MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Automatické hledání závislostí u proudových šifer projektu eStream

DIPLOMOVÁ PRÁCA

Matej Prišťák

Brno, Jaro 2012

Prehlásenie

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: Petr Švenda

Poďakovanie

Rád by som poďakoval v prvom rade vedúcemu práce RNDr. Petrovi Švendovi, Ph.D. za odborné vedenie, pravidelné zhodnotenie a pripomienky k práci. Poďakovanie patrí aj kolegovi Ondrejovi Dubovcovi za spoluprácu pri úpravách použitej aplikácie. Nakoniec by som chcel poďakovať celej mojej rodine a priateľke Zuzke za podporu a trpezlivosť počas písania tejto práce.

Zhrnutie

Cieľom práce je príprava existujúcich implementácií širšieho okruhu kandidátov na prúdovú šifru v rámci projektu eSTREAM do jednotného rozhrania tak, aby bolo možné automatické hľadanie nežiadúcich závislostí vo výstupe (ktoré by sa nemali vyskytovať) všetkých týchto kandidátnych funkcií naraz. Pre automatické hľadanie bol využitý existujúci nástroj využívajúci techniky evolučných obvodov. K dispozícii bola možnosť spúšťať výpočty distribuovane pomocou nástroja BOINC s využitím laboratórnych počítačov. Implementačná časť je doplnená analýzou a diskusiou získaných výsledkov a porovnaním úspešnosti hľadania závislostí evolučným obvodom verzus závislosti nájdené bežnými batériami pre štatistické testovanie výstupu dát (Diehard a NIST).

Kľúčové slová

genetický algoritmus, genetic algorithm, prúdová šifra, stream cipher, ECRYPT, eSTREAM, kryptoanalýza, cryptoanalysis, štatistické testovanie, statistical testing

Obsah

Kapitola 1

Úvod

Ľudia od nepamäti túžili po možnosti písať dôverné texty tak. Dalo by sa povedať, že kryptografia je stará, ako písmo samé. To isté platí aj o kryptoanalýze, pretože tak isto túžili ludia aj po možnosti prelomiť dôvernosť textu a prečítať si, čo nebolo určené im. V modernej ére význam kryptografie a kryptoanalýzy prudko stúpol, pretože objem dôverného textu sa dramaticky zvýšil.

Prúdové šifry naproti tomu sú pomerne mladé. Za prvú prúdovú šifru sa dá považovať One-Time Pad. Ich význam stúpol až v posledných rokoch, kedy sa objavila potreba šifrovania dát tzv. online. Bežne sa prúdové šifry začali používať až v 90-tych rokoch s nástupom GSM sietí, kde dominuje šifra A5. U wi-fi sietí a SSL sa zas presadila šifra RC4.

S rozmachom online služieb sa objavila nutnosť ďalšieho vývoja prúdových šifier. RC4 aj A5 boli vymyslené v 80-tych rokoch minulého storočia a ich bezpečnosť nie je v momentálnych podmienkach dobrá. Z tohoto dôvodu prišiel projekt eSTREAM. Jeho úlohou bolo predstaviť nové princípy návrhu prúdových šifier.

Klasická kryptoanalýza pozná širokú škálu útokov. Napriek tomuto faktu sa nedá šifra považovať za bezpečnú ani po otestovaní všetkých známych útokov. Mať možnosť hľadať nové útoky na šifry automatizovane by bola pre kryptoanalytikov veľká pomoc.

Naším primárnym cieľom bolo upraviť už existujúci nástroj pre evolúciu obvodov tak, aby sa dal použiť pre testovanie prúdových šifier projektu eSTREAM. Ďalším cieľom bolo navrhnúť také testy, ktoré by boli schopné dobre otestovať kandidátov. Porovnanie s inými výsledkami je dôležité, a preto sme sa rozhodli, že naše výsledky budeme porovnávať s výsledkami štatistických testov. Posledným cieľom bolo samotné spustenie testov paralelne pomocou architektúry BOINC a spísanie dosiahnutých výsledkov do textu tejto práce spolu s ich zhrnutím.

Prvá časť práce popisuje teóriu stojacu za genetickými algoritmami, evolučnými obvodmi a štatistickým testovaním. Druhá časť popisuje projekt eSTREAM, jeho kandidátne šifry s dôrazom na kryptoanalytické výsledky. Tretia časť sa venuje našej testovacej aplikácii, hlavne jej návrhu a popisu dôležitých častí. Štvrtá a posledná časť popisuje testy, výsledky a ako sme s k nim dospeli. Venuje sa testom štatistickým aj testom vykonanými našou testovacou aplikáciou.

Kapitola 2

Definícia pojmov

V prvej kapitole popisujeme teóriu stojacu za genetickými algoritmami, evolúciou a evolučnými obvodmi. Ďalej popisujeme základy štatistického testovania.

2.1 Genetický algoritmus

Genetický algoritmus je algoritmus, ktorého úlohou je nájsť riešenie zložitého problému, pre ktorý nie je ľahké nájsť exaktný algoritmus. Využíva k tomu metódy napodobňujúce biologické procesy, ako dedičnosť, prirodzený výber, kríženie a mutácia. Základom genetického algoritmu je populácia, ktorá sa časom mení tzv. evolúciou. Populácia môže obsahovať rôzny počet jedincov. Na začiatku bývajú jedinci zvolení náhodne. Krokom evolúcie nazývame použitie operácií kríženia a mutácie na existujúcich jedincov populácie, následné ohodnotenie pomocou tzv. fitness funkcie a vybratie tých najschopnejších (najlepšie ohodnotených) jedincov pre ďalšie kolo genetického algoritmu. Existuje veľa rôznych knižníc využívajúcich genetické algoritmy. Nami použitá a asi najznámejšia knižnica je GAlib vyvinutá na MIT v 90-tych rokoch.

2.1.1 Initializer

Initializer je funkcia, ktorá nastaví počiatočné hodnoty jedinca. Môžeme ho prirovnať ku konštruktoru v triedach.

2.1.2 Fitness

Základnou funkciou pre funkčnosť genetického algoritmu je tzv. fitness funkcia. Jej hlavnou úlohou je vrátiť algoritmu číselne ohodnoteného jedinca. Toto ohodnotenie sa použije pre porovnanie s ostatnými jedincami. Táto funkcia musí byť rýchla aj preto, že v každom kroku sa vyhodnucuje každý jedinec celej populácie zvlášť. Počet volaní tejto funkcie pri hodinovom behu nášho programu môže presiahnuť aj číslo milión.

2.1.3 Mutator

Funkcia, ktorá určuje, ako presne sa má odvodzovať nový jedinec zo starého. V prípade zložitých jedincov (náš prípad) je nutné túto funkciu definovať. Základnou myšlienkou je obmena jednotlivých častí jedinca.

2.1.4 Crossover

Kríženie je proces, pri ktorom sa časť jedinca vymení s inou časťou iného jedinca.

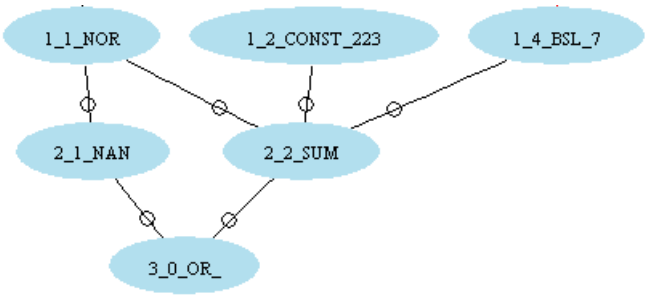
2.2 Evolučný obvod

Pre naše účely bolo nutné správne sformulovať úlohu pre genetický algoritmus. Na základe predchádzajúcich skúseností doktora Švendu s používaním genetických algoritmov pre podobný účel, sme sa rozhodli použiť evolučný obvod, ako jedinca v populácii. Evolučný obvod pozostáva z viacerých vrstiev. Prvá vrstva vždy obsahuje vstupy. Do nej sú zapísané všetky vstupy pri vyhodnocovaní pomocou fitness funkcie. Nasleduje niekoľko vnútorných vrstiev obvodu, ktoré určujú jedinca. Obsahujú vždy elementárne funkcie, ktoré môžu upraviť vstup a predať ho na výstup. Jednotlivé vrstvy obvodu sú spojené medzivrstvami. Tie určujú, aké hodnoty sa použijú ako vstupy do funkcií uložených na vrstve. Výsledky sú zase poslané ďalšou medzivrstvou ako vstupy do ďalšej vrstvy. Posledná vrstva je vždy výstupná. Na nej očakávame od obvodu výsledok podľa zadania. Tento výsledok ohodnocuje fitness funkcia.

Samotnú reprezentáciu evolučného obvodu, ako jedinca v genetickom algoritme, sme zvolili jednoduché pole celých čísiel. Každá funkcia v ňom má jednoznačný identifikátor. Tak isto sú v poli zapísané prepojenia jednotlivých vrstiev, ako bitové polia. Maximálna veľkosť jednej vrstvy je preto 32 prvkov pri štandardnej veľkosti celého čísla (32 bitov). Pole je organizované tak, ako výsledný grafický obvod, tj. najprv je v ňom zapísané jedno číslo pre každý prvok vrstvy reprezentujúcej funkcie. Následne je zapísané jedno číslo pre každý prvok mrdzivrstvy označujúce prepojenia medzi vrstvami. Vstupy sú pevne dané a teda nie sú zapisované. Základné funkcie použité vnútri obvodu sú:

- NOP - žiadna funkcia - svoj vstup predá priamo na výstup
- OR - vykoná na vstupoch operáciu disjunkcie (logický súčet) a výsledok predá na výstup
- AND - vykoná na vstupoch operáciu konjunkcie (logický súčin) a výsledok predá na výstup
- CONST - predá na výstup konštantu
- XOR - vykoná na vstupoch operáciu exkluzívného logického súčtu a výsledok predá na výstup
- NOR - vykoná na vstupoch tzv. Peirceovu funkciu a výsledok predá na výstup
- NAND - vykoná na vstupoch tzv. Shefferovu funkciu výsledok predá na výstup
- ROTL - vykoná na vstupoch bitovú rotáciu vľavo a výsledok predá na výstup
- ROTR - vykoná na vstupoch bitovú rotáciu vpravo a výsledok predá na výstup

- BITSELECTOR - vykoná na vstupoch bitový súčin a výsledok predá na výstup
- SUM - vykoná na vstupoch súčet a výsledok predá na výstup
- SUBS - vykoná na vstupoch odčítanie a výsledok predá na výstup, základ je výsledok funkcie na rovnakej pozícii z predchádzajúcej vrstvy
- ADD - vykoná na vstupoch súčet a výsledok predá na výstup, rozdiel oproti SUM je v tom, že ADD ako prvý vstup vezme výsledok funkcie na rovnakej pozícii z predchádzajúcej vrstvy
- MULT - vykoná na vstupoch operáciu násobenia a výsledok predá na výstup
- DIV - vykoná na vstupoch celočíselného delenia a výsledok predá na výstup, delenec je výsledok funkcie na rovnakej pozícii z predchádzajúcej vrstvy
- READX - špeciálna funkcia, ktorá umožní obvodu prečítať zo vstupu hodnotu s indexom vstupu funkcie



Obr. 2.1: Príklad časti obvodu

Číselná reprezentácia obvodu by vyzerala takto:

```
0 5 3 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 22 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 6 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

3, 5, 6, 9 a 10 sú kódy funkcií CONST, NOR, NAND, BITSELECTOR a SUM. 2, 22 a 6 sú bitové mapy označujúce prepojenie s druhým prvkom (2^1), druhým, tretím a piatym prvkom (2^1 + 2^2 + 2^4) a druhým a tretím prvkom (2^1 + 2^2).

2.2.1 GA na evolučnom obvode

Evolučný obvod kvôli svojej zložitosti nemôže používať štandardnú množinu evolučných funkcií. Inicializácia obvodu prebieha náhodným rozdelením funkcií do vrstiev. Fitness funkcia vyhodnocuje celý obvod na všetkých dostupných vstupoch, väčšinou teda spúšťa samotný obvod veľakrát. Mutácia vymieňa funkcie v obvode za náhodné funkcie. Taktiež

dokáže pridať, alebo odobrať spojenie medzi vrstvami. Body kríženia sa vyberajú vždy len na celé vrstvy, takže umožnujeme vymeniť celé vrstvy medzi jednotlivými génami. Mutácia aj Crossover sa dejú na základe nastaviteľnej pravdepodobnosti pre mutáciu a kríženie.

2.3 Random Generator

Pre správny chod genetického algoritmu je nutný dobrý generátor (pseudo)náhodných čísiel. Pre našu aplikáciu to platilo dvojnásobne, pretože akákoľvek závislosť vstupu sa mohla premietnuť do výstupu a ovplyvniť výsledky. Použili sme preto viac typov generátorov. Štandardne používaný systémový zdroj náhodnosti je závislý na čase a platforme. Je preto použitý len ako posledná možnosť, keď iné zdroje nie sú k dispozícii. Najlepším zdrojom náhodnosti je kvantový generátor náhodných čísiel. Pre naše účely stačili aj predgenerované dáta z kvantového generátora uložené v súboroch[1]. Na druhú stranu sme potrebovali aj zdroj, ktorý by bol ovplyvniteľný a určitým spôsobom predvídateľný. Preto sme vytvorili tzv. bias generator, ktorý generuje bity s určitou nastaviteľnou pravdepodobnosťou.

2.4 Štatistické testovanie

Štatistické testovanie pseudonáhodného generátora (PRNG) sa používa pre zistenie chýb náhodnosti a predvídateľnosti v ním vygenerovanom prúde dát. Ako najjednoduchší príklad štatistického testu si môžeme predstaviť spočítanie 1 a 0 v bitovom prúde a porovnaniu ich počtu. Dobrý PRNG by mal mať rozloženie približne 50:50. Testovanie šifier na náhodnosť a predvídateľnosť je založené na predpoklade, že o zašifrovanom prúde dát (texte) by sme by sme nemali byť schopní povedať vôbec nič - mal by byť teda pre nás nerozlíšiteľný od náhodných dát.

Teoreticky sa štatistické testovanie zakladá na tzv. null hypothesis. Ňou označujeme nedokázateľný predpoklad. V našom prípade by sme ho mohli formulovať, ako testovaný prúd dát je náhodný. Ďalej máme alternatívny predpoklad (tzv. alternate hypothesis), ktorý je jeho negáciou - testovaný prúd nie je náhodný. Pre každý vykonaný test dôjdeme k nejakému záveru - buď je prúd náhodný, alebo nie je. Rozhoduje sa na základe tzv. critical value. Tá určuje hranicu medzi výsledkami prijateľnými a neprijateľnými pre daný test. Kritická hodnota je nastavená vždy tak, aby ju aj naozaj náhodné dáta občas prekročili. Toto by sa ale malo stávať v minime prípadov. Rozdiel medzi naozaj náhodnými dátami a nie náhodnými dátami je práve v počte prekročení kritickej hodnoty.

Pre testovanie sme použili batérie NIST a DIEHARD. Popisy konkrétnych testov nájdeme v Oddiel 5.1.

1. Boli získané z kvantového generátora zo stránky Department of Physics, Humboldt Universitaet zu Berlin - <https://qrng.physik.hu-berlin.de/>

Kapitola 3

eSTREAM

V tejto kapitole popisujeme projekt eSTREAM, jeho účastníkov s dôrazom na kryptoanalytické výsledky a taktiež použitie pre naše testovanie.

Projekt eSTREAM vznikol v roku 2004, jeho úlohou bolo nájsť nové prúdové šifry. V prvom kole v roku 2005 bolo získaných 34 rôznych návrhov šifier. Tieto boli ďalej rozvíjané, testované a v ďalších kolách vyraďované z portfólia eSTREAM. V roku 2008 zostalo po 3. kole 7 šifier, ktoré boli vybraté, ako najlepšie. Základnou požiadavkou na nové šifry bolo, aby mali aspoň jednu vlastnosť lepšiu oproti AES, ktorá bola použitá, ako základná šifra pre porovnávanie. Nasleduje základný popis šifier, ktoré sa zúčastnili projektu eSTREAM a boli voľne dostupné pre použitie.

3.1 ABC

Synchrónna prúdová šifra, ktorá bola považovaná za jednu z najrýchlejších z celého portfólia. Po prvom kole bola zlomená pomerne jednoduchým útokom typu rozdeľuj a panuj [4]. Do druhého kola bola odovzdaná upravená verzia, ktorá ale negenerovala dostatočne náhodný výstup bitov[30]. Tretia verzia bola taktiež úspešne napadnutá [32] a šifra vyradená z portfólia kvôli chybám v návrhu šifry. Táto nedokonalosť by nám mohla poslúžiť pri testovaní a vyvíjaní evolučného obvodu, pretože by pre neho malo byť jednoduchšie nájsť konkrétnu závislosť u tejto šifry. Použili sme poslednú verziu (ABCv3), ktorá využívala 128-bitové kľúče a inicializačné vektory. Šifru bolo možné obmedziť na menší počet kôl (1-16) bez úprav. Projektovaná ale bola tak, aby fungovala aj pri 1 kole. Bohužial šifra aj po veľkej snahe dávala na výstup rovnaké prúdy a preto sme ju pri testovaní nepoužili.

3.2 Achterbahn

Aditívna bitová šifra Achterbahn je náchylná na tzv. key-recovery útoky. Boli ukázané spôsoby, akými sa dá získať kľúč v čase 2^80 a bola preto vyradená z ďalšieho priebehu súťaže [14] [24]. Pre nás neposkytovala šifru jednoduchú možnosť oslabenia.

3.3 CryptMT

Proti tejto šifre neboli nájdené žiadne útoky. Toto môže byť spôsobené neobvyklým návrhom šifry. Autori finálového reportu [3] projektu si neboli istí dostatočným otestovaním šifry a preto ju nezaradili do finálového výberu šifier. Ďalšia zo šifier, ktoré nejde efektívne zoslabiť. Navyše je to jedna z troch šifier, ktorú sa nám nepodarilo rozchodiť, takže sme s ňou ďalej nepracovali.

3.4 DECIM

Prvá verzia bola úspešne napadnutá [29] a následne kompletne prerobená pred vstupom do 2.fázy projektu. Odvtedy už prelomená nebola, do finálového výberu ale nepostúpila kvôli rýchlosti. Je možné ju oslabiť na menší počet kôl (1-4). Obmedzenie bolo možné na základe toho, že počas výpočtu sa opakuje posúvanie registrov v jednom kroku 4-krát.

3.5 DICING

Pomerne nezaujímavá šifra neposkytujúca oproti AES žiadnu výhodu. Nepostúpila preto do 3.fázy. Z nášho pohľadu neumožnovala šifra jednoduché obmedzenie tak, aby bolo možné ju rozumne škálovať a testovať rôzne oslabené výstupy.

3.6 Dragon

Šifra, proti ktorej neboli nájdené útoky aj napriek rozsiahlej snahe kryptoanalytikov. Poskytuje pomerne slušnú rýchlosť v porovnaní s AES v counter móde. Na druhú stranu sa radí skôr medzi tie pomalšie z portfólia. Celé šifrovanie a dešifrovanie prebieha v jednom kole, preto šifra neumožnuje jednoduché obmedzenie a tým pádom zníženie bezpečnosti pre účely testovania.

3.7 Edon80

Originálne navrhnutá šifra, ktorá síce nebola úplne prelomená, ale aj napriek tomu niektoré výsledky nasvedčovali tomu, že nie je kompletne bezpečná [15]. Ďalšou jej nevýhodou podľa autorov finálového reportu projektu eSTREAM bola rýchlosť algoritmu. Šifra v každom kole zakóduje 2 bity v každom byte 4 kolá za sebou, čo nám neumožnuje rozumne obmedziť algoritmus tak, aby generoval oslabené prúdy. Jej obmedzenie by znamenalo akurát, že výsledný prúd by mal najvyššie bity nastavené na 0. Navyše šifra pracovala s 80b inicializačnými vektormi a kľúčami, preto sme sa rozhodli ju netestovať. Všetky ostatné šifry dokázali pracovať so 128b IV a kľúčami [1].

1. Niektoré šifry nepoužili celých 128b z IV/kľúču, ale na rozdiel od Edon80 dokázali tento spracovať.

3.8 F-FCSR

Ďaľšia z šifier, proti ktorým neboli nájdené žiadne útoky, ale aj napriek tomu nepostúpila do finále. Priniesla ale zaujímavý návrh založený na špeciálnych automatoch, tzv. Feedback with Carry Shift Registers a podobne, ako Trivium a Grain, bodovala svojou jednoduchosťou. Napriek tomu nepostúpila do finále kvôli priemernému výkonu. Šifru nebolo možné obmedziť bez zásahu do posuvných registrov.

3.9 Fubuki

Šifra, ktorá bola po prvom kole upravená a premenovaná na CryptMT. Jej popis je nižšie. Pôvodnú prelomenú šifru Fubuki sme tiež použili. Obmedzili sme ju na 1-32 kôl vďaka tomu, že generujúca funkcia je v originálnom kóde spúštaná viackrát. Podľa autorov by ale pre zaistenie pseudonáhodnosti výstupu mali stačiť 4 kolá.

3.10 Grain

Extrémne jednoducho implementovaná šifra, ktorej kód má len pár riadkov. Aj kvôli tejto skutočnosti bola obľúbeným terčom kryptoanalytikov. Žiadne slabiny v nej ale neboli nájdené a postúpila do finále. Obmedzením výstupov z posuvných registrov sa nám ju podarilo oslabiť na 1-13 kôl.

3.11 HC

HC-128 a HC-256 sú veľmi rýchle šifry, na ktoré neboli nájdené žiadne útoky aj napriek veľkej snahe. Vďaka tomuto postúpili do finále. Šifry boli dokonca už zakomponované do knižnice CyaSSL používanej u embedded zariadení. Oslabenie šifry sa ukázalo, ako nie úplne jednoduchá úloha a nakoniec sme sa radšej sústredili na šifry, u ktorých bolo toto možné s menšou námahou.

3.12 Hermes

Hermes-8 a Hermes-8F sú dve podobné šifry navrhnuté pre projekt eSTREAM. Rýchlejšia verzia 8F bola úspešne zlomená a aj napriek tomu, že sa útok nedal jednoducho previesť na základnú verziu šifry, autori ukázali chyby v návrhu a je vysoko pravdepodobné, že je napadnuteľná podobným spôsobom a bola preto vyradená zo súťaže v 2. kole [2]. Pre naše testovanie sme použili rýchlejšiu verziu 8F. Šifra už sama o sebe umožňuje škálovanie. Zrýchlená verzia bola na dolnej hranici nastavenia. Počet kôl sa znížil na minimálne 1 z 2 pôvodných.

3.13 LEX

Šifra založená na AES (Rijndael) neponúkala žiadne výrazné zlepšenie oproti AES a preto bola zo súťaže vyradená. Je vysoko pravdepodobné, že bezpečnosť tejto šifry bude na úrovni AES. Možnosti obmedzenia tejto šifry sú celkom rozsiahle, Rijndael sám o sebe pracuje v 10-14 kolách. Podobne u tejto šifry, ktorá z Rijndael vychádza, vieme obmedziť generovanie prúdu na 1 až 13 kôl. Technické problémy tejto šifry pri skompilovaní pod Release profilom v Microsoft Visual Studiu 2010 spôsobili, že sme túto šifru museli spúšťať skompilovanú pod Debug profilom s približne polovičným výkonom, oproti Release profilu.

3.14 MAG

Šifra, ktorá nepostúpila do 2. kola po tom, ako bol nájdený algoritmus pre rozpoznávanie prúdu generovaného šifrou a prúdom náhodných bitov s dĺžkou len 129 bytov [18]! Kód obsahoval definíciu hlavnej funkcie pre prácu s vnútorným stavom šifry na 750 riadkov, čo efektívne znemožňovalo úpravy vedúce k oslabeniu šifry.

3.15 MICKEY

Veľmi dobre navrhnutá šifra, ktorá postúpila do finále. Jediné nájdené kryptoanalytické výsledky sa týkali nesprávnych HW implementácií šifry [13]. Šifra si ukladá stav po bitoch v jednotlivých kolách a preto nebolo možné ju rozumne otestovať v oslabenej verzii.

3.16 Mir-1

Ďalšia šifra, ktorá nepostúpila do 2. kola kvôli nájdenému algoritmu pre rozpoznávanie prúdu generovaného šifrou a prúdom náhodných bitov. Tentokrát je nutných 2^10 slov (1 slovo = 64 bitov) [27]. Nepodarilo sa nám ju jednoducho oslabiť.

3.17 Pomaranch

Podobný výsledok, ako u šifry Edon80 - postup do 3.kola. Neboli nájdené žiadne útoky, ktoré by ju prelomili, napriek tomu nasvedčovali tomu, že sa časom úspešný útok môže podariť [10]. Výsledok - nepostúpenie do finálového výberu má za následok aj slabý výkon šifry v porovnaní s ostatnými. Pre naše testovanie sa nám šifru nepodarilo oslabiť.

3.18 Py

Py predstavuje nový prístup k návrhu šifry. Algoritmus je založený na tzv. posuvných poliach (rolling arrays), ktoré by mali byť rýchle. Algoritmus ale obsahuje chyby a nebol navrhnutý do ďalších fáz projektu eSTREAM [16] [25] [31]. Z pôvodného algoritmu bola navrhnutá nová šifra s názvom PyPy. My sme testovali originálnu šifru Py.

3.19 Rabbit

Ďalší úspešný návrh šifry, ktorý postúpil až do finálového výberu. Šifra bola zakomponovaná do knižnice CyaSSL používanej u embedded zariadení. Algoritmus pre rozpoznávanie prúdu generovaného šifrou Rabbit a náhodným prúdom dát je známy. Jeho zložitosť je však vyššia, ako hádanie kľúča naslepo [20]. Pokus o oslabenie šifry nebol úspešný a preto bola testovaná len jej neoslabená verzia.

3.20 Salsa20

Asi najväčší favorit v celom projekte. Od začiatku poskytovala šifra jednoduchý a škálovatelný dizajn. Nakoniec sa autori rozhodli pre odporučenie algoritmu s 12 kolami a v prípade, že rozhoduje rýchlosť, je možné ho použiť aj s obmedzením na 8 kôl. Kryptoanalýza preukázala najprv náchylnosť na tzv. key-recovery útoky pre šifru obmedzenú na 5 kôl [1]. Postupom času sa dostali autori až k úspešnému útoku pre algoritmus obmedzený na 8 kôl [28]. Šifru bolo kvôli testovaniu možné oslabiť tak, že sa obmedzil počet kôl nutných pre šifrovanie a dešifrovanie (1-20 kôl) prakticky bez zmien v kóde.

3.21 Sfinks

Jednoduchá šifra založená na jednom veľkom LFSR. Tak, ako u veľa ďalších podobných šifier, aj u Sfinks bol nájdený algebraický útok [8]. Obmedzenie šifry nebolo v tomto prípade možné.

3.22 SOSEMANUK

Ďalší z úspešných finalistov projektu. Šifra používa podobný princíp, ako šifra Serpent [5]. Z mnohých útokov na šifru stoja za zmienku tzv. guess-and-determine útoky, pomocou ktorých bolo možné zistiť vnútorný stav šifry hneď po inicializácii. Zložitosť najlepšieho známeho guess-and-determine útoku je 2^176 bitov [11]. Tzv. linear masking method útok umožnil získať vnútorný stav šifry v čase 2^135 [7]. Oslabenie šifry nebolo možné jednoduchým spôsobom.

3.23 Trivium

Jedna s najlepších šifier podľa poroty SASC 2008. Elegantný a čo najviac zjednodušený návrh motivoval mnoho kryptoanalytikov k pokusu o prelomenie tejto šifry. Známa je interpretácia šifry cez sústavu rovníc obsahujúca 954 premenných rozdelených do rovníc po 6 premenných. Aj napriek veľkej snahe nebol nájdený spôsob, akým ju vyriešiť [21] [26] [6]. Šifru je možné obmedziť z 3 na 2 kolá. V takejto konfigurácii boli nájdené útoky priamo na získanie kľúča, alebo jeho častí [12] [9]. Takéto obmedzenie ale bolo technicky náročné a preto sme k jeho realizácii nepristúpili.

3.24 TSC

Sada šifier TSC-1 až TSC-4 bola okrem poslednej verzie prelomená viackrát rôznymi metódami [17] [22]. Predpokladalo sa teda, že aj posledná verzia bude obsahovať podobné chyby. Šifry navyše neposkytovali žiadnu výhodu oproti AES a preto boli vyradené z portfólia eSTREAM v 2. kole. TSC-4 sme dokázali oslabiť na 1-32 kôl. Pre rozumné použitie je nutných minimálne 9 kôl, pretože šifra v prvých ôsmych kolách len napĺňa prázdne štruktúry a jej výstupy v tomto prípade sú nekompletné.

3.25 WG

Šifra s maximálnou dĺžkou vygenerovaného prúdu 2^45 bitov. Navyše jej implementácia zaberá pomerne velké časti pamäti a preto bola vyradená už v 2. kole. Neumožnovala žiadnu relatívne jednoduchú úpravu pre oslabenie šifry.

3.26 Yamb

Šifra, ktorá skončila hneď v prvom kole kvôli nájdenému algoritmu pre rozlíšenie medzi jej výstupom a prúdom náhodných čísiel [19].

3.27 Zk-Crypt

Veľmi zle zdokumentovaná šifra. Tento problém odradil pravdepodobne väčšinu potenciónalnych kryptoanalytikov. Posledná zo šifier, ktorú sa nám nepodarilo vôbec rozbehnúť.

3.28 Zhrnutie

V tabuľkách ([Tabuľka 3.1], [Tabuľka 3.2] a [Tabuľka 3.3]) dávame prehľad nastavenia použitých šifier. Šifry ABC, CryptMT, Edon80 a ZK-Crypt sme sa rozhodli netestovať z dôvodov uvedených u každej šifry v popise (Oddiel 3.1, Oddiel 3.3, Oddiel 3.7 a Oddiel 3.27).

/	Achterbahn	Decim	Dicing	Dragon	FCSR	Grain	FUBUKI
MIN-MAX kôl	N/A	1-8	N/A	N/A	N/A	1-13	1-32
štandardný počet kôl	N/A	8	N/A	N/A	N/A	13	4

Tabuľka 3.1: Prehľad šifier poďľa možností oslabenia. 1/3

/	HC-128	Hermes	LEX	MAG	Mickey	Mir	Pomaranch	Py
MIN-MAX kôl	1-2	1-13	N/A	N/A	N/A	N/A	N/A	N/A
štandard. počet kôl	2	10	N/A	N/A	N/A	N/A	N/A	N/A

Tabuľka 3.2: Prehľad šifier podľa možností oslabenia. 2/3

/	Rabbit	Salsa20	Sfinks	Sosemanuk	Trivium	TSC	Wg	Yamb
MIN-MAX kôl	N/A	1-20	N/A	N/A	N/A	1-32	N/A	N/A
štandard. počet kôl	N/A	20/12	N/A	N/A	N/A	32	N/A	N/A

Tabuľka 3.3: Prehľad šifier podľa možností oslabenia. 3/3

Kapitola 4

Testovacia aplikácia

V tejto kapitole popisujeme samotnú aplikáciu, ktorú sme používali na testovanie. Vzhľadom na to, že naša aplikácia mala vychádzať z už existujúcej aplikácie SensorSim primárne určenej pre simuláciu senzorových sietí, popisujeme v nej funkcionalitu, ktorú sme pridali a zmenili.

4.1 Programové úpravy aplikácie

Existujúca aplikácia SensorSim bola použitá ako základ našej aplikácie. Extrahovali sme z nej funkcionalitu evolučného obvodu a nastavenie evolúcie. Následne sme vytvorili objektový návrh novej aplikácie. Nejde ale o kompletné aplikovanie princípu zapúzdrenia, teda čisto objektový model. Návrh zahŕňa odčlenenie funkcií pre výpočty fitness, mutácie a kríženia do vlastnej triedy. Táto trieda obsahuje ďalej funkcie pre načítavanie, výpis, výpočet a prunovanie obvodov. Od hlavnej aplikácie sa oddelilo generovanie testovacích vektorov (TestVector Generator) a metódy vyhodnocovania úspešnosti obvodov nad testovacou sadou (CircuitEvaluator). Podobným spôsobom bol upravený pseudonáhodný generátor pre podporu viacerých generátorov. Všetky tieto tri časti boli odčlenené do vlastných adresárov a pre každú z nich bola vytvorená virtuálna trieda, ktorú musia všetci potomkovia dediť. Vzhľadom na to, že pri spustení a inicializácii aplikácie nevieme, ktorý konkrétny potomok sa použije, na začiatku vždy inštanciujeme rodiča celej triedy a následne ho pretypujeme na konkrétneho potomka. Tento prístup sme aplikovali na všetky tri odčlenené časti. Zjednodušený class diagram môžeme vidieť na obrázku nižšie (Obrázok 4.1).

4.1.1 CircuitGenome

Trieda reprezentuje jedného jedinca - obvod v populácii. Ako už bolo uvedené, obsahuje všetky funkcie nutné k spúšťaniu a vyhodnocovaniu obvodu.

4.1.2 Evaluator

Trieda zabezpečujúca volanie a spracovanie výsledkov fitness funkcie jedinca pre účely následnej kontroly priebehu evolúcie. Zároveň sa jedná o meno fitness funkcie v triede CircuitGenome.

4.1.3 ITestVectGener

Baseclass ITestVectGener slúži ako predok všetkých generátorov testovacích vektorov. Na základe nastavenia direktívy TEST_VECTOR_GENERATION_METHOD sa rozhodne, ktorý jej potomok bude používaný. V našej aplikácii sme chceli používať len jeden spôsob generovania testovacích vektorov, a to generovanie prúdov z projektu eSTREAM. Preto sme vytvorili jedného potomka tejto triedy s názvom EstreamTestVectGener, ktorý rieši všetky typy generovania testovacích vektorov pre testovanie šifier projektu eSTREAM. V ňom na základe direktívy ESTREAM_GENERATION_METHOD určíme, aké konkrétne typy dát chceme, aby obvod dostával. Nami používaný TESTVECT_ESTREAM_DISTINCT napĺňal testovacie vektory náhodne buď prúdom č.1, alebo prúdom č.2 v závislosti od nastavenia direktív ESTREAM_ALGORITHM, ESTREAM_ALGORITHM2 a ďalších pomocných direktív. Na očakávaný výstup dal vždy číslo prúdu, ktorý použil. Celkovú schému generovania testovacích vektorov a ich následného umiestnenia do obvodu a vyhodnotenia nájdeme na obrázku nižšie (Obrázok 4.2).

4.1.4 Výstupy

Náš program poskytuje viac možností výstupov. Každý z nich má inú funkciu. Niektoré slúžia na odladenie generátora, iné na lepšie pochopenie výsledkov.

4.1.4.1 Najlepšie obvody v priebehu počítania

Počas počitania sa priebežne ukladá fitness najlepšieho jedinca globálne a v prípade, že sa vyskytne lepší jedinec, je zapísaný do nových súborov v adresári, z ktorého sme spustili program. Fitness zapísaného jedinca sa vyskytuje priamo v názvoch súborov. Po ukončení počítania sa výsledný najlepší obvod z populácie taktiež zapíše. Výpisy jedinca sú vždy 4:

- Textový čitatelný formát (.txt) - Obsahuje po riadkoch vypísané skratky funkcií v obvode. Každý riadok reprezentuje jednu vrstvu. Používa sa pre kontrolu obvodu v prípade, že nemáme možnosť zobraziť .dot graf.
- Grafový výstup (.dot) - Formát čitateľný programom Graphviz.
- Priama reprezentácia jedinca (.bin) - Obsahuje reťazec vhodný pre priame načítanie jedinca a pokračovanie v evolúcii. Jeho prítomnosť v spúšťanom adresári automaticky spôsobuje načítanie jedinca pri spustení programu.
- Zdrojový kód obvodu (.c) - skompilovateľný kód obvodu, ktorý je možno použiť pre rýchly test obvodu nezávisle na našej aplikácii. Taktiež je možné jeho prikompilovanie k aplikácii a následné spustenie testu pomocou prepínača „-staticcircuit“. V tomto prípade je nutné mať k dispozícii vstupné súbory s testovacími dátami.

4.1.4.2 Testovacie dáta

Nastavením konfiguračnej direktívy SAVE_TEST_VECTORS na hodnotu 1 signalizujeme programu, aby ukladal pri generovaní testovacie vektory do súboru. Výstupom je viac súborov:

- TestData.txt - Binárny súbor, v ktorom sú testovacie vektory uložené za sebou. Pokiaľ generujeme aj druhý prúd (napr. u distinctoru), uloží sa do súboru TestData2.txt
- TestVectors.txt - Súbor obsahujúci podrobný výpis testovacích vektorov. Obsahuje kľúč a IV pre každý prúd, seed a presný výpis poradia testovacích vektorov spolu s ich plaintextom, ciphertextom a odšifrovaným ciphertextom pre kontrolu.

4.1.4.3 Fitness záznamy

Výstupy fitness funkcie sú nutné k získaniu prehľadu o priebehu a výsledku počítania. Máme 3 direktívy, ktoré ich ovplyvňujú. Sú to TEST_VECTOR_CHANGE_GENERATION, EVALUATE_EVERY_STEP a TVCG_PROGRESSIVE.

- scores.log - výstup knižnice GAlib, ukazuje priemerné, maximálne hodnoty fitness a odchýlky v rámci každej populácie.
- bestfitgraph.txt - Uchováva zoznam fitness najlepšieho jedinca v populácii hneď po zmene testovacích vektorov. Direktíva TEST_VECTOR_CHANGE_GENERATION kontroluje teda, ako často sa bude fitness zapisovať. Pokiaľ chceme ukladať do súboru fitness každú generáciu, použijeme direktívu EVALUATE_EVERY_STEP.
- avgfitgraph.txt - Uchováva priemernú fitness všetkých jedincov v populáciách medzi dvoma zmenami testovacích vektorov. Je teda priamo naviazaný na direktívu TEST_VECTOR_CHANGE_GENERATION.

4.1.4.4 Ostatné výstupy

Kvôli použitiu prostredia BOINC pre spúšťanie testov sme pridali ďalší výstupný súbor, ktorý nesie stav počítania. Súbor fraction_done.txt obsahuje jedno desatinné číslo medzi 0 a 1, ktoré určuje, ako ďaleko je program v počítaní.

Kvôli možnosti zopakovať výpočet sme pridali súbor LastSeed.txt, ktorý ukladá posledné použité seed. V prípade, že nastavíme konfiguračnú direktívu USE_FIXED_SEED na 1, program pri inicializácii bude hľadať tento súbor a použije prvý seed v ňom zapísaný. Pokiaľ ho nenájde, použije seed daný direktívou RANDOM_SEED.

4.1.5 Kontrola výsledkov

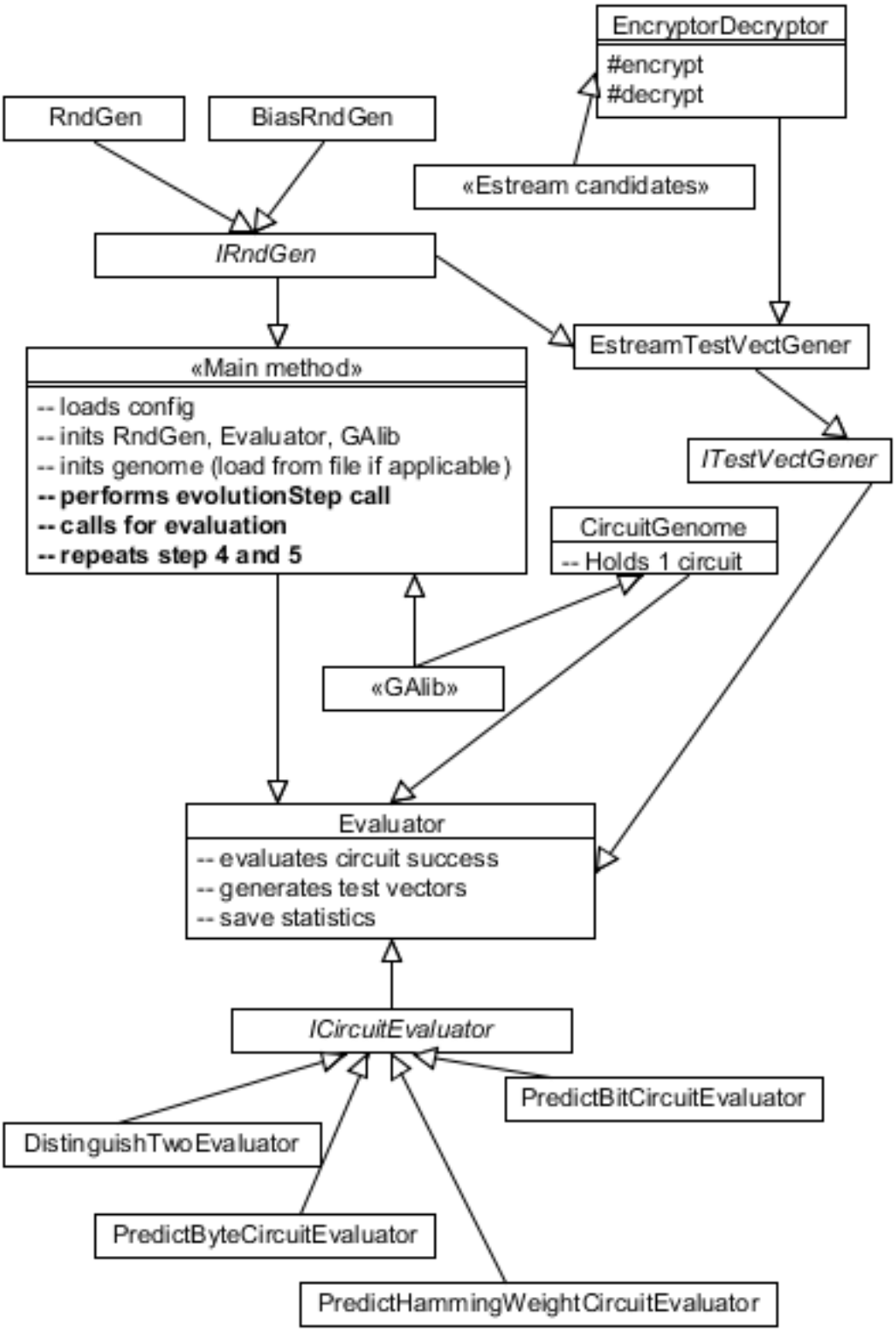
Pre kontrolovanie správnosti výsledkov sme do aplikácie pridali možnosť spúšťania staticky prikompilovaného obvodu a spúšťanie aplikácie s vypnutou evolúciou.

4.1.5.1 Statický obvod

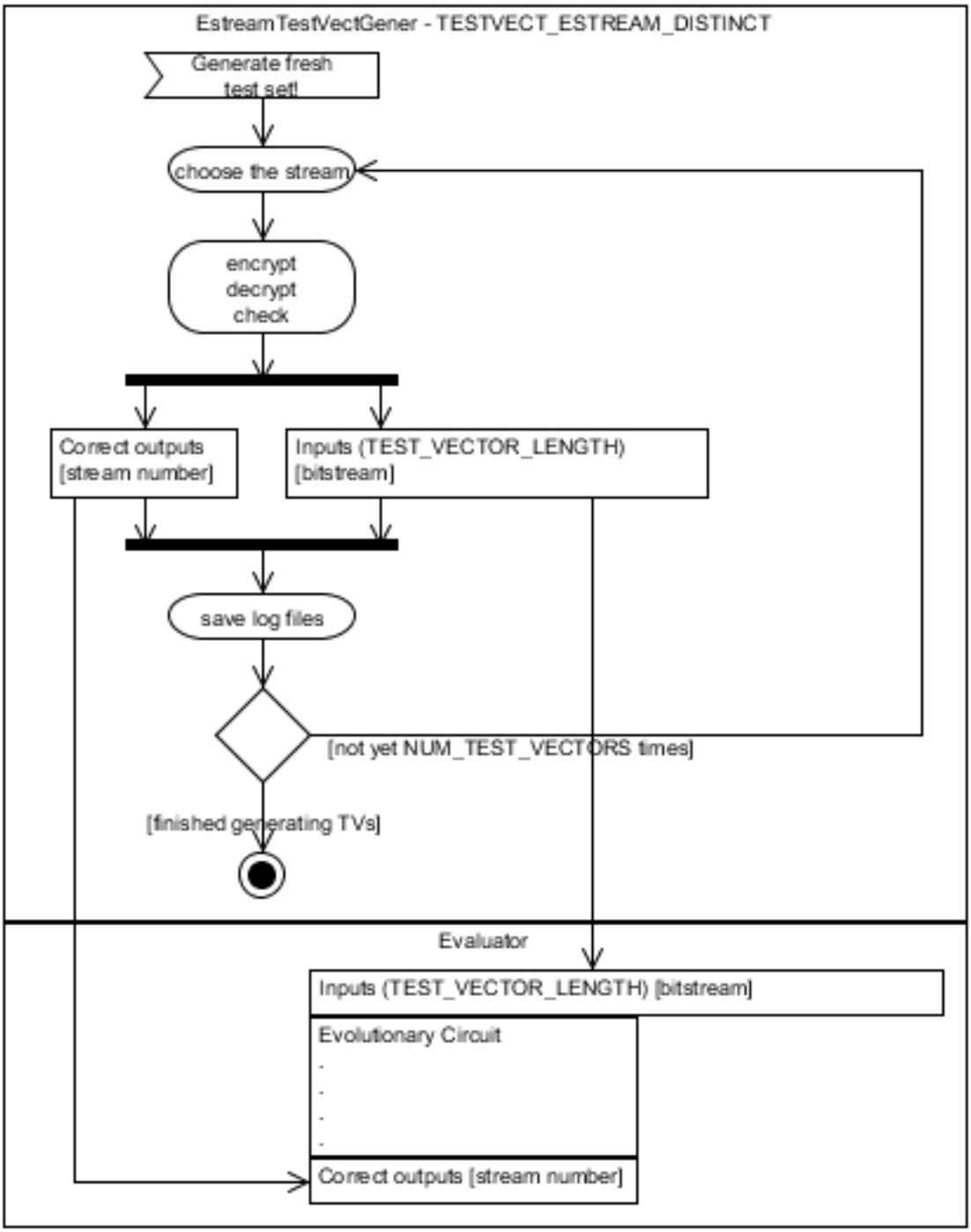
Pred týmto testom je nutné vždy nahradiť súbor EAC_circuit.h v koreňovom zdrojovom adresári požadovaným testovacím obvodom - najčastejšie získaným počas predchádzajúcej evolúcie (výstupný súbor EAC_circuit.c) a prekompilovať celú aplikáciu. Následne je nutné spustiť aplikáciu s prepínačom -staticcircuit. Po spustení sa dozvieme zoznam dostupných testovačov. Implementovali sme len testovač pre rozpoznávač medzi dvoma prúdmi dát. Tento vyžaduje, aby v spúštacom adresári boli prístupné predgenerované dáta pre oba prúdy v súboroch TestData1.txt (pre 1.prúd - zodpovedá poradiu v config.xml) a TestData2.txt (pre 2.prúd - tak isto podľa poradia v config.xml). Pre jeho aktivovanie spustíme aplikáciu s prepínačmi -staticcircuit -distinctor. V prípade, že sú oba testovacie súbory k dispozícii v adresári, začne aplikácia testovať prikompilovaný statický obvod. Výsledky vypisuje čiastkovo podľa nastavenia direktívy NUM_TEST_VECTORS v config.xml. Táto je zdieľaná s nastavením evolučného obvodu. Formát výstupu je jednotný. Každých NUM_TEST_VECTORS spustení obvodu vypíše na výstup podiel úspešných vyhodnotení výstupov obvodu a všetkých vyhodnotení - pomyselnú fitness pri evolúcii.

4.1.5.2 Vypnutá evolúcia

Po spustení aplikácie s prepínačom -evolutionoff sa spustí všetko, ako pri spúštaní bez prepínačov s jedným rozdielom - evolúcia nebude prebiehať. Výsledkom teda bude jedna a tá istá populácia testovaná na vždy nových testovacích vektoroch. V tomto prípade je dobré si populáciu naplniť aspoň jedným jedincom pridaním súboru EAC_circuit.bin do spúštacieho adresára. Všetky výstupy budú rovnaké, ako pri evolúcii. Typické použitie je na otestovanie vyvinutého jedinca na dlhom prúde dát.



Obr. 4.1: Class diagram našej aplikácie



Obr. 4.2: Schéma generovania testovacích vektorov

Kapitola 5

Testy

V tejto kapitole popisujeme jednotlivé prevedené testy a ich výsledky. Najprv sme sa sústredili na štatistické testy DIEHARD a NIST battery tests. Následne uvádzame naše výsledky dosiahnuté genetickým evolučným obvodom.

5.1 Štatistické testovanie

Pre ľahšie začatie testovania a nájdenie aspoň nejakých výsledkov bolo nutné jednotlivé šifry vytriediť a zohľadniť už existujúce výsledky. Vyšli sme z tvrdenia, že výstup dobrej prúdovej šifry musí byť nerozlíšiteľný od prúdu náhodných bitov. Nami testované šifry by mali teda úspešne prejsť štatistickými testami pre generátory pseudonáhodných čisiel. Pre vytriedenie sme použili batériu testov DIEHARD vytvorenú na Florida State University a batériu NIST. Vzorka testovacích dát zahŕňala vždy 32MB súbor s vygenerovaným prúdom z jedného kľúča. Tieto sa vždy použili ako vstup do jednotlivých testov. Každý test dal na výstup okrem presných výsledkov, aj tzv. p-hodnotu v rozmedzí [0,1) - percentil. Tá by mala byť uniformná. Pre nedokonalé pseudo-náhodné dáta by táto mala byť často blízko hodnoty 0, alebo 1. Nasleduje krátky popis jednotlivých testov. [23]

5.1.1 DIEHARD - Birthday spacings

Vyberieme náhodne m dní v roku, ktorý má n dní, uložme si počty dní medzi výbermi do zoznamu. Ďalej spočítajme počet hodnôt, ktoré sa opakujú v zozname viackrát. Pri naozaj náhodnom toku dát je možné presne vypočítať počty jednotlivých súm. Pokiaľ dáta nie sú dostatočne náhodné, rozdiely oproti predpočítaným počtom budú pomerne veľké.

5.1.2 DIEHARD - Overlapping permutations

V tomto teste sa pozeráme na sekvencie 32-bitových integerov dĺžky 5. Každá sekvencia môže byť v 120 stavoch (5!). Algoritmus prechádza sekvencie tak, že najprv načíta zo súboru prvých 5 integerov, následne prvý zahodí a načíta ďalší. Takto pokračuje, kým nenačíta určité množstvo stavov. Všetky stavy sa priebežne spočítavajú a na konci uložia do matice kovariancie veľkosti 120x120.

5.1.3 DIEHARD - Ranks of matrices

Z každého 32-bitového integeru sa najľavejších 31 a 32 bitov použije na vytvorenie matíc veľkosti 31x31 a 32x32. Ich hodnosti by pre naozaj náhodné čísla mala byť skoro vždy nad 28, resp. 29. Spočítame počet jednotlivých hodností a porovnáme s predpokladaným množstvom pre naozaj náhodné čísla.

5.1.4 DIEHARD - Bitstream test

Vstupný súbor sa zoberie, ako prúd bitov. Zoberieme 2^21 slov skladajúcich sa zo za sebou idúcich 20 bitov (tzn. b1 až b20, b2 až b21 atď. z prúdu) a spočítame, koľko rôznych 20-bitových slov nám chýba. Pre náhodné dáta by mal byť počet chýbajúcich slov veľmi blízko konkrétnej hodnote. Pre nedostatočne náhodné dáta bude tento počet výrazne odlišný.

5.1.5 DIEHARD - OPSO, OQSO a DNA

Sada testov Overlapping Pairs Sparse Occupancy, Overlapping Quadruples Sparse Occupancy a ich kombinácia. Počítame vždy slová konkrétnej dĺžky nad 1024-písmennou abecedou. Každé písmeno je kódované 10-timi bitmi. Všetky testy sú založené na počte slov, ktoré sa v určitom bloku (2^21 bytov) nevyskytujú (zo všetkých možných - 1024^2, 1024^4). Predpočítaný počet slov nevyskytujúcich sa v bloku sa porovná s počtom, ktorý vypočítame pre naše dáta. Znova platí, že rozdiel oproti predpočítaným dátam by mal byť čo najmenší.

5.1.6 DIEHARD - Count the 1s

V každom byte spočítame počet jednotiek. Pravdepodobnosť, že dostaneme určitý počet je pevne daná (1,8,28,56,70,56,28,8,1 z 256). Následne zakódujeme reťazec nasledovne. Každé slovo nahradíme písmenom A pokiaľ malo počet výskytov jednotiek 0,1,2, B pre 3 výskyty, C pre 4 výskyty, D pre 5 výskytov, E pre 6,7,8 výskytov. Pravdepobonosť výskytu konkrétneho písmena je teda 37,56,70,56,37 z 256. Slov skladajúcich sa z 5 písmen môže byť 5^5. Zoberieme reťazec obsahujúci 256000 presahujúcich slov dĺžky 5 (tj. dĺžka reťazcu 256004 znakov). Spočítame v ňom počty výskytov jednotlivých slov dĺžky 5. Tieto majú byť čo najbližšie predpočítaným hodnotám pre náhodný generátor.

5.1.7 DIEHARD - Parking lot test

Algoritmus sa snaží zaparkovať na parkovisku veľkosti 100x100 boxov autá (neberie sa do úvahy cesta) pomocou čísiel zo vstupu. Každý pokus sa buď podarí, alebo nie. Neúspešné pokusy o zaparkovanie sa spočítavajú a ich počet na konci sa porovná s predpočítaným počtom pre náhodný generátor. Pre 12000 pokusov by malo dôjsť k priemerne 3523 neúspechom.

5.1.8 DIEHARD - Minimum distance test

Vyberieme náhodne 8000 bodov na štvorci veľkosti 10000x10000 a zmeriame vzdialenosti medzi každými dvoma. Nájdeme minimálnu vzdialenosť d. Pre naozaj náhodné čísla by malo byť d^2 blízke 0.995. Opakujeme 100-krát.

5.1.9 DIEHARD - Random spheres test

Podobne, ako v predchádzajúcom teste vyberieme 4000 bodov na kocke s hranou veľkosti 1000. Spočítame objem gule s priemerom rovným najmenšej vzdialenosti medzi dvoma bodmi. Pri dobrých náhodných dátach by sa mali objemy blížiť číslu 20. Test opakujeme 20-krát.

5.1.10 DIEHARD - The squeeze test

Z každých 32-bitov vytvoríme desatinné číslo medzi [0,1). Na začiatku sa vezme číslo 2^31. Spočítame počet nutných násobení tohoto čísla naším náhodným číslom tak, aby sme dostali číslo 1. Opakujeme test 100000-krát a spočítame počty násobení menej, ako 7 a viac, ako 47. Tieto sa porovnajú s predpočítanými hodnotami.

5.1.11 DIEHARD - Overlapping sums test

Z každých 32-bitov vytvoríme desatinné číslo medzi [0,1). Ku každému číslu (okrem posledných 99) pripočítame 99 hodnôt bezprostredne nasledujúcich po ňom. Takto vytvorené sumy by mali mať normálnu distribúciu.

5.1.12 DIEHARD - Runs test

Algoritmus berie vždy 32-bitov, ako číslo s pohyblivou desatinnou čiarkou (float). Každé číslo porovná s predchádzajúcim a počíta počty čísiel za sebou se nemeniacich postupností (napr. 0.321, 0.333, 0.563, 0.241, 0.871 je postupnosť 3 rastúcich čísiel, následne 1 klesajúceho a 1 stúpajúceho). Následne sa počty použijú pre výpočet matice kovariancie a spočíta hodnosť. Výpočet sa opakuje každých 10000 floatov. Predpočítané hodnosti pre náhodné dáta musia byť čo najbližšie našim výsledkom, pokiaľ máme dobré náhodné dáta.

5.1.13 DIEHARD - The craps test

Algoritmus zahrá 200000 kôl hry Craps a počíta, ako dlho trvalo každé kolo. Následne sa počty dĺžok kôl porovnajú s predpokladanými počtami. Znova, náhodný prúd by mal mať počet čo najbližší k predpokladaným počtom.

5.1.14 DIEHARD - Výsledky - Bias generátor -> key

Každý test dostal na vstup 32MB vstupný súbor s binárne zapísanými prúdmi. V tomto prípade boli vygenerované tak, že každá šifra dostala na vstup nulový inicializačný vektor a náhodný kľúč vygenerovaný bias generátorom s nastavením 95% (šanca na vygenerovanie 1 je 95% a 0 na 5%). Nastavenie parametrov batérie zostalo štandardné. Výsledky interpretujeme tak, že pokiaľ bol percentil konkrétneho testu menší, ako 0.025, alebo väčší, ako 0.975, test neprebehol úspešne. Znamenalo by to, že taký výsledok vygeneruje dobrý PRNG pri jednom z 20 pokusov. Pokiaľ je podobných neúspechov viac, je vysoko pravdepodobné, že prúd je vygenerovaný zlým PRNG.

Výsledky nie sú nijak prekvapivé, predpokladá sa, že autori šifier dostatočne otestovali svoje návrhy dostupnými štatistickými testami. Toto jasne vidno u tabuľky s výsledkami neoslabených šifier ([Tabuľka 5.2] a [Tabuľka 5.3]). V prípade oslabenia šifier sme u niektorých šifier získali zaujímavé výsledky a u väčšiny sa objavila ostrá hrana medzi určitými dvoma počtami kôl, pred ktorou oslabená šifra neprejde cez veľkú väčšinu testov a po nej prejde cez takmer všetky testy. Toto vidno u šifier Decim, Grain, Fubuki, Lex, Salsa20 a TSC. Výsledky sme uviedli prehľadne do tabuľky ([Tabuľka 5.1]).

Každý typ testu dal na výstup celkový percentil za všetky testy jedného typu. Počet neúspešných celých testov uvádzame ako prvé číslo v tabuľke. Maximálny počet je 16. V zátvorke za ním uvádzame v neúspešných celkových testoch počet neúspešných čiastkových testov a maximálny počet čiastkových testov. Toto má význam hlavne v prípade, že prúd neprejde väčšinou testov jedného typu. Vtedy je pomer čiastkových testov vysoký a náhodnosť s vysokou pravdepodobnosťou nedostatočná. V tomto prípade sa maximálny počet mení podľa toho, ktorý typ testov bol neúspešný (niektoré testy sa neopakovali, niektoré opakovali 20-krát atď.).

5.1.15 DIEHARD - Výsledky - QRND generátor -> key

Veľmi podobne, ako u výsledkov s vygenerovaným kľúčom z Bias generátoru, aj v tomto prípade sme mali 32MB súbory s binárnym prúdom s IV a plaintextom nastaveným na 0. Rozdielom bol len kľúč vygenerovaný úplne náhodne náhodne. Očakávané výsledky mali byť o trochu lepšie (náhodné), ako výsledky testov na Bias generátore.

Trochu prekvapivo sa výsledky ([Tabuľka 5.4]) nelíšili od výsledkov získaných z prúdov s biasnutým kľúčom ([Tabuľka 5.1]) s výnimkou šifry Decim.

5.1.16 NIST - Frequency test

Test spočíta počet bitov s hodnotou 1 a 0 v celom prúde, alebo v bloku určitej veľkosti. Zakladá sa na myšlienke, že dobrý pseudo-náhodný generátor by mal mať približne polovicu 1 a polovicu 0.

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-5	1-32	1-2	1-13	1-20	1-32
1 kolo	16	16	0	1(0/9)	16	16	-
2 kolá	14	16	-	-	16	16	-
3 kolá	15	1(0/0)	-	-	16	0	-
4 kolá	10	-	-	-	0	-	-
5 kôl	9	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	-
9 kôl	-	-	-	-	-	-	16
10 kôl	-	-	-	-	-	-	15
11 kôl	-	-	-	-	-	-	7

Tabuľka 5.1: DIEHARD - bias key - šifry obmedzitelné na kolá - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú (hlavne u nižšieho počtu neúspešných testov) počet čiastkových nedostatočných výsledkov (počet/celkovo).

Achterbahn	Dicing	Dragon	FCSR	HC-128	MAG	Mickey	Mir
2(0/10)	0	1(0/25)	0	0	0	0	1(0/0)

Tabuľka 5.2: DIEHARD - bias key - ostatné šifry - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú počet čiastkových nedostatočných výsledkov (počet/celkovo). 1/2

5.1.17 NIST - Runs test

Spočíta počty dĺžok sekvencií 1 a 0. Sekvencia sa počíta vždy po najbližšiu zmenu bitu. 0111110 obsahuje sekvenciu dĺžky 5. Podstata testu je v tom, že dobrý pseudo-náhodný generátor by nemal mať príliš veľa ani príliš málo sekvencií určitej dĺžky.

5.1.18 NIST - Longest Runs of Ones test

Spočíta najdlhšiu sekvenciu jednotiek v bloku dĺžky M. Dĺžky pre dobrý pseudo-náhodný generátor (PRNG) sú predpočítané a nemali by sa príliš líšiť.

Pomaranch	Py	Rabbit	Sfinks	Sosemanuk	Wg	Yamb
1(0/0)	0	1(0/0)	1(1/25)	0	0	0

Tabuľka 5.3: DIEHARD - bias key - ostatné šifry - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú počet čiastkových nedostatočných výsledkov (počet/celkovo). 2/2

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-5	1-32	1-2	1-13	1-20	1-32
1 kolo	16	16	1(0/0)	0	16	16	-
2 kolá	13	16	-	-	16	16	-
3 kolá	13	0	-	-	16	0	-
4 kolá	10	-	-	-	0	-	-
5 kôl	10	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	-
9 kôl	-	-	-	-	-	-	16
10 kôl	-	-	-	-	-	-	15
11 kôl	-	-	-	-	-	-	7

Tabuľka 5.4: DIEHARD - qrnd key - šifry obmedzitelné na kolá - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú (hlavne u nižšieho počtu neúspešných testov) počet čiastkových nedostatočných výsledkov (počet/celkovo).

Achterbahn	Dicing	Dragon	FCSR	HC-128	MAG	Mickey	Mir
2(0/9)	1	0	2	0	0	0	0

Tabuľka 5.5: DIEHARD - qrnd key - ostatné šifry - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú počet čiastkových nedostatočných výsledkov (počet/celkovo). 1/2

5.1.19 NIST - Binary Matrix Rank test

Rovnaký test, ako DIEHARD - Ranks of matrices.

5.1.20 NIST - Discrete Fourier Transform test

Vstupný prúd 0 a 1 transformujeme na čísla -1 a +1. Na ne následne aplikujeme diskrétnu Fourierovu transformáciu. Následne skúmame jej vlastnosti.

Pomaranch	Py	Rabbit	Sfinks	Sosemanuk	Wg	Yamb
0	1(1/9)	0	0	0	0	2

Tabuľka 5.6: DIEHARD - qrnd key - ostatné šifry - počet neúspešných testov zo 16 dokopy. Hodnoty v zátvorke upresňujú počet čiastkových nedostatočných výsledkov (počet/celkovo). 2/2

5.1.21 NIST - (Non-)overlapping Template Matching test

Test spočíta počty rôznych vzorov v prúde. Počty jednotlivých vzorov by nemali byť príliš odlišné pre dobrý PRNG.

5.1.22 NIST - Maurer's „Universal Statistical" test

Test počíta počty bitov medzi rovnakými vzormi. Ľahko komprimovateľné dáta sa považujú za menej náhodné. Komprimovateľnosť je priamo naviazaná na opakujúce sa vzory.

5.1.23 NIST - Linear Complexity test

Test kontroluje, po ako dlhom prúde sa začnú dáta opakovať. Toto je priamo naviazané na LFSR používané v prúdových šifrách. Príliš časté opakovanie je považované za nevhodné pre náhodné dáta.

5.1.24 NIST - Serial test

Účelom testu je spočítať počet rôznych n-bitových sekvencií v prúde. Pokiaľ sa nejaká sekvencia vyskytuje ovela častejšie, ako ostatné, alebo príliš málo, prúd bol vygenerovaný zlým PRNG.

5.1.25 NIST - Approximate Entropy test

Podobne, ako v predchádzajúcom teste, aj v tomto prípade počítame počty rôznych n-bitových sekvencií v prúde. Následne ale spočítame počty rôznych (n+1)-bitových sekvencií a porovnáme, nakoľko sa výsledky líšia od predpokladaných výsledkov pre dobrý PRNG.

5.1.26 NIST - Cumulative Sums (Cusum) test

Vstupný prúd 0 a 1 transformujeme na čísla -1 a +1. Následne počítame sumy určitých dĺžok. Výsledné sumy by nemali byť ani príliš malé, ani príliš veľké pre dobrý PRNG.

5.1.27 NIST - Random Excursions test

Vstupný prúd 0 a 1 transformujeme na čísla -1 a +1. Následne spočítame pre každé číslo S_i čiastočné sumy $S_{\_i}$ + ... $S_{\_i+n}$. Počty sa následne porovnajú s predpočítanými dátami pre naozaj náhodný prúd.

5.1.28 NIST - Výsledky - Bias generátor

Každý test dostal na vstup 32MB vstupný súbor s binárne zapísanými prúdmi. V tomto prípade boli vygenerované tak, že každá šifra dostala na vstup nulový inicializačný vektor a náhodný kľúč vygenerovaný bias generátorom s nastavením 95% (šanca na vygenerovanie

1 je 95% a 0 na 5%). Nastavenie parametrov batérie zostalo štandardné ([Tabuľka 5.7]). Počet testov dokopy bol viac, ako 17000.

Frequency Test - dĺžka bloku	128
NonOverlapping Template Test - dĺžka bloku	9
Overlapping Template Test - dĺžka bloku	9
Approximate Entropy Test - dĺžka bloku	10
Serial Test - dĺžka bloku	16
Linear Complexity Test - dĺžka bloku	500
Počet prúdov pre testovanie	100

Tabuľka 5.7: Nastavenie parametrov batérie NIST

Výstupy sú v tomto prípade oveľa ľahšie čitateľné, ako u batérie DIEHARD. Ku každému testu máme k dispozícii počet úspešných testov a celkový počet opakovaní. Spolu sme počítali takmer 18000 testov. Výsledky testov môžeme interpretovať tak, že spočítame približne počet úspešných testov vo výsledkovom súbore. Čím vyššie číslo dostaneme, tým bol test úspešnejší. Nie je nezvyklé dostať aj u dobrých PRNG výsledok o 10% horší, ako je získateľné maximum. Predpokladali sme, že výsledky by mali približne kopírovať výsledky batérie DIEHARD.

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-5	1-32	1-2	1-13	1-20	1-32
1 kolo	0	0	17000+	17000+	100	300	0
2 kolá	2800	100	-	-	500	400	0
3 kolá	7500	17000+	-	-	950	17000+	0
4 kolá	10000	-	-	-	17000+	-	0
5 kôl	11000	-	-	-	-	-	0
6 kôl	17000+	-	-	-	-	-	0
7 kôl	-	-	-	-	-	-	0
8 kôl	-	-	-	-	-	-	0
9 kôl	-	-	-	-	-	-	0
10 kôl	-	-	-	-	-	-	1000
11 kôl	-	-	-	-	-	-	17000!

Tabuľka 5.8: NIST - Bias, šifry obmedzitelné na kolá - počet úspešných testov spolu z 17000+

Achterbahn	Dicing	Dragon	HC-128	MAG	Mickey	Mir	Pomaranch
17000+	17000+	17000+	17000+	17000+	17000+	17000+	17000+

Tabuľka 5.9: NIST - ostatné šifry - počet úspešných testov spolu z takmer 18000 1/2

Py	Rabbit	Sfinks	Sosemanuk	Wg	Yamb
17000+	17000+	17000+	17000+	17000+	17000+

Tabuľka 5.10: NIST - ostatné šifry - počet úspešných testov spolu z takmer 18000 2/2

5.1.29 NIST - Výsledky - QRND generátor

Veľmi podobne, ako u rozdielov medzi výsledkami DIEHARD s bias generátorom a kvantovým generátorom sa aj v tomto prípade dali očakávať výsledky podobné NIST s bias generátorom (Oddiel 5.1.28). Nastavenie batérie NIST zostalo rovnaké, jediným rozdielom boli vstupné dáta, ktoré boli generované z úplne náhodného kľúča. Výsledky interpretujeme rovnakým spôsobom, ako u testov s bias generátorom.

Rozdiely sú podľa predpokladov naozaj minimálne ([Tabuľka 5.11]).

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-5	1-32	1-2	1-13	1-20	1-32
1 kolo	0	0	17000+	17000+	100	300	0
2 kolá	2900	100	-	-	600	400	0
3 kolá	7500	17000+	-	-	1000	17000+	0
4 kolá	11000	-	-	-	17000+	-	0
5 kôl	14000	-	-	-	-	-	0
6 kôl	17000+	-	-	-	-	-	0
7 kôl	-	-	-	-	-	-	0
8 kôl	-	-	-	-	-	-	0
9 kôl	-	-	-	-	-	-	100
10 kôl	-	-	-	-	-	-	2000
11 kôl	-	-	-	-	-	-	17000!

Tabuľka 5.11: NIST - QRND, šifry obmedzitelné na kolá - počet úspešných testov spolu z 17000+

5.2 Genetický evolučný obvod - výsledky

Tak isto, ako u štatistických testov, aj spúšťanie nášho programu sme rozdelili na časť s použitím Bias generátora a časť, kde sa používali úplne náhodné dáta. Ďalej sme výsledky rozdelili na výsledky získané pri generovaní prúdov veľkosti 16B a prúdov veľkosti 256B. Výslednú fitness sme (pokiaľ nie je uvedené inak) počítali vždy zo súboru bestfitgraph.txt (Oddiel 4.1.4.3). Všetky testy prebehli 26 až 30-krát[1] za pomoci nástroja BOINC pre paralelné počítanie na viacerých procesoroch podľa dostupnosti jednotlivých jadier pre výpočty. Výsledky sú počítané, ako priemery zo všetkých prebehnutých testov. K dispozícii sme mali

1. Nie vždy boli dostupné všetky procesory pre počítanie.

maximálne 30 jadier na 15 procesoroch pre naše výpočty. Všetky použité výsledky sa nachádzajú v elektronickej prílohe zotriedené podľa šifry.

5.2.1 16B

Pre spúšťanie testov s generovaním prúdu veľkosti 16B bol rozumný dôvod. Na vstup nášho obvodu je možné dať maximálne 32 UCHAR-ov. Pre zvýšenie rýchlosti počítania sme sa rozhodli použiť veľkosť vstupných dát 16B. Ďalej väčšina šifier aktualizovala svoj vnútorný stav po každých 16B vygenerovaného prúdu, takže rozdelenie na 16B dlhé bloky bolo prirodzené.

5.2.1.1 Bias generátor

Vzhľadom na to, že výstupný prúd všetkých šifier je priamo závislý na kľúči, rozhodli sme sa najprv pomôcť nášmu rozpoznávaču generovaním nie úplne náhodných kľúčov. Generoval ich bias generátor a boli použité pri inicializácii šifry. Následne sa šifra už počas celého behu programu nepreinicializovávala. Ostatné volania PRNG už vybavoval kvantový generátor. Nastavenie evolučného obvodu nájdeme v tabuľke nižšie ([Tabuľka 5.12]).

Typ PRNG pre použitie mimo generovania testovacích vektorov	kvantový zo súboru
Veľkosť populácie - počet jedincov (obvodov)	20
Pravdepodobnosť mutácie	0.05
Pravdepodobnosť kríženia	0.5
Maximálny počet krokov	100000
Evolučný obvod - počet vrstiev	5
Evolučný obvod - počet vstupov	16
Evolučný obvod - počet výstupov	2
Evolučný obvod - veľkosť vnútorných vrstiev	8
Počet testovacích vektorov	1000
Dĺžka testovacích vektorov	16B
Generátor testovacích vektorov - IV	Samé 0
Generátor testovacích vektorov - vstupný plaintext	Samé 0
Generátor testovacích vektorov - kľúč	Bias generátor 95%
Zmena testovacích vektorov	každých 100 generácií

Tabuľka 5.12: Nastavenie evolučného algoritmu - Bias generátor

Na základe výsledkov štatistických testov sme očakávali, že náš genetický algoritmus pravdepodobne nenájde nič navyše. U výsledkov neobmedzených algoritmov sme dostali očakávané výsledky - algoritmus sa nedokázal naučiť rozpoznávať náhodný prúd dát od prúdu vygenerovaného konkrétnou šifrou na 16 bytoch u žiadnej šifry ([Tabuľka 5.13])). V tabuľkách ďalej sú uvedené pomlčkou výpočty, ktoré neboli realizované a ako N/A výpočty, ktoré neviedli k žiadnemu rozumnému výsledku. V tomto prípade sme za rozumný

výsledok považovali jedinca, ktorý by na dlhom prúde (aspoň niekoľko MB) dokázal rozlišovať medzi prúdom náhodným a vygenerovaným konkrétnou šifrou s fitness (pravdepodobnosťou) výrazne vyššou, ako 0,5.

Achterbahn až Yamb
N/A

Tabuľka 5.13: 16B, Bias, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - neobmedzené algoritmy

Následne sme sa pokúsili u šifier, u ktorých to bolo možné, spustiť evolúciu s rovnakým nastavením a zníženým počtom kôl (obmedzené šifry). Počet opakovaní sa znižoval podľa toho, aké výsledky sa očakávali z predchádzajúcich testov a výsledky v tabuľke sú znova priemerované ([Tabuľka 5.14]). Nemalo by význam spúšťať evolúciu na 100000 opakovaní pre šifru, o ktorej vieme, že ju určite každý z 30 paralelných behov dokáže rozpoznať na maximálne 1000 krokov. Znižovali sme preto maximálny počet krokov na 30000. Počítali sme s tým, že úspešný výsledok je taký, ktorý vie rozlišovať medzi prúdom vygenerovaným šifrou a náhodným s pravdepodobnosťou aspoň 99%. V tomto prípade sme sa už dostali k zaujímavejším výsledkom.

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-13	1-32	1-2	1-13	1-20	1-32
1 kolo	475	158	N/A	N/A	334	2754*	-
2 kolá	*	187	-	-	3139	3512*	-
3 kolá	*	N/A	-	-	2579	N/A	-
4 kolá	N/A*	-	-	-	N/A	-	-
5 kôl	-	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	166
9 kôl	-	-	-	-	-	-	300
10 kôl	-	-	-	-	-	-	220
11 kôl	-	-	-	-	-	-	N/A

Tabuľka 5.14: 16B, Bias, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - obmedziteľné algoritmy

K vyššie uvedenej tabuľke ([Tabuľka 5.14]) je nutné dodať, že všetky výsledky, okrem výsledkov šifry Salsa20 dali na výstup obvod, ktorý dokázal rozlíšiť medzi dvoma prúdmi ľubovolnej dĺžky s pravdepodobnosťou blížiacou sa 1. U šifry Salsa20 sa obvod celý čas prispôsoboval generovanému prúdu a vedel po čase rozlišovať náhodný a vygenerovaný prúd v jednom momente. Toto ale neplatilo obecne, pokiaľ dostal obvod na vstup oveľa dlhší prúd, na akom sa učil, bol schopný rozpoznávať len dáta v určitom momente. Za

úspešný sme u Salsy označili taký obvod, ktorý bol schopný rozpoznávať aspoň prúd veľkosti 100kB v jednom momente s pravdepodobnosťou aspoň 0,9. V tabuľke uvádzame preto výsledky Salsy s „*“. Ostatné dáta v prúde boli pre neho úplne náhodné.

V prípade šifry Decim na 2 a 3 kolách sa algoritmu nepodarilo nájsť univerzálny rozpoznávač. Namiesto toho ale obvod hľadal iný typ chyby. Šifra totiž raz za čas (približne každých 100 bytov) vygenerovala niekoľko rovnakých bytov. Navyše bol algoritmus schopný sa naučiť až príliš ľahko rozlišovať medzi prúdmi na rovnakých testovacích vektoroch. Keďže sa generovanie testovacích vektorov a následné učenie na nich dialo vždy po 100 krokoch genetického algoritmu, mal algoritmus k dispozícii vždy tie isté dáta nejaký čas. Dĺžka týchto dát nebola príliš veľká (1000 x 16B) a preto bolo nutné meniť testovacie vektory každých 100 krokov. Obvody na šifrách s dobrým pseudo-náhodným výstupom sa za 100 krokov dokázali naučiť len minimum a priemerne končili po 100 krokoch na výsledku fitness vždy 0,535 (odchýlka menej, ako 0,001). Následne po zmene testovacích vektorov spadla fitness na priemerne 0.5. Toto ale neplatilo u šifry Decim na 2 kolách. V jej prípade sa dokázal obvod väčšinou naučiť na rovnakých dátach po 100 krokoch rozlišovať prúdy s priemerne 55,6% pravdepodobnosťou (fitness 0,556). Tento výsledok môže taktiež znamenať slabosť testovanej obmedzenej šifry a preto je v tabuľke ([Tabuľka 5.14]) označený znakom „*“ namiesto priemerného počtu krokov nutných na získanie úspešného obvodu. Pre Decim na 3 kolách sa situácia opakovala s pravdepodobnosťou 54,7% (fitness 0,547). Tieto údaje boli získané z výstupných súborov avgfitgraph.txt, ako priemery za všetky vyhodnocovania (Oddiel 4.1.4.3). Pre Decim na 4 kolách sme dosiahli fitness 0,537, čo by sa vzhľadom na bežnú odchýlku už dalo považovať za problém.

5.2.1.2 Kvantový generátor

Po testoch s bias generátorom sme rovnaké testy chceli spustiť aj s kvantovým PRNG a následne porovnať, či sa líšia v rozsahu, alebo rýchlosti hľadania. Ako sme už uviedli, výsledný prúd je priamo závislý na použitom kľúči. Preto nás zaujímalo, či sa objaví nejaký rozdiel medzi prúdmi generovanými z úplne náhodného kľúču a prúdmi generovanými z biasnutého kľúču v zmysle zrýchlenia procesu nájdenia rozpoznávacieho obvodu, alebo zlepšenia/zhoršenia schopnosti hľadania rozponávacích obvodov. Nastavenie evolučného obvodu zostalo rovnaké ([Tabuľka 5.12]) s výnimkou nastavenia generátora kľúčov ([Tabuľka 5.15]).

Generátor testovacích vektorov - kľúč	kvantový zo súboru

Tabuľka 5.15: Nastavenie evolučného algoritmu - QRND generátor - rozdiely oproti 16B-Bias

Predpokladané výsledky boli horšie, ako s bias generátorom. To znamená, že sa náš genetický obvod mal naučiť rozpoznávať prúdy pomalšie (na vyšší počet iterácií) oproti prúdom generovaným z kľúča bias s generátorom Oddiel 5.2.1.1. Tento predpoklad sa potvrdil takmer bez výnimky. Výsledné obvody mali priemerný čas učenia výrazne vyšší (niektoré

v priemere aj viac, ako o 100%) ([Tabuľka 5.14]).

Ďalším predpokladom bolo, že by sa výsledky niektorých ťažšie nájditeľných obvodov z testov s bias generátorom vôbec nemuseli opakovať a prúd by nebol rozpoznávateľný v tomto prípade. Tento predpoklad sa ukázal, ako nepravdivý. Všetky výsledky z testov s bias generátorom sa zopakovali aj v tomto prípade (akurát pomalšie).

U šifier Decim na 2 a 3 kolách sa situácia opakovala a aj napriek tomu, že sa nenašiel univerzálny rozpoznávač prúdov, priemerná fitness sa pohybovala na rovnakých číslach, ako u Bias generátora, a to na 55,6% a 54,7% (0,556 a 0,547).

U výsledkov šifier neobmedzených na kolá sa žiadne zmeny nepredpokladali a ani vo výsledku neudiali ([Tabuľka 5.16]).

Achterbahn až Yamb
N/A

Tabuľka 5.16: 16B, QRND, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - neobmedzené algoritmy

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-13	1-32	1-2	1-13	1-20	1-32
1 kolo	479	295	N/A	N/A	200	5458*	-
2 kolá	*	433	-	-	3250	5591*	-
3 kolá	*	N/A	-	-	5591	N/A	-
4 kolá	N/A*	-	-	-	N/A	-	-
5 kôl	-	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	133
9 kôl	-	-	-	-	-	-	529
10 kôl	-	-	-	-	-	-	333
11 kôl	-	-	-	-	-	-	N/A

Tabuľka 5.17: 16B, QRND, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - obmedziteľné algoritmy

5.2.2 256B

Predpokladali sme, že dĺžka prúdu 16B nie je dostačujúca pre obvod na to, aby rozhodol o tom, kam daný prúd patrí, preto sme pomocou READ inštrukcie (Oddiel 2.1.4) rozšírili obvodu možnosť vidieť určitú časť dát. V tomto prípade bolo pri každej zmene vygenerovaných 1000 testovacích vektorov po 256B. Pri spúštaní testov na obvode mal tento vždy na vstupe prvých 16B a celý 256B blok pomocou READ funkcie k dispozícii. Výsledky by

všeobecne podľa našich predpokladov mali byť viditeľné skôr, ako v predchádzajúcich prípadoch. Ďalšia predpokladaná zmena oproti predchádzajúcim výsledkom by malo byť rozšírenie množiny rozpoznávaných oslabených šifier.

5.2.2.1 Bias generátor

Dĺžka testovacích vektorov	256B

Tabuľka 5.18: Nastavenie evolučného algoritmu - Bias generátor - rozdiely oproti 16B-Bias

Achterbahn až Yamb
N/A

Tabuľka 5.19: 256B, Bias, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - neobmedzené algoritmy

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-13	1-32	1-2	1-13	1-20	1-32
1 kolo	537	179	N/A	N/A	312	6844*	-
2 kolá	*	250	-	-	1550	5058*	-
3 kolá	*	N/A	-	-	2968	N/A	-
4 kolá	N/A*	-	-	-	N/A	-	-
5 kôl	-	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	133
9 kôl	-	-	-	-	-	-	372
10 kôl	-	-	-	-	-	-	296
11 kôl	-	-	-	-	-	-	N/A

Tabuľka 5.20: 256B, Bias, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - obmedziteľné algoritmy

Pre väčšinu šifier znamenala možnosť pozrieť sa dopredu v prúde predĺženie času nutného pre nájdenie rozpoznávacieho obvodu ([Tabuľka 5.20]). V tomto prípade sa predpoklad nenaplnil. Dôvodom bol pravdepodobne už tak krátky čas, za ktorý algoritmus bol schopný nájsť obvody. S možnosťou pozrieť sa dopredu musí algoritmus chvíľu pracovať, kým príde na to, ktorý konkrétny vstup je pre neho výhodné použiť. Výsledné obvody vôbec nevyužívali funkciu READX.

Taktiež sme nenašli prípad, kde by na rozdiel od predchádzajúcich testov bol algoritmus schopný nájsť obvod pre nastavenie, pre ktoré nedokázal nájsť riešenie u 16B blokov.

U šifry Decim sa situácia opakovala, tentokrát s fitness zníženou na 54,6% a 54,1%. Aj v tomto prípade vidno, že možnosť pozrieť sa dopredu vyrábala obvodu skôr problémy.

5.2.2.2 Kvantový generátor

Dĺžka testovacích vektorov	256B
Generátor testovacích vektorov - kľúč	kvantový zo súboru

Tabuľka 5.21: Nastavenie evolučného algoritmu - QRND generátor - rozdiely oproti 16B-Bias

Achterbahn až Yamb
N/A

Tabuľka 5.22: 256B, QRND, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - neobmedzené algoritmy

Kolá-Výsledky	Decim	Grain	FUBUKI	Hermes	LEX	Salsa20	TSC
MIN-MAX kôl	1-8	1-13	1-32	1-2	1-13	1-20	1-32
1 kolo	604	245	N/A	N/A	260	8256*	-
2 kolá	*	235	-	-	1404	9019*	-
3 kolá	*	N/A	-	-	3275	N/A	-
4 kolá	N/A*	-	-	-	N/A	-	-
5 kôl	-	-	-	-	-	-	-
6 kôl	-	-	-	-	-	-	-
7 kôl	-	-	-	-	-	-	-
8 kôl	-	-	-	-	-	-	169
9 kôl	-	-	-	-	-	-	244
10 kôl	-	-	-	-	-	-	312
11 kôl	-	-	-	-	-	-	N/A

Tabuľka 5.23: 256B, QRND, koľko kôl trvalo algoritmu, kým sa naučil rozpoznávať daný prúd od náhodného - obmedziteľné algoritmy

Veľmi podobne, ako u výsledkov 16B prúdov, sú rozdiely medzi šiframi, ktoré dostali náhodný kľúč a biasnutý kľúč, hlavne v tom, ako dlho trvalo algoritmu nájsť zodpovedajúci rozlišovací obvod. U šifry Decim sa situácia znova opakovala, tentokrát s fitness 54,6% a 54,1% (takmer rovnako, ako u 256B, Bias).

5.3 Nájdené obvody

V tejto podkapitole popisujeme niektoré výsledné obvody, ktoré sme dostali v predchádzajúcich testoch. Ku každému obvodu uvádzame aj konkrétnu slabinu v šifre, ktorú obvod využíva pre rozlíšenie prúdov. Všetky sa nachádzajú v elektronickej prílohe v adresári circuits.

5.3.1 Lex - 1r - qrnd - 256B

Obvod vyvinutý s nastavením šifry Lex na 1 kolo, ako prvý prúd. Druhým prúdom boli náhodné dáta. Dĺžka každého testovaného prúdu bola v tomto prípade 256B. Nájdený rozpoznávač pre tento obvod využíva skutočnosti, že pre malý počet kôl šifra nevyplní dostatočne celú dĺžku prúdu. Z každých 16B je až 12B prednastavených na štandardnú hodnotu. Príklad prúdu:

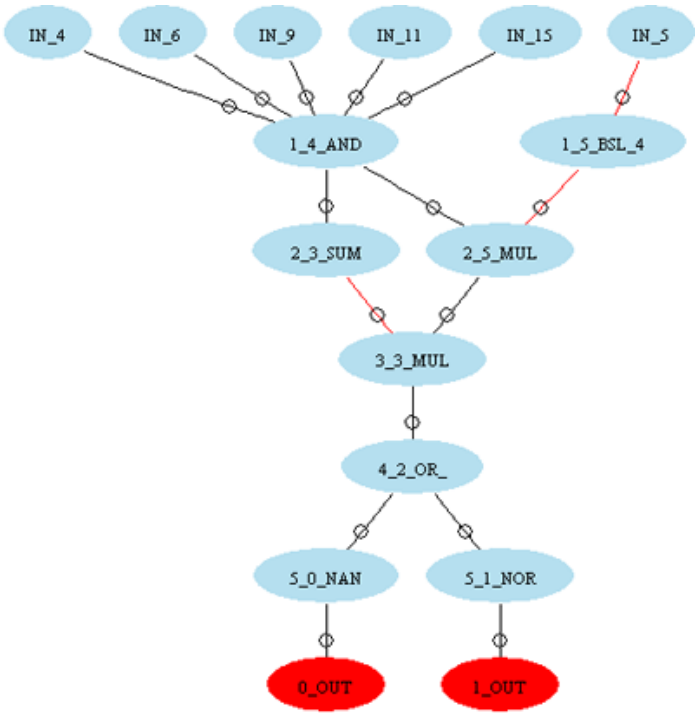
```
03 F2 77 CA CD CD CD CD CD CD CD CD CD CD CD CD
66 FD FA 69 CD CD CD CD CD CD CD CD CD CD CD CD
C0 A4 53 A8 CD CD CD CD CD CD CD CD CD CD CD CD
71 40 D9 16 CD CD CD CD CD CD CD CD CD CD CD CD
A6 43 A5 07 CD CD CD CD CD CD CD CD CD CD CD CD
EF 98 CB 22 CD CD CD CD CD CD CD CD CD CD CD CD
94 B2 00 9E CD CD CD CD CD CD CD CD CD CD CD CD
B2 61 CC 34 CD CD CD CD CD CD CD CD CD CD CD CD
A8 FB 0A D3 CD CD CD CD CD CD CD CD CD CD CD CD
4A 8F B1 F6 CD CD CD CD CD CD CD CD CD CD CD CD
42 0F 7D 0E CD CD CD CD CD CD CD CD CD CD CD CD
7B 73 21 45 CD CD CD CD CD CD CD CD CD CD CD CD
96 B6 63 D4 CD CD CD CD CD CD CD CD CD CD CD CD
18 B6 13 1B CD CD CD CD CD CD CD CD CD CD CD CD
31 55 7C 97 CD CD CD CD CD CD CD CD CD CD CD CD
B5 CE 92 98 CD CD CD CD CD CD CD CD CD CD CD CD
```

Technicky spraví najprv bitový AND vstupov číslo 5, 7, 10, 12 a 16. Pre náhodné dáta by mal mať výsledok tejto operácie všetky bity nastavené na 0 (Priemerne má na začiatku nastavené 4 z 8 bitov na 1. Po prvom AND priemerne zostanú 2 z 8, po druhom AND 1 z 8 atď.). Pre prúd vygenerovaný šifrou LEX na 1 kole by mala operácia dávať výsledok CD. Zvyšné operácie už sa na samotnom rozlišovaní nepodieľajú, upravujú len formát výstupu.

Obvody sme vždy testovali, ako statické obvody (Oddiel 4.1.5.1) a s vypnutou evolúciou (Oddiel 4.1.5.2). V obidvoch prípadoch prešiel obvod s výsledkami fitness blízko 1 - rozpoznal takmer všetky vstupné prúdy. Veľmi podobné obvody boli výsledkom evolúcie pre šifru Lex na 2 a 3 kolách (8B, resp. 4B nastavených na štandardnú hodnotu).

5.3.2 Salsa - 1r - bias - 16B

Získaný vyvinutý obvod pre rozlišovanie šifry Salsa na 1 kole proti náhodnému prúdu dát má určité špecifiká. Nenaučil sa rozpoznávať paušálne všetky prúdy, ktoré sme vygenero-



Obr. 5.1: Rozpoznávač obvodu Lex - 1r - qrnd - 256B

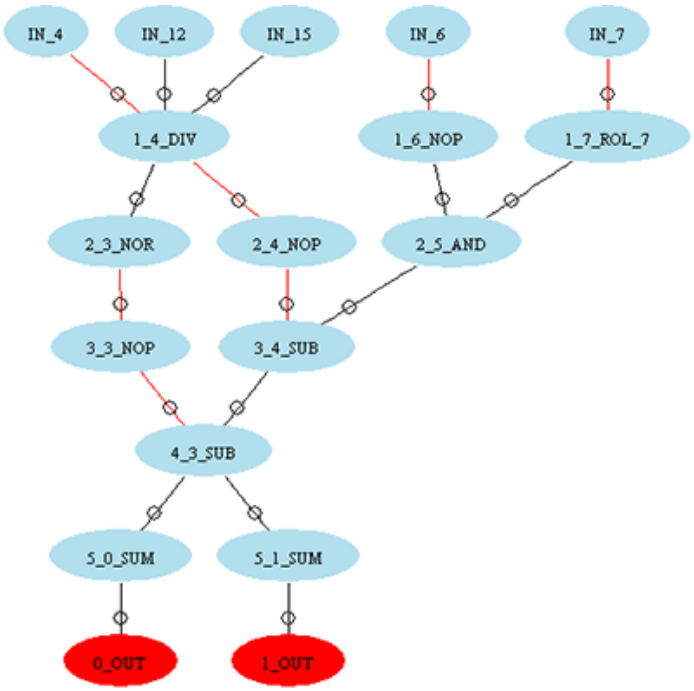
vali. Dokázal ale vždy dospieť do stavu, kedy vedel s približne 75% pravdepodobnosťou uhádnuť u náhodných dát, že nejde o Salsu (a teda ide o náhodné dáta). Pokiaľ dostal ale na vstup Salsu, triafal buď všetko, alebo nič. Tieto časti sa striedali v rôznych dĺžkach (~10kB až ~1MB prúd s podobnými výsledkami, následne zmena na opačný pól) počas celého testovania. Testovanie prebiehalo tak pomocou statického obvodu s predgenerovanými dátami, ako aj s vypnutou evolúciou. Obidve testy potvrdili horeuvedené tvrdenie.

Technicky obvod našiel problém na najvyššom bite v tomto prípade vstupu č.7 a 8 (IN_6 a IN_7, číslované od 0). Vzhľadom na výsledky ďalších obvodov (Obrázok 5.3 a Obrázok 5.4) rozpoznávajúce iné časti prúdov môžeme povedať, že problém šifry Salsa na 1 kole je práve 7. byte (z každých 16B). Každý obvod totiž po vyvinutí vychádzal zo vstupu 7.byte [2].

5.4 Zhrnutie výsledkov

V tejto kapitole sme ukázali, že štatistické testy dokážu rozpoznať určitú množinu chýb v šifrách projektu eSTREAM. Išlo hlavne o problémy v oslabených šifrách. V prípade, že sme použili neoslabené verzie šifier, nenašiel sa žiaden problém použitím štatistických testov DIEHARD a NIST. Pokiaľ šifru inicializujeme biasnutým kľúčom namiesto úplne ná-

2. Všetky obvody sú v elektronickej prílohe v adresári circuits/salsa1rbias16 v .dot formáte.

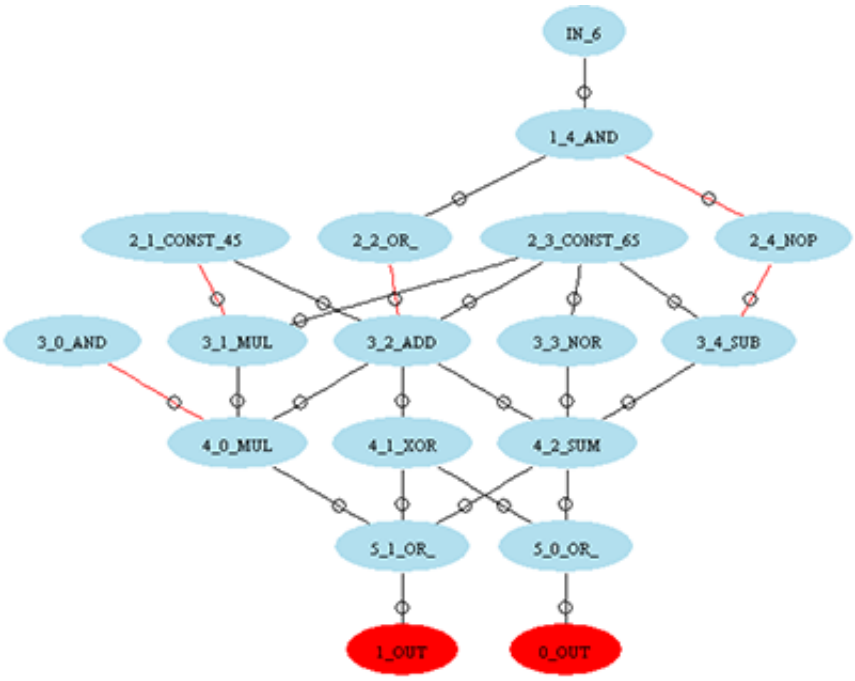


Obr. 5.2: Rozpoznávač obvodu Salsa - 1r - bias - 16B

hodného kľúča, výsledky sú takmer totožné s veľmi malým zhoršením.

Ako porovnávací test u našej aplikácii založenej na genetických evolučných obvodoch sme zvolili rozlišovanie medzi dvoma prúdami dát. Úlohou evolučného obvodu bolo rozhodnúť na základe znalosti len zašifrovaného textu, či sa jedná o prúd č.1, alebo č.2 (konkrétne prúdy sa dali nastaviť v konfiguračnom súbore). U testov pomocou našej aplikácie sme ukázali, že sa správajú podobne, ako štatistické testy, ale nie úplne rovnako. Náš obvod našiel rovnako problémy na šifrách Grain (1-2 kolá), Salsa20 (1-2 kolá), TSC (1-10 kôl), Lex (1-3 kolá) a Decim (1-4 kolá). U šifry Decim (5 kôl) ale na rozdiel od štatistických testov nenašiel žiaden problém. Bohužial sa nám nepodarilo naraziť na prípad, kde by náš evolučný obvod dokázal nájsť chybu v prípade, kde štatistický test žiadnu nájsť nedokázal. Toto mohlo byť spôsobené nesprávne zvoleným testom, nedostatočne dlhým behom evolúcie, alebo aj neschopnosťou nájdenia podobných výsledkov evolučným obvodom. Testy bežali vždy v 26 až 30 inštanciách naraz (v závislosti od dostupného procesorového času) po dobu približne 2 hodín. Každý z testov teda bežal približne 2 dni procesorového času. Dokopy naše testy zabrali okolo 300 dní čistého procesorového času.

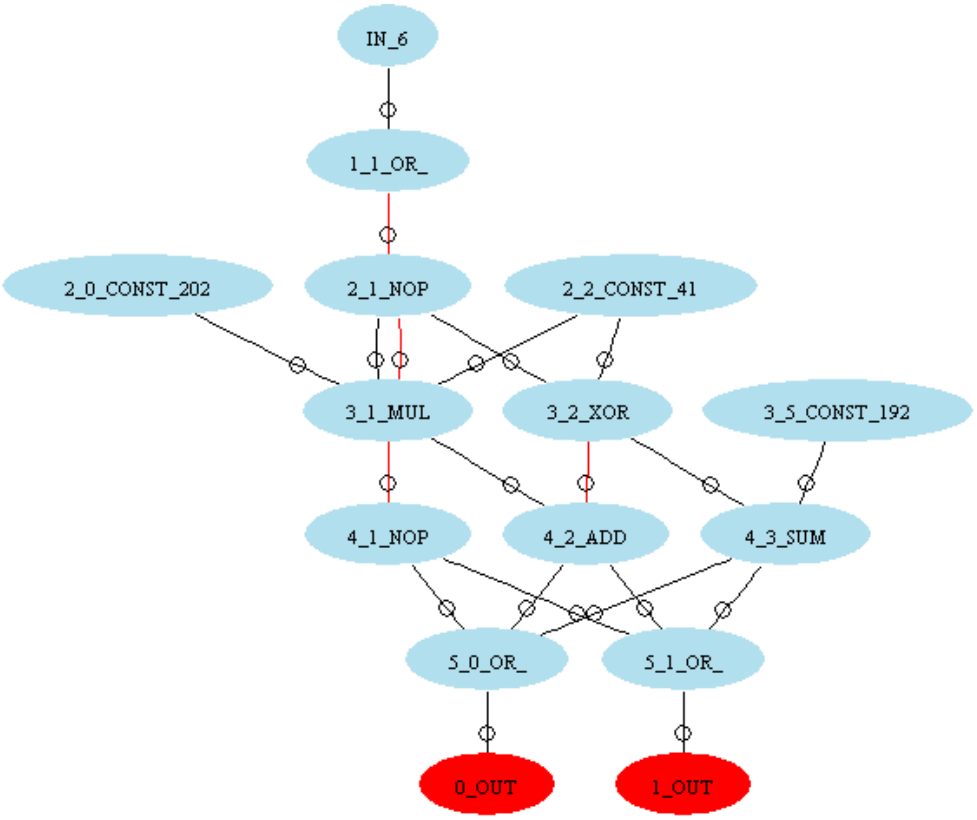
Ďalším zistením bol rozdiel vo výsledkoch medzi kvantovým a biasnutým generátorom a medzi 16B a 256B vstupným prúdom. Zatiaľ čo u štatistických testov sa výsledky prúdov generovaných z kvantového a biasnutého kľúča prakticky nelíšili, naša aplikácia tento rozdiel poznala. Výsledky sa líšili na úrovni niekoľko desiatok percent v čase potrebnom na získanie distinctoru (obvodu). Na druhú stranu rozdiely medzi výsledkami so vstupnými



Obr. 5.3: Rozpoznávač obvodu Salsa - 1r - bias - 16B

dátami o veľkosti 16B a 256B boli prakticky zanedbateľné. Toto mohlo byť spôsobené príliš rýchlou evolúciou, kde si algoritmus nestihol vyskúšať, ako sa správajú ďalšie vstupy obvodu.

V poslednej časti sme ukázali, že výsledné obvody našej aplikácie je možné jednoducho interpretovať. Ďalej sme poukázali na to, že šifru je možné testovať nie len tým, že hľadáme absolútny výsledok (napr. rozlišovač obvodov). Za výsledok sa dá považovať aj to, že sa obvod dokáže rýchlejšie naučiť rozpoznávať rovnakú vzorku dát (aj v prípade, že sa po zmene dát zistí, že obvod sa nič nenaučil). Toto sa nám stávalo u šifry Decim na 2 až 4 kolách a sčasti u šifry Salsa20 na 1 a 2 kolách.



Obr. 5.4: Rozpoznávač obvodu Salsa - 1r - bias - 16B

Kapitola 6

Záver

Diplomová práca sa zaoberá testovaním šifier projektu eSTREAM. Toto bolo založené na genetických evolučných obvodoch. Hlavným cieľom bolo okrem spomenutých testov aj príprava šifier pre testovanie a prispôsobenie testovacieho programu. Jej hlavné prínosy sú v ďalšom rozvoji myšlienky využitia evolučných algoritmov pre testovanie šifier. Konkrétne sa jednalo o návrh a vykonanie vhodných testov pre testovanie prúdových šifier a následné porovnanie s výsledkami štatistických testov. Ďalším prínosom je úprava a príprava aplikácie pre testovanie prakticky akejkoľvek šifry tak, aby boli jednotlivé funkčné bloky prehľadne oddelené. Do aplikácie sme pridali možnosť testovania už vyvinutých obvodov a ukázali použitie pri analýze niekoľkých výsledných obvodov. V teoretickej časti práce sa nachádza popis genetických algoritmov a ich skĺbenia s evolučnými obvodmi. Ďalej sa tu nachádza popis štatistických testov, a to najmä batérií NIST a DIEHARD. Poslednou časťou teórie je popis kandidátnych šifier projektu eSTREAM s dôrazom na už nájdené kryptoanalytické výsledky. Táto časť sa dá použiť, ako prehľad útokov na šifry projektu eSTREAM.

Čo sa týka pokračovania práce, tu je možné ísť viacerými cestami. V rámci projektu eSTREAM by bolo dobré rozobrať viac šifier na jednotlivé časti (posuvné registre) a otestovať ich takto obmedzené. V budúcnosti by naša testovacia aplikácia mohla podporovať viac možností testovania šifier (my sme používali len distinctor). Tak isto by mohla aplikácia podporovať viac rôznych šifier (napríklad aj starších návrhov). Úplná automatizácia testovania s predbežným vyhodnocovaním výsledkov (napr. na základe priemernej fitness celej evolúcie, alebo fitness najlepšieho jedinca po zmene testovacích vektorov) by značne znížila režijné náklady na spúšťanie a vyhodnocovanie testov. V tomto prípade by boli nutné určité úpravy rozhrania pre zadávanie úloh systému BOINC. Bez úprav aplikácie by bolo možné ešte spúšťať testy s rôznym nastavením evolučného obvodu tak, aby sa zistilo, pre aké najmenšie obvody je možné získať rovnaké výsledky, ako sme získali my. Na druhú stranu je stále možné, že niektoré výsledky nám zostali skryté z dôvodu príliš obmedzujúceho nastavenia obvodu. Tento proces sme začali, ale kvôli nedostatku času nedokončili a preto nie je v práci popisovaný. Jednalo sa hlavne o znižovanie veľkosti obvodov u oslabených šifier.

Praktické výstupy práce (zdrojové kódy a výsledky testov) sú umiestnené v elektronickej prílohe práce.

Literatúra

[1] Aumasson, J. a Fischer, S. a Khazaei, S. a Meier, W. a Rechberger, C.: *New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba*, Proceedings of FSE 2008, LNCS 5086, pp. 470-488, Springer, 978-3-540-71038-7, 2008. 3.20

[2] Babbage, S. a Cid, C. a Pramstaller, N. a Raddum, H.: *Cryptanalysis of Hermes8F*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <http://www.ecrypt.eu.org/stream/papersdir/2007/009.pdf> . 3.12

[3] Babbage, S. a De Canniere, C. a Canteaut, A. a Cid, C. a Gilbert, H. a Johansson, T. a Parker, M. a Preneel, B. a Rijmen, V. a Robshaw, M.: *The eSTREAM Portfolio*, eSTREAM, ECRYPT Stream Cipher Project, 2008, <http://www.ecrypt.eu.org/stream/portfolio.pdf> . 3.3

[4] Berbain, C. a Gilbert, H.: *Cryptanalysis of ABC*, eSTREAM, ECRYPT Stream Cipher Project, 2005, <http://www.ecrypt.eu.org/stream/papersdir/048.pdf> . 3.1

[5] Aumasson, J. a Fischer, S. a Khazaei, S. a Meier, W. a Rechberger, C.: *Serpent: A New Block Cipher Proposal*, Proceedings of FSE 1998, LNCS, volume 1372, pp. 222-238, Springer, 1998. 3.22

[6] Borghoff, J. a Knudsen, L. a Matusiewicz, K.: *Hill climbing algorithms and Trivium*, Proceedings of SAC 2010, LNCS, volume 6544, pp. 57-73, Springer, 978-3-642-19573-0, 2010. 3.23

[7] Feng, X. a Liu, J.: *Improved linear cryptanalysis of SOSEMANUK*, Proceedings of ICISC '09, LNCS 5984, pp. 101-106. Springer, 978-3-642-14422-6, 2009. 3.22

[8] Courtois, N.: *Cryptanalysis of Sfinks*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <http://www.ecrypt.eu.org/stream/papersdir/2006/002.pdf> . 3.21

[9] Dinur, I. a Shamir, A.: *Cube Attacks on Tweakable Black Box Polynomials*, Proceedings of Eurocrypt 2009, LNCS 5479 pp. 278-299, Springer, 978-3-642-01000-2, 2009. 3.23

[10] Englund, H. a Hell, M. a Johansson, T.: *Two General Attacks on Pomaranch-like Keystream Generators*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <http://www.ecrypt.eu.org/stream/papersdir/2007/001.pdf> . 3.17

[11] Feng, X. a Liu, J. a Zhou, Z. a Wu, C. a Feng, D.: *A byte-based guess and determine attack on SOSEMANUK*, Proceedings of Asiacrypt '10, LNCS 6477, pages 146-157. Springer, 978-3-642-17373-8_9, 2010. 3.22

[12] Fischer, S. a Khazaei, S. a Meier, W.: *Chosen IV statistical analysis for key recovery attacks on stream ciphers*, In Proceedings of Africacrypt 2008, LNCS 5023, pp. 236-245, Springer, 978-3-540-68159-5, 2008. 3.23

[13] Gierlichs, B. a Batina, L. a Clavier, C. a Eisenbarth, T. a Gouget, A. a Handschuh, H. a Kasper, T. a Lemke-Rust, K. a Mangard, S. a Moradi, A. a Oswald, E.: *Susceptibility of eSTREAM Candidates towards Side Channel Analysis*, eSTREAM, ECRYPT Stream Cipher Project, 2008, Dostupné z: <http://www.ecrypt.eu.org/stvl/sasc2008/> . 3.15

[14] Hell, M. a Johansson, T.: *Cryptanalysis of Achterbahn-128/80*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <www.ecrypt.eu.org/stream/papersdir/2006/054.pdf> . 3.2

[15] Hell, M. a Johansson, T.: *A Key Recovery Attack on Edon80*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <http://www.ecrypt.eu.org/stream/papersdir/2007/044.pdf> . 3.7

[16] Isobe, T. a Ohigashi, T. a Kuwakado, H. a Morii, M.: *How to Break Py and Pypy by a Chosen-IV Attack*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <www.ecrypt.eu.org/stream/papersdir/2007/035.pdf> . 3.18

[17] Kunzli, S. a Junod, P. a Meier, W.: *Distinguishing attacks on T-functions*, Mycrypt 2005, LNCS 3715, pp. 2-15, Springer, 2005. 3.24

[18] Künzli, S. a Meier, W.: *Distinguishing Attack on MAG*, eSTREAM, ECRYPT Stream Cipher Project, 2005, <http://www.ecrypt.eu.org/stream/papersdir/053.pdf> . 3.14

[19] Lebedev, A. a Starodubtzev, S. a Volchkov, A.: *Real Distinguishing of Yamb Output and a True Random Sequence*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <http://www.ecrypt.eu.org/stream/papersdir/2006/035.pdf> . 3.26

[20] Lu, Y. a Wang, H. a Ling, S.: *Cryptanalysis of Rabbit*, Proceedings of ISC 2008. LNCS 5222, pp. 204-214, Springer, 2008, 978-3-540-85886-7_14. 3.19

[21] McDonald, C. a Charnes, C. a Pieprzyk, J.: *An algebraic analysis of Trivium ciphers based on the boolean satisffability problem*, Cryptology ePrint Archive, Report 2007/129, 2007, Dostupné z: <http://eprint.iacr.org/2007/129> . 3.23

[22] Muller, F. a Peyrin, T.: *Linear Cryptanalysis of TSC Stream Ciphers - Applications to the ECRYPT proposal TSC-3*, eSTREAM, ECRYPT Stream Cipher Project, 2005, <http://www.ecrypt.eu.org/stream/papersdir/042.ps> . 3.24

[23] Rukhin, R. a Soto, J. a Nechvatal, J. a Smid, M. a Barker, E. a Leigh, S. a Levenson, M. a Vangel, M. a Banks, D. a Heckert, A. a Dray, J. a Vo, S.: *A Statistical Test Suite*

for Random and Pseudorandom Number Generators for Cryptographic Applications, National Institute of Standards and Technology, 2010, Dostupné z: <http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf> . 5.1

[24] Naya-Plasencia, M.: *Cryptanalysis of Achterbahn-128/80*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <www.ecrypt.eu.org/stream/papersdir/2007/019.pdf> . 3.2

[25] Paul, S. a Preneel, B. a Sekar, G.: *Distinguishing Attacks on the Stream Cipher Py*, eSTREAM, ECRYPT Stream Cipher Project, 2005, <http://www.ecrypt.eu.org/stream/papersdir/081.pdf> . 3.18

[26] Raddum, H.: *Cryptanalytic Results on Trivium*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps> . 3.23

[27] Tsunoo, Y. a Saito, T. a Kubo, H. a Shigeri, M.: *Cryptanalysis of Mir-1, a T-function Based Stream Cipher*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <http://www.ecrypt.eu.org/stream/papersdir/2006/020.pdf> . 3.16

[28] Tsunoo, Y. a Saito, T. a Kubo, H. a Suzaki, T. a Nakashima, H.: *Differential Cryptanalysis of Salsa20/8*, eSTREAM, ECRYPT Stream Cipher Project, 2007, <http://www.ecrypt.eu.org/stream/papersdir/2007/010.pdf> . 3.20

[29] Wu, H. a Preneel, B.: *Cryptanalysis of Stream Cipher DECIM*, eSTREAM, ECRYPT Stream Cipher Project, 2005, <http://www.ecrypt.eu.org/stream/papersdir/049.pdf> . 3.4

[30] Wu, H. a Preneel, B.: *Cryptanalysis of the Stream Cipher ABC v2*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <http://www.ecrypt.eu.org/stream/papersdir/2006/029.pdf> . 3.1

[31] Wu, H. a Preneel, B. a Preneel, G.: *Key Recovery Attack on Py and Pypy with Chosen IVs*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <www.ecrypt.eu.org/stream/papersdir/2006/052.pdf> . 3.18

[32] Zhang, H. a Li, L. a Wang, X.: *Fast Correlation Attack on Stream Cipher ABC v3*, eSTREAM, ECRYPT Stream Cipher Project, 2006, <www.ecrypt.eu.org/stream/papersdir/2006/049.pdf> . 3.1

Dodatok A

Obsah elektronickej prílohy

- `tests.zip` - Balíček obsahujúci všetky výsledky získané pomocou našej aplikácie a batérií NIST a DIEHARD popisované v práci.
- `sourcecode.zip` - Balíček obsahujúci zdrojové kódy našej aplikácie.
- `172546-xprist-dp.xml` - Zdrojový kód práce vo formáte xml.
- `172546-xprist-dp.tex` - Vygenerovaný kód práce vo formáte tex.
- `172546-xprist-dp.pdf` - Text práce vo formáte pdf.
- `uml.png` - Obrázok z práce (Obrázok 4.1).
- `obvodpriklad.png` - Obrázok z práce (Obrázok 2.1).
- `lex1rqrnd256.png` - Obrázok z práce (Obrázok 5.1).
- `salsa1rbias16.png` - Obrázok z práce (Obrázok 5.2).
- `salsa1rbias16-2.png` - Obrázok z práce (Obrázok 5.3).
- `salsa1rbias16-3.png` - Obrázok z práce (Obrázok 5.4).
- `testvect.png` - Obrázok z práce (Obrázok 4.2).