pri ráde 4 aj 5 a že je dosť veľká. V prípade rádu 4 to bolo 2048 hesiel a pre rád 5 až 25625 hesiel. Toto sú najväčšie skupiny, čo má značný vplyv na voľbu hesla v prípade, že chceme docieliť čo najvyššiu úroveň bezpečnosti.

4.6 Štatistické testovanie kvázigrupovej blokovej šifry

V tejto časti sa venujeme štatistickému testovaniu kvázigrupovej blokovej šifry, ktorá bola navrhnutá v práci [18].

Cieľom autorov návrhu tejto šifry bolo priblížiť funkčnosť a efektivitu kvázigrupových blokových šifier populárnemu štandardu AES. Využili metódu viacnásobného šifrovacieho jadra. Použili konkrétne 32 rôznych jadier pre každé šifrovacie kolo. To znamenalo vyšší stupeň ochrany pre šifrovací systém. Možno vidieť istú podobnosť návrhu metódam trojitého DES či AES. Každé jadro má veľkosť jeden bajt, čiže 32 rôznych jadier tvorí 32 bajtov, čo je ekvivalentné s 256 bitovou dĺžkou kľúča pri šifre AES. Vstupné dáta sa následne rozdelia do 128 bitových (16 bajtových) blokov a každý blok sa šifruje zvlášť pomocou nasledovného algoritmu:

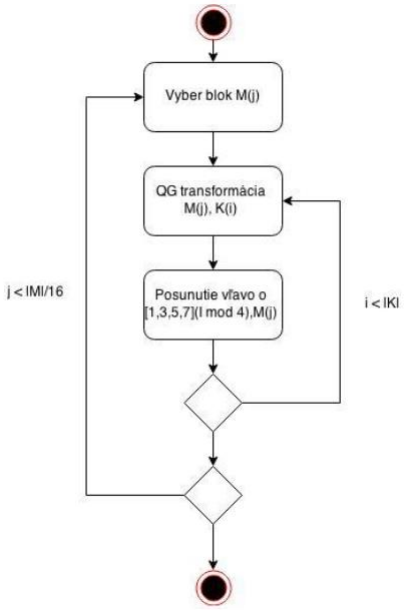
1. Vytvor kvázigrupu s rozmermi 256 x 256
2. Generuj náhodný, 256 bitový, šifrovací kľúč a rozdeľ ho do 32 rôznych 8 bitových blokov, ktoré budú použité ako jadrá v každom kole šifrovania.
3. Rozlož vstupné dáta na 128 bitové bloky.
4. Pre každý blok urob nasledovné:
   Pre každý 8 bitový blok šifrovacieho kľúča urob nasledovné:
   - Aplikuj súčasný 8 bitový kľúč ako kvázigrupové šifrovacie jadro a zašifruj podľa algoritmu kvázigrupového šifrovania daný blok tvorený prúdom 8 bitových celých čísel (prvkov kvázigupy).
   - Aplikuj blokový posun doľava o 1, 3, 5 alebo 7 bitov, podľa indexu používaného 8 bitového bloku kľúča modulo 4.

Každý blok veľkosti 128 bitov teda rozdelíme do šestnásť 1-bajtových podblokov a po každom šifrovacom kole sa všetky podbloky spoja a vykoná sa rotácia. Celá procedúra sa následne opakuje.

Pseudokód pre daný algoritmus je nasledovný:

```
BlockSize = 16, KeySize = 16
Definuj ShiftDistance ako [1,3,5,7], QGMS ako Array[256,256], Key ako Array[KeySize]
Definuj Source ako Array(N, BlockSize) , Output ako Array(N,BlockSize)
Pre každý blok:
        CipherText = Block
        Pre každé K v Key
        CipherText = QuasiGroupCipher(QGMS,K,CipherText)
        CipherText = LeftShift(CipherText,ShiftDistance[IndexOf(K, Key) modulo 4])
        Ďalšie K
Output[IndexOf(Block,Source)] = CipherText
Ďalší Block
```

Posun rovný 1,3,5 a 7 je nesúdeliteľný s číslom 2 a každé posunutie je aplikované 8 krát. Súčet posunutí sa teda rovná 128, teda veľkosti bloku, na ktorý sa rozdelia vstupné dáta. Jedna celá rotácia bloku teda využije všetkých 32 rôznych jadier. To spôsobuje, že všetky bajty v zašifrovanom bloku sú vzájomne závislé. Nasledujúca schéma znázorňuje algoritmus kvázigrupovej blokovej šifry. M je celá vstupná správa, M(j) je j-ty blok vstupnej správy, K je kľúč, K(i) je i-te jadro kľúča, |M| a |K| sú veľkosti správy a kľúča v bajtoch.



Obr. 4.4 Schéma činnosti kvázigrupovej blokovej šifry

Venovali sme sa štatistickej analýze kvázigrupovej blokovej šifry [18]. Cieľom testov bolo overiť, či šifrovacia transformácia, použitá v tejto kvázigrupovej blokovej šifre, spĺňa štandardné požiadavky na náhodnosť. Preto sme overovali, či vytváraný zašifrovaný text javí znaky náhodnej postupnosti a prechádza štandardnými štatistickými testami, používanými na overovanie štatistických vlastností kryptografických primitív. Jedným z najrozsiahlejších balíkov štatistických testov pre kryptografické účely je NIST-STS, ktorý zahŕňa množstvo testov s priamymi kryptografickými dopadmi. Tento balík bol použitý aj pri výbere štandardu AES. Bolo preto prirodzene, že sme použili práve NIST-STS. Implementácie testov z tohto balíka sú obsiahnuté v nástroji ParanoYa Academic. Navyše sú v tomto nástroji zahrnuté aj testy podľa normy FIPS 140-2.

Testovanie pozostávalo z dvoch častí, kde v každej časti bolo analyzovaných sto 1 MB postupností, vytvorených odlišným spôsobom. Prvých sto súborov tvorili postupnosti blokov samých núl zašifrovaných pomocou algoritmu kvázigrupovej blokovej šifry. Každý blok postupnosti bol zašifrovaný iným náhodne vygenerovaným kľúčom. Druhých sto súborov obsahovalo postupnosti, kde vstupné bloky pre šifrovanie boli náhodne generované a zašifrované náhodne generovanými kľúčmi.

Testovanie prebiehalo na hladine významnosti 0,01. Generátor neprešiel niektorým z testov, ak týmto testom prešlo menej ako 96 % postupností.

V nasledujúcej tabuľke vidno zhrnutie výsledkov testovania prvej sady postupností (z priestorových dôvodov uvádzame len sumárny výsledok a pre ilustráciu konkrétne výsledky prvých 5 postupností z danej sady).

Tab.4.4 Zhrnutie výsledkov štatistických testov na náhodnosť - nulové vstupné bloky

	A	B	C	D	E	F	G
1	Názov testu	Počet nesplnených postupností	Post.1	Post.2	Post.3	Post.4	Post.5
2	Frequency (monobit) test	2	0,012388	0,531274	0,002966	0,005837	0,814579
3	Frequency test within a block	0	0,254093	0,019224	0,901328	0,785062	0,087505
4	Runs test	1	0,521733	0,21862	0	0,015314	0,378147
5	Test for longest run of one in block	0	0,141643	0,774549	0,365139	0,09603	0,352686
8	Binary matrix rank test	0	0,755906	0,772782	0,608113	0,258272	0,804139
9	Discrete fourier transform (spectral) Test	0	0,559222	0,017553	0,333624	0,281413	0,795208
10	Non-overlapping template matching	1	0,479456	0,309133	0	0,021434	0,956441
11	test	1	0,486895	0,256427	0,000813	0,347571	0,694725
158	Overlapping template matching test	0	0,189053	0,960446	0,16861	0,610947	0,572669
159	Maurer's "Universal statistical" test	0	0,720914	0,372037	0,482179	0,599891	0,886412
160	Linear comlexity test	0	0,099065	0,454118	0,954129	0,600779	0,18423
163	Serial test	1	0,930486	0,051418	0	0,076153	0,679459
167	Approximate entropy test	1	0,584203	0,09903	0,000075	0,512465	0,701369
170	Cumulative Sums Test	2	0,022298	0,322502	0,004135	0,009243	0,818523
172	Random excursions test	0	0,029558	Irr	0,962063	Irr	0,694961
180	Random excursions variant test	0	0,980758	Irr	0,616733	Irr	0,502728
198	FIPS 140-2 Monobit test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
199	FIPS 140-2 Poker test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
200	FIPS 140-2 Runs test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
201	FIPS 140-2 Long runs test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED

Ako možno vidieť z výsledkov zo stĺpca B, analyzovaná bloková šifra úspešne prešla testami na náhodnosť.

V nasledujúcej tabuľke vidno zhrnutie výsledkov testovania druhej sady postupností (z priestorových dôvodov uvádzame len sumárny výsledok a pre ilustráciu konkrétne výsledky prvých 5 postupností z danej sady).

Tab.4.5 Zhrnutie výsledkov štatistických testov na náhodnosť - náhodne generované vstupné bloky

	A	B	C	D	E	F	G
1	Názov testu	Počet nesplnených postupností	Post.1	Post.2	Post.3	Post.4	Post.5
2	Frequency (monobit) test	1	0,308868	0,01929	0,987739	0,748933	0,070706
3	Frequency test within a block	0	0,134954	0,093505	0,216383	0,48409	0,016694
4	Runs test	1	0,829905	0,487239	0,178842	0,653889	0,770997
5	Test for longest run of one in block	0	0,498856	0,346683	0,748645	0,849529	0,056316
8	Binary matrix rank test	0	0,176797	0,591957	0,853797	0,637542	0,74604
9	Discrete fourier transform (spectral) Test	0	0,937934	0,346764	0,95343	0,529075	0,745603
10	Non-overlapping template matching test	1	0,967297	0,434671	0,408101	0,854154	0,914917
158	Overlapping template matching test	0	0,244717	0,925918	0,673057	0,735854	0,409098
159	Maurer's "Universal statistical" test	0	0,159704	0,521356	0,650327	0,942975	0,01601
160	Linear comlexity test	0	0,155652	0,776127	0,252802	0,085036	0,893787
163	Serial test	1	0,864986	0,934474	0,727539	0,880993	0,929506
167	Approximate entropy test	0	0,757227	0,253546	0,326516	0,732336	0,38176
170	Cumulative Sums Test	1	0,328938	0,015248	0,873754	0,654858	0,049708
172	Random excursions test	0	0,819582	0,9521	0,809527	0,329504	0,40242
180	Random excursions variant test	0	0,612835	0,211793	0,983277	0,230176	0,970673
198	FIPS 140-2 Monobit test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
199	FIPS 140-2 Poker test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
200	FIPS 140-2 Runs test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED
201	FIPS 140-2 Long runs test		ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED	ACCEPTED

Ako možno vidieť z výsledkov zo stĺpca B, analyzovaná bloková šifra úspešne prešla testami na náhodnosť aj pre druhú sadu postupností.

4.6.1 Zhrnutie výsledkov štatistických testov

Cieľom tejto časti bolo overiť štatistické vlastnosti kvázigrupovej blokovej šifry [18]. Testovanie bolo realizované v dvoch scenároch - pre nulové vstupné bloky a pre náhodne generované vstupné bloky. Ako vidno z vyššie uvedených tabuliek s výsledkami štatistických testov pre oba scenáre, analyzovaná kvázigrupová bloková šifra spĺňa štandardné požiadavky na náhodnosť, ktoré sú kladené na kryptografické primitíva.

4.7 Zhrnutie výsledkov kryptoanalýzy vybraných kvázigrupových šifier

V tejto časti sme sa venovali dvom šifrovacím schémam založeným na 3-kvázigrupách. Prinášame podrobný popis daných schém, uvádzame príklady šifrovania a navrhujeme útok na jednu zo schém. Obidve schémy vychádzajú zo spoločného základu. Majú rovnaké šifrovacie a dešifrovacie transformácie aj spôsob výberu, ktorá bude použitá. Hlavným rozdielom je spôsob použitia 3-kvázigrupy a z toho vyplývajúca veľkosť hesla.

Prvá schéma používa heslo veľkosti 5 znakov, pričom posledný znak určuje, ktorá transformácia bude použitá. 3-kvázigrupa je v tomto prípade zadaná algebricky, čo spôsobuje menšie pamäťové nároky. Táto schéma je vlastne asynchrónnou prúdovou šifrou pretože prúdový kľúč vzniká kombináciou hesla a ZT. Pri tejto schéme sme navrhli útok pri znalosti OT, ZT a dešifrovacích transformácií.

Druhá schéma využíva heslo, ktorého veľkosť je závislá od veľkosti použitej abecedy. V tejto časti sme sa venovali prípadu, kedy abeceda obsahuje 16 alebo menej prvkov. V tomto prípade je veľkosť hesla dvojnásobkom počtu prvkov v abecede plus jeden znak, ktorý určuje použité transformácie. Prvým krokom schémy je generovanie kvázigrupy pomocou daného algoritmu použitím hesla. Následne sa z kvázigrupy vygeneruje 3-kvázigrupa, ktorá je použitá pri šifrovaní respektíve dešifrovaní. Pri tejto schéme sme sa bližšie venovali procesu generovania kvázigrúp pomocou hesla. Podarilo sa nám prehľadať všetky heslá pre rády kvázigrupy 3, 4 a 5, pričom sme našli skupiny slabých hesiel, ktoré generujú rovnakú kvázigrupu. Pri vyšších rádoch sme prehľadali špeciálne určené heslá a opäť našli skupiny hesiel. Ďalej sme objavili, že skupina hesiel, ktorá generuje základnú kvázigrupu je pri rádoch 4 aj 5 najväčšia.

5 Zle iniciované generátory náhodných čísiel v sieťových zariadeniach

Mnohé zariadenia využívajúce SSL a TLS šifrovanie zvyčajne hneď po prvom spustení generujú kľúče. Po prvom spustení ale ešte nemusia mať tieto zariadenia – napr. nezapojené sieťové zariadenia ako prepínače, smerovače a pod. – dostatočnú entropiu na kryptograficky prijateľnú inicializáciu náhodného generátora. Preto sa stáva, že dve zariadenia rovnakého typu vygenerujú rovnaké kľúče alebo ich časti – napríklad rovnaké prvočíslo slúžiace pre RSA kľúč.

Výskumy dokazujú, že toto je nezanedbateľný problém pri web certifikátoch (serverových RSA kľúčoch resp. RSA kľúčoch v sieťových a serverových zariadeniach) [19,20] ako aj pri generovaní prvočísiel v chytrých „smart“ / čipových kartách, ktoré si RSA kľúč generujú priamo v čipe [21].

5.1 Ciele

Prvým cieľom tohto výskumu bolo overiť situáciu verejne dostupných SSL a TLS certifikátov a kľúčov používajúcich RSA na Slovensku – koľko RSA modulusov je rovnakých a koľko RSA modulusov má spoločné prvočíslo.

Druhým cieľom bolo zhrnúť spôsoby vytvárania entropie v zariadeniach, podľa dostupných prvkov a senzorov, resp. nutnosti používateľskej interakcie na vytvorenie dostatočnej prvotnej entropie.

5.2 Terminológia

Poznáme dva spôsoby šifrovania: symetrické a asymetrické. Symetrickým šifrovaním nazývame také, kde ten istý kľúč sa používa ako na šifrovanie, tak aj dešifrovanie. Tento kľúč zdieľajú obidve strany a poväčšine je jeho veľkosť viac ako 128 bitov. Takéto šifrovanie je rýchle, avšak nastávajú problémy s distribúciou kľúča. Medzi najznámejšie symetrické algoritmy partia: AES, DES a pod.

Problém výmeny kľúča rieši asymetrická kryptografia, alebo aj kryptografia verejného kľúča (PKC). Táto využíva pár kľúčov, jeden pre šifrovanie, druhý pre dešifrovanie. Jeden je verejný, druhý je tajný, nazývaný aj privátny. Pre takto navrhnutý systém je výpočtovo náročné získať privátny klúč z verejného kľúča. Veľkosti týchto kľúčov dnes prevyšujú 1024 bitov. Partia sem algoritmy ako RSA, Elgamal a pod.

V našom výskume sa zameriame na šifrovací systém (algoritmus) RSA. Verejný kľúč algoritmu RSA je zložený z dvoch čísel: exponentu e a modulu N. Modul N je výsledok súčinu dvoch prvočísel p a q. Ktokoľvek, kto pozná faktory N, teda p a q, vie efektívne odvodiť privátny kľúč. Ak nepoznáme p a q, je najjednoduchšou cestou vypočítať privátny kľúč faktorizáciou modulu N na p a q, a až následne vypočítať privátny kľúč.

Verejný kľúč prepojený s identifikátorom a tento vzťah potvrdený treťou (dôveryhodnou) stranou, tzv. CA, sa nazýva certifikát. Zameranie bolo na certifikáty typu ANSI X.509.

5.3 Metodika

Pre experiment boli vykonané nasledovné kroky:

1. Získanie zoznamu IP adries z RIPE, ktoré prislúchajú Slovensku
2. Skenovanie IP adries a príslušných TCP portov na dostupnosť
3. Sťahovanie certifikátov z cca. 82 TCP portov, kde sa môže SSL/TLS nachádzať
4. Lokálne vyhodnotenie medzi sebou aj s celosvetovými certifikátmi

Nedostatok celosvetového aj nášho výskumu je, že pokiaľ sa na jednej IP adrese nachádza viacero virtuálnych web serverov, tak k ich certifikátom sa nedá cez IP adresy dostať (len cez známe FQDN názvy, teda cez známe www.názov.tld). Väčšina sieťových zariadení však virtuálne web servery nemá.

5.4 Sumarizácia experimentálnych výsledkov: Stav SSL a TLS certifikátov na Slovensku

Tento experiment nadviazal na celosvetový experiment z r. 2012 a teda odzrkadľuje stav generovania privátnych RSA kľúčov najmä v sieťových zariadeniach na Slovensku 3 roky po celosvetovom výskume.

Celosvetový výskum v r. 2012 prebehol skenovaním všetkých verejných smerovateľných IP adries. Výsledky z r. 2012 boli:

- 51 mil. certifikátov nájdených globálne
- 5592 certifikátov prislúchajúcich Slovensku
- 127 tis. certifikátov globálne s RSA kľúčom malo rovnaký modulus N alebo aspoň jeden rovnaký faktor p alebo q s iným modulom-certifikátom
    - z toho 8 slovenskych certifikátov

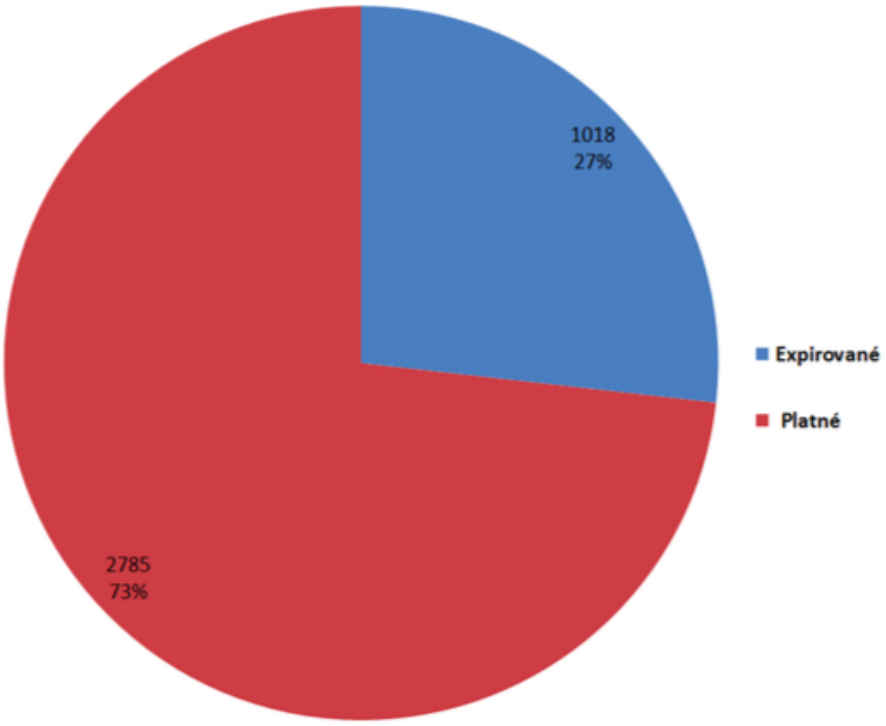
Stav generovania privátnych RSA kľúčov najmä v sieťových zariadeniach na Slovensku 3 roky po celosvetovom výskume podľa výsledkov z metodiky popísanej vyššie a vykonanom v r. 2015 je:

- 2,5 mil. oskenovaných IP adries prislúchajúcich Slovensku
- 3767 nájdených certifikátov
    - Toto číslo je menšie ako v r. 2012, pretože tu sa začalo skenovaním iba slovenských IP adries, kdežto v r. 2012 sa oskenovali všetky a až potom sa priraďovalo, ktoré certifikáty patria Slovensku. V r. 2012 boli teda zahrnuté za

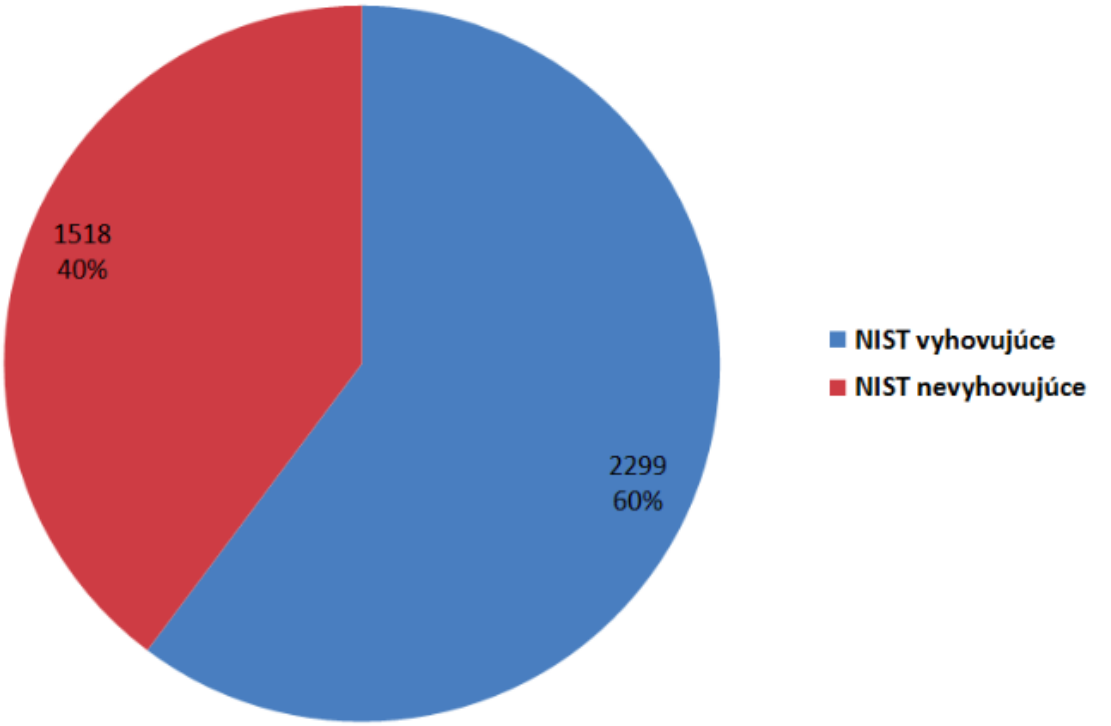
slovenské certifikáty tie certifikáty, ktoré prislúchajú TLD/doméne .sk, ale nenachádzajú sa na IP adresách a serveroch na Slovensku.

- nové(!) nie-bezpečné kľúče, ktoré majú rovnaký faktor s iným modulom. Jedná sa o nové kľúče, teda kľúče, ktoré neboli známe v r. 2012 a boli teda vygenerované po tom, a napriek tomu, že boli v r. 2012 zverejnené bezpečnostné chyby generovania RSA modulov.

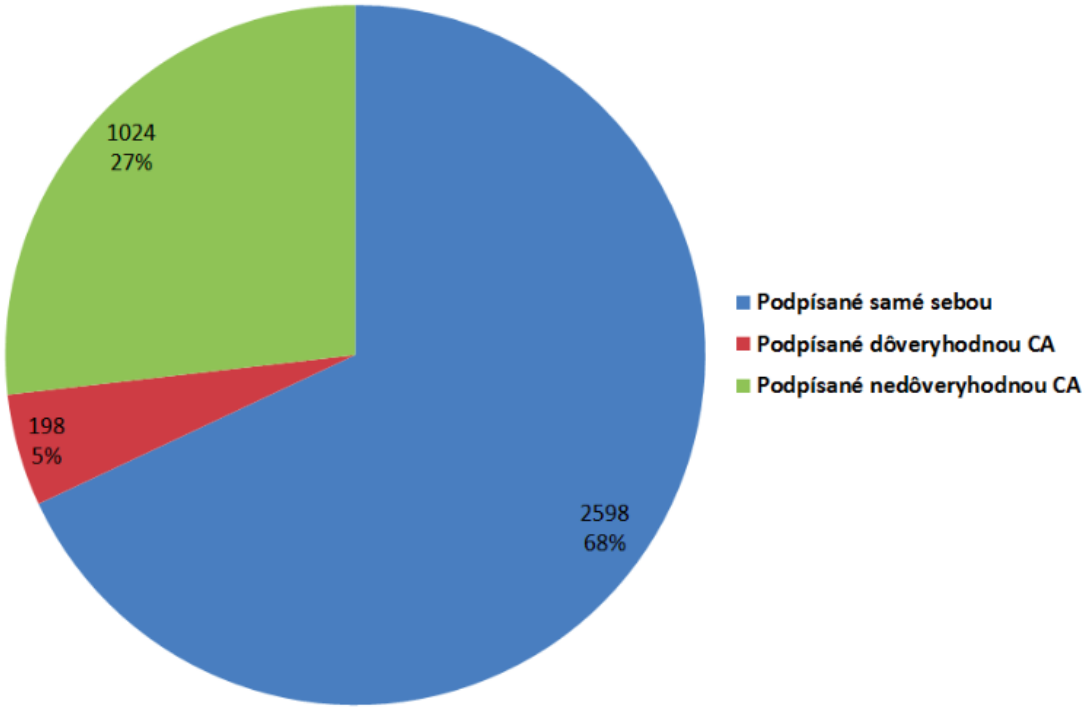
Pozn.: Experimenty boli vykonané v rámci diplomovej práce Jozefa Kudláča pod vedením Michala Šrámku.



Obr. 5.1 Platnosť verejných RSA kľúčov na Slovensku (2015)



Obr. 5.2 Dĺžky verejných RSA kľúčov na Slovensku (2015)



Obr. 5.3 Podiel slovenských certifikátov podpísaných samých sebou alebo certifikačnou autoritou

5.5 Dopad

Výskumy dokazujú, že napriek zverejneným bezpečnostným chybám v r. 2012 [19,20] a r. 2013 [21] ohľadne generovaní prvočísiel v sieťových zariadeniach v chytrých „smart“ / čipových kartách, stále neboli tieto bezpečnostné problémy odstránené.

Je nutné sa zamerať na správnu implementáciu zberu prvotnej entropie a na správnu implementáciou inicializovania náhodných generátorov slúžiacich pre generovanie prvočísel pre RSA modulus alebo iného kľúčového materiálu.

5.6 Odporúčania

V prípade prvotného spustenia, či už sieťových prvkov, mobilov, čipových kariet alebo akýchkoľvek zariadení je potrebné zvážiť, či sa dá entropia (potrebná pre kryptografický náhodný generátor slúžiaci na generovanie kľúčového materiálu) pozbierať automaticky z dostupných zdrojov alebo je nutná používateľská interakcia na vytvorenie tejto entropie.

5.6.1 Používateľom vytvorená entropia

Príklady:

- Písanie na klávesnici – sekvencie, časové dĺžky medzi jednotlivými znakmi, a pod.
- Pohyb myšou – rýchlosť, smer, časy pohybu, a pod.
- Pohyb zariadením (napr. mobilom) – využitie senzorov: rýchlomer (accelerometer), gyroskop, a pod. [22]

5.6.2 Automaticky vytvoriteľná entropia

Príklady:

- Sila signálov (GSM, WiFi, GPS, ...)
- Prijaté signály – dátové sekvencie, štatistiky, časové dĺžky medzi sekvenciami/samotných sekvencií, a pod.
- Štatistiky a merania z dostupných senzorov – napr. batérií (odber), teploty procesora, teploty okolia, a pod. ale aj opätovné využitie rýchlomeru alebo gyroskopu pri pohybe zariadenia napr. v aute.
- Obsah pamäte, disku, procesov a pod. (ale len v prípade, že zariadenie prijalo alebo príjma signály alebo malo používateľskú interakciu)

6 Generovanie náhodných čísiel v zariadeniach s obmedzeným výkonom alebo pamäťou

Príspevok k štúdiu lightweight kryptografie (LWC). LWC sa pokúša o optimalizáciu pomeru medzi bezpečnosťou a implementačnými požiadavkami na šifru smerom k šetreniu dostupného výpočtového výkonu na úkor niektorých bezpečnostných rizík.

Mnohé zariadenia zvyčajne hneď po prvom spustení generujú šifrovacie a-alebo podpisovacie kľúče. Po prvom spustení ale ešte nemusia mať tieto zariadenia – napr. nezapojené sieťové zariadenia ako prepínače, smerovače a pod. – dostatočnú entropiu na kryptograficky prijateľnú inicializáciu náhodného generátora. Preto sa stáva, že dve zariadenia rovnakého typu vygenerujú rovnaké kľúče alebo ich časti – napríklad rovnaké prvočíslo slúžiace pre RSA kľúč. Výskumy dokazujú, že toto je nezanedbateľný problém.

V súvislosti s kapitolou 5 sme sa zamerali na vytvorenie generátora náhodných čísiel, ktorý by nemal nedostatky vyplývajúce z nedostatku entropie. Takýto generátor by mal byť schopný pracovať aj na zariadeniach s obmedzeniami, či už na zariadení s obmedzeným výkonom (výpočtovou kapacitou a silou) a/alebo obmedzenou pamäťou.

6.1 Terminológia

Generátorom náhodných čísel nazývame algoritmus, ktorého výstupom je postupnosť čísel – najčastejšie bitov, ktoré sa javia ako „náhodné“ pre každého pozorovateľa. Definovať náhodnosť je pomerne náročné. Najčastejšie sa stretávame s pojmami ako nepredvídateľné, alebo rovnomerne rozdelené. Tieto pojmy popisujú vlastnosti, ktoré náhodné čísla musia nutne mať.

Náhodný generátor čísiel (true RNG) je algoritmus, ktorého výstupom sú čísla a kde ďalšie číslo je rovnako pravdepodobné a teda nezávislé od predchádzajúcich.

Pseudonáhodný generátor (PRNG) je algoritmus, ktorého výstupom je postupnosť náhodných čísiel odvodená od počiatočnej hodnoty – tzv. seed.

Entropia, tak ako ju zaviedol Claude E. Shannon, je mierou neurčitosti medzi vstupom a výstupom údajov.

6.2 Metodika a vykonané úlohy

Aj na základe odporúčaní z predchádzajúceho výskumu, popísaného v kapitole 5, sme zvolili nasledovný postup:

1. Identifikácia a analýza metód a techník na zber entropie v zariadeniach
2. Analýza možného dopadu aplikácie vybraných metód a techník na zariadenia s obmedzeniami

3. Špecifikácia požiadaviek na zber údajov z mobilov
4. Návrh vybraných metód na zber entropie fungujúcich na zariadeniach s obmedzenými zdrojmi
5. Implementácia prototypu zbierajúceho entropiu na Android zariadení
    a. zo sieťovej prevádzky,
    b. z operačného systému a aplikácií,
    c. z rádiových signálov, a
    d. z polohových a iných senzorov
6. Testovanie a experimentovanie s implementovanými prototypmi
7. Meranie entropie z testov, zo získaných údajov
8. Vyhodnotenie experimentov

Testovanie sme navrhli pre chytré telefóny na platforme OS Android. Táto voľba nám dáva možnosť interpretácie výsledkov na podobných systémoch a platformách, od Unix-u, Java virtuálnych mašín, cez chytré telefóny s obmedzeniami, až po počítače, servery a zariadenia s plným výpočtovým výkonom.

Tabuľka 6.1 uvádza zdroje náhodných údajov, ktoré sme v rámci výskumu identifikovali.

Tab.6.1 Možné zdroje náhodných údajov

Typ zdroja	**Možné zdroje náhodných údajov**
Sieťová prevádzka	• Diferencie časov (milisekúnd) medzi príjmanými a/alebo posielanými správami alebo paketmi • Počty prijatých a/alebo odoslaných paketov za jednotku času • Veľkosti prijatých a/alebo odoslaných paketov za jednotku času • Iné štatistiky ohľadne prijatých a/alebo odoslaných paketov
Operačný systém a aplikácie	• Systémové logy • Aplikačné logy • Súbory – vrátane ich charakteristík, práva, veľkosti súborov, časy vytvorenia, časy modifikácií, a pod.
Rádiové signály	• Podľa typu vysielania (frekvencie, modulácie, a pod): GSM 2G/3G/4G, WiFI a/b/g/n/ac, Bluetooth, Infrared, aNFC, GPS, Glonass • Sila signálu v danom čase • Informácie ohľadne okolitých alebo pripojených zariadeniach,

	ohľadne aktívnych prvkov siete • Rôznorodé štatistiky z prijatých a/alebo odoslaných údajov
Polohové a iné senzory	• Podľa typu senzoru: rýchlomer, gyroskop, proximity, kamera, fitness senzory, polohové senzory (GPS, Glonass), a pod. • Rôznorodé štatistiky (počty, priemery, diferencie, atď.) zo získaných údajov
Používateľská interakcia	• Pohyb myšou – rýchlosť, lokácia, smerovanie, časy medzi pohybmi, a pod. • Klávesnica – rýchlosť písania, časové odstupy, samotný text • (Multi) dotyková obrazovka – pohyby, smer, veľkosť dotykov, príp. aj sila stisku ak to podporuje HW

6.3 Merania entropie a náhodnosti v praxi

Entropia, tak ako ju zaviedol Claude E. Shannon, je mierou neurčitosti medzi vstupom a výstupom údajov. Entropia *H(X)* pre diskrétnu náhodnú premennú *X* s pravdepodobnostným rozdelením p_i je

$$H(X) = -\sum_i p_i \log_2(p_i).$$

V praxi, pre 8-bitové bajty, sa používa logaritmus s bázou 256. Pre reťazec znakov (bajtov, ASCII) dĺžky `length` sa entropia vypočíta nasledovne. Nech `bytecount` je pole, ktorého hodnoty sú počty jednotlivých ASCII znakov v reťazci, a teda dĺžka reťazca je suma cez toto pole. Potom entropia reťazca je, pseudokódom:

```
entropy = 0
for i = 0 to 255
      if bytecount[i] != 0 then
           p = bytecount[i] / length
           entropy += p * log(p, 256)
```

Mieru náhodnosti reťazcov (súborov) môžeme tiež efektívne merať cez kompresiu. Ak algoritmus *compress* je ideálny kompresný algoritmus, teda poskytuje maximálnu kompresiu súboru, tak entropia súboru je pomer

$$\frac{veľkosť\ súboru\ po\ maximálnej\ kompresii}{veľkosť\ súboru\ bez\ kompresie}$$

Teda v praxi pre priblíženie stačí použiť zip, gzip alebo bzip2 na výpočet približnej entropie. Aj keď to nebude presná entropia, na porovnanie experimentov postačuje. Čím väčšia neurčitosť, teda čím väčšia entropia, tým väčšia hodnota údajov pre PRNG seed.

6.4 Výsledky merania entropie a náhodnosti v praxi

Cieľom bolo z automaticky (z prevádzky alebo senzorov) alebo manuálne (od používateľa) získaných údajov *data* vytvoriť *seed*, teda dostatočne dlhý, náhodný a neurčitý inicializačný vstup do kryptograficky bezpečného PRNG.

Tab.6.2 zobrazuje namerané náhodnosti (odhadovaná entropia) reťazcov z vybraných zdrojov.

Pre praktické použitie sme navrhli nasledovné metódy. Pokiaľ odhadujeme entropiu komprimáciou, získame aj skomprimované údaje. Ďalej uvažujeme už len skomprimované reťazce *compress*(*data*), bez hlavičky a iných podporných údajov. Pokiaľ skomprimované údaje nemáme, požijeme priamo metódu č. 4, s tým rizikom, že možno nemáme dostatočnú entropiu:

1. Skomprimované reťazce veľkosti pre seed sa dajú priamo použiť pre seed:

   seed := *compress*(*data*)

2. Skomprimované reťazce dlhšie ako veľkosť pre seed je vhodné hašovať a tak získať kombináciu všetkých bitov:

   seed := *hash*(*compress*(*data*))
3. Skomprimované reťazce kratšie ako veľkosť pre seed nie sú kryptograficky bezpečné:

   seed := *compress*(*data*) + ďalšie údaje!
4. Pôvodné (neskomprimované) reťazce je vhodné zahašovať cez kryptograficky bezpečnú šifru na samotný seed pre PRNG:

   seed := *hash*(*data*)

Tab.6.2 Nameraná náhodnosť z vybraných zdrojov

Typ zdroja	**Vybraný zdroj**	**Nameraná náhodnosť**	**Veľkosť po kompresii**
Systémové a aplikačné logy	logcat za dobu 2 sekundy	0,0582	~32 kB
Sieťová prevádzka	Vzorkovanie netstat, 5-krát, s 1 s odstupmi	0,1034	~512 B

6.5 Odporúčania

Aby sa nestalo, že dve zariadenia rovnakého typu vygenerujú rovnaké kľúče alebo ich časti – napríklad rovnaké prvočíslo slúžiace pre RSA kľúč, je nutné, aby aj nové predtým neiniciované zariadenia nazbierali dostatočné množstvo entropie na inicializáciu PRNG pred samotným generovaním kľúčov. Typ informácií pre inicializáciu PRNG záleží na dostupných prvkoch a senzoroch zariadenia a v prípade, že dostatočná entropia sa nedá dosiahnuť automaticky je nutné doplniť túto chýbajúcu entropiu od používateľa.

Tieto odporúčania sú platné ako pre počítače, servery, sieťové zariadenia (ako prepínače, smerovače, firewally, a pod.) tak aj pre chytré telefóny, vnorené systémy, IoT zariadenia, a smart metre.

6.5.1 Najvyššia praktická bezpečnosť

Pokiaľ chceme dosiahnuť najvyššiu bezpečnosť, musíme iniciovať PRNG seedom, ktorý má dostatočnú entropiu. Pre najvyššiu bezpečnosť je preto najlepšie vyzbierať entropiu zo všetkých dostupných zdrojov, teda kombináciou všetkých zdrojov z Tab.6.1 a zberom za čo najdlhší čas. Takto zozbierané údaje nie je nutné spracovávať, filtrovať, alebo komprimovať. Kryptograficky bezpečná haš (hash) funkcia zo všetkých skombinovaných údajov-súborov dokáže ľahko vytvoriť seed pre PRNG.

6.5.2 Teoretická bezpečnosť

V prípade použitia true RNG výstup true RNG spĺňa, že každý ďalší bit alebo číslo má rovnakú pravdepodobnosť byť vygenerované a teda je nezávislé od predchádzajúcich. Napriek tomu výstup z true RNG nespĺňa štatistické testy vyžadované pre kryptograficky bezpečné náhodné postupnosti. Preto je vhodné výstup z true RNG ešte pričítať (XOR v prípade bitov resp. modulo bázy) s výstupom z kryptograficky bezpečného PRNG.

6.5.3 Optimálna praktická bezpečnosť

Optimálna bezpečnosť pre praktické použitie, aj v rámci konceptu LWC.

Najvhodnejšie je použiť používateľom generované údaje, či už priamo alebo prostredníctvom aplikačného alebo systémového logovania. Ostatné zdroje je ťažké ovplyvniť - sieťová prevádzka, rádiové signály a ich dostupnosť, alebo systémové a aplikačné súbory napr. hneď po inštalácii a prvom spustení.

Praktické experimenty poukazujú na to, že už 1 sekunda systémových logov, tzv. Android ‘logcat’, postačujú na zber dostatočnej entropie (pre všetky prakticky používané PRNG a ich veľkosti seed). V kombinácii s používateľskou interakciou počas zberu údajov z logcat sú nazbierané údaje prakticky neopakovateľné. Samotná používateľská interakcia nemusí byť

nič zložité ani dlhé, napr. použitie senzorov – gyroskop (pohyb mobilom), rýchlomer (akcelerometer, pohyb s mobilom), alebo jednoduchá interakcia s dotykovou obrazovkou počas.

7 Zoznam použitej literatúry

[1] Genkin Daniel, Shamir Adi, Tromer Eran: RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis, rev. Advances in Cryptology – CRYPTO 2014, Springer Berlin Heidelberg, 2014, pp. 444 - 461.

[2] Mangard, Stefan, Oswald, Elisabeth, Popp, Thomas, Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer US, 2007.

[3] S. Skorobogatov: „Side-channel attack: new direction and horizons“, Design and Securityof Cryptographic Algorithms and Devices(ECRYPTII), Albena, 2011.

[4] EBV, „http://www.ebv.com“, 20 Jún 2013. [Online]. Available: http://www.ebv.com/download/?file=fileadmin/relaunch/npi_data/files/5797/SoCrates_Datasheets_V1.2.pdf&no_cache=1.

[5] F. Uhrecký, „github.com“, 2014. [Online]. Available: https://github.com/FrUh/BitPunch.

[6] P. Technology, „picotech.com“, 2012. [Online]. Available: https://www.picotech.com/download/manuals/PicoScope6000ABSeries-UsersGuide-en-1.pdf.

[7] Ondráček, O.: Signály a sústavy. Edícia vysokoškolských učebníc. Vydavateľstvo STU Bratislava, 3. vydanie 2008, 341 str., ISBN 978-80-227-2956-7.

[8] S. Heyse, A. Moradi, C. Paar: „Practical Power Analysis Attacks on Software Implementations of McEliece“, PQCrypto10, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings

[9] Gligoroski, D. - Markovski, S.: Cryptographic potentials of quasigroup transformations. Institute of Informatics, Skopje, Macedonia. 13s, 2003

[10] Petrescu, A.: Applications of quasigroups in cryptography. "Petru-Maior" University of Târgu-Mures, Romania. 2007, 5s, http://www.upm.ro

[11] Chakrabarti, S. - Pal, S. - Gangopadhyay, S.: An Improved 3-Quasigroup based Encryption Scheme. Indian Statistical Institute, India., 12s, 2012, ISBN: 008021680, http://ictinnovations.org/2012/htmls/papers/icti2012_submission_19.pdf

[12] Petrescu, A.: n-Quasigroup cryptographic primitives: stream ciphers, vol. LV. Studia Univ. Babes(Bolyai, Informatica), 2010

[13] Dvorský, J. - Ochodková, E. - Snášel, V.: Hash functions based on large quasigroups. Proc. of Velikonočná kryptologie, Brno. 2002

[14] Dvorský, J. - Ochodková, E. - Snášel, V.: Hashovací funkce založená na kvázigrupách. Mikulášská kryptobesídka, Praha. ,10s, 2001

[15] Krapež, A.: An application of quasigroups in cryptology, Math. Maced., vol. 8., 6s, 2010

[16] Shcherbacov, V.: Quasigroups in cryptology, vol. 17. Computer Science Journal of Moldova., 2009, 36s

[17] Shcherbacov, V.: On some known possible applications of quasigroups in cryptology, 2003, http://www.karlin.mff.cuni.cz/~drapal/krypto.pdf, 15s

[18] Battey, M. – Parakh, A.: An Efficient Quasigroup Block Cipher. Springer Science+Business Media, New York, 2012

[19] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: Detection of widespread weak keys in network devices," USENIX 2012.

[20] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, whit is rightg,", IACR e-print archive, 2012 --- a --- "Public keys," CRYPTO 2012.

[21] D.J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, N. van Someren, "Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild," ASIACRYPT 2013.

[22] V. Hromada, J. Varga, "Entropy assessment of Android OS," ISCAMI 2014.