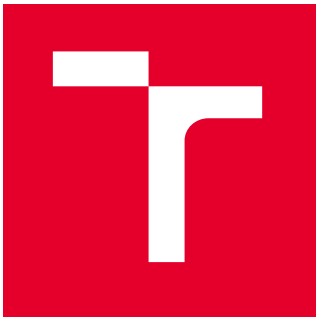


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

GENEROVÁNÍ NÁHODNÝCH ČÍSEL NA ČIPOVÝCH KARTÁCH

RANDOM NUMBER GENERATION ON SMART CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Radka Suchomelová**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Lukáš Malina, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Studentka: Radka Suchomelová ***ID:*** 200682

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Generování náhodných čísel na čipových kartách

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s technologií čipových karet. Nastudujte a porovnejte možnosti generování náhodných čísel na programovatelných čipových kartách. Dále nastudujte metody testování generátorů a náhodných čísel (např. Diehard, NIST, TestU01). Navrhněte vhodný postup pro automatizované testování výstupů generátorů a implementujte základy testovací aplikace pro ověřování kvality náhodných čísel produkovaných na čipových kartách.

Cílem bakalářské práce je vytvoření funkční aplikace, která i pomocí integrovaných standardních testů umožní ověřit celkovou kvalitu a bezpečnost metod generování náhodných čísel na čipových kartách.

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C VAN OORSCHOT a Scott A VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] RANKL, Wolfgang. Smart Card Applications: Design Models for Using and Programming Smart Cards. 2007. 217 s. ISBN 9780470058824.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

Abstrakt

Bakalářská práce je zaměřena na testování výstupů generátorů náhodných čísel na čipových kartách. Teoretická část práce se zabývá problematikou generování náhodných čísel a jejich testování a také popisuje několik druhů programovatelných čipových karet. Práce také porovnává různé nástroje pro testování kvality náhodnosti čísel a popisuje změnu Python implementace testovací sady NIST STS. Součástí práce je popis vytvořené aplikace pro automatické testování čísel generovaných kartami JavaCard a BasicCard. Touto aplikací bylo pro tuto práci otestováno 7 čipových karet. Testováním bylo zjištěno, že otestované generátory negenerují dokonale náhodné posloupnosti, ale jejich kvalita je ve většině případů poměrně ucházející.

Klíčová slova

Čipové karty, smart karty, BasicCard, JavaCard, náhodná čísla, testování generátorů náhodných čísel, NIST Statistical Test Suite

Abstract

This Bachelor's thesis focuses on the testing of the outputs of smart cards random number generators. The theoretical section of the thesis covers the problematics of random numbers generating and its testing, as well as descriptions of several types of programmable smart cards. The thesis compares various tools for testing the quality of randomnes and describes the changes in the Python implementation of the testing suite NIST STS. The thesis includes description of the created application for the automatic testing of the number generated by JavaCard and BasicCard cards. This application was used for the testing of 7 cards for this thesis. The testing revealed that the tested generators do not generate perfectly random sequences, but their quality is somewhat acceptable in most cases.

Keywords

Smart cards, BasicCard, JavaCard, Random numbers, Random number generators testing, NIST Statistical Test Suite

Bibliografická citace

SUCHOMELOVÁ, Radka. *Generování náhodných čísel na čipových kartách*. Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/152446. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Lukáš Malina.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Radka Suchomelová*
VUT ID studenta:	*200682*
Typ práce:	*Bakalářská práce*
Akademický rok:	*2022/23*
Téma závěrečné práce:	*Generování náhodných čísel na čipových kartách*

Prohlašuji, že svou závěrečnou práci jsem vypracovala samostatně pod vedením vedoucího závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 11. května 2023 ----------

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Lukáši Malinovi, Ph.D. za odborné vedení práce, poskytnuté rady, podněty a další přínosy při zpracování mé práce.

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Kvalita náhodných čísel je důležitá pro kryptograficky bezpečné zabezpečení dat. Čipové karty jsou součástí běžného života v podobě platebních karet, eObčanek nebo například SIM karet. Čipové karty často uchovávají citlivá data, která musí být patřičně kryptograficky chráněna. Bakalářská práce se věnuje možnostem generování náhodných čísel na různých platformách čipových karet a testováním čísel vygenerovaných kartami s BasicCard a JavaCard operačním systémem.

Cílem bakalářské práce je vytvoření aplikace, která pomocí statistických testů umožní automaticky testovat kvalitu výstupů generátorů náhodných čísel čipových karet.

Práce je dělena na teoretickou a praktickou část. Teoretická část práce popisuje programovatelné čipové karty, princip testování náhodnosti, druhy generátorů náhodných čísel požadavky, které se na generátory kladou, a také se zabývá generátory náhodných čísel na různých platformách čipových karet. Praktická část práce porovnává nástroje pro testování kvality náhodných čísel, popisuje způsob získávání dat z generátorů na kartách s BasicCard a JavaCard operačním systémem a popisuje vytvořenou aplikaci, která data z karet sbírá a ukládá do souboru a poté tento soubor otestuje statistickými testy pro testování kvality náhodnosti.

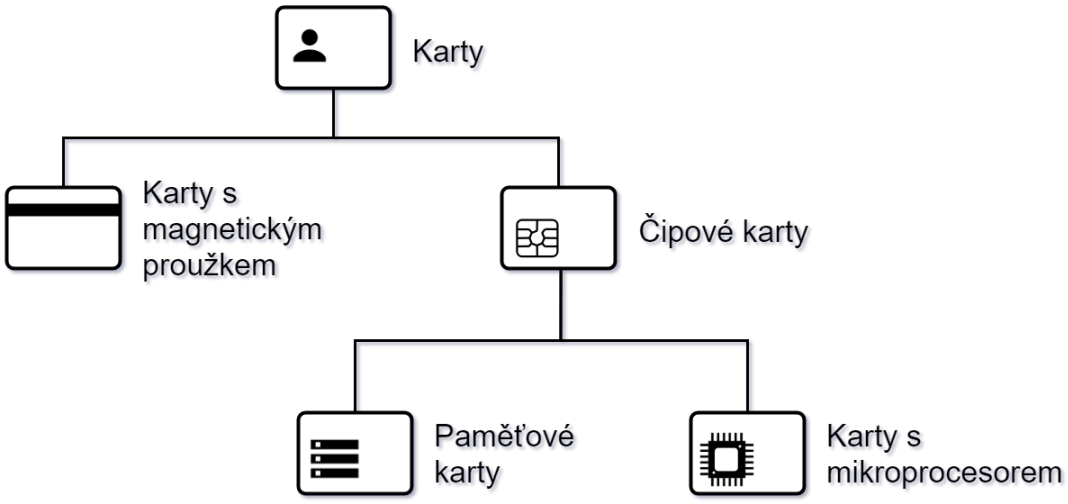
Kapitola 1 popisuje čipové karty, jejich dělení a tři platformy čipových karet, JavaCard, MultOS a BasicCard, vybrané pro tuto práci. Náhodná a pseudonáhodná čísla, způsob jejich generování, užití generátorů náhodných čísel a testování jejich kvality je představeno v kapitole 2. V této kapitole je také popsána možnost generování náhodných čísel na jednotlivých platformách a jejich srovnání.

Kapitola 3 se zabývá různými nástroji pro testování kvality generátorů a jejich možnosti použití na platformě Windows. Kapitola 4 představuje způsoby dosavadních testování generátorů náhodných karet na čipových kartách. Kapitola 5 popisuje proces vývoje a kompilace aplikací, které mohou být nahrány na čipové karty a které zasílají výstup generátorů terminálu. V 6. kapitole je představena samotná aplikace, která volá funkci aplikací z kapitoly 5, ukládá výstup těchto karet a poté testuje tento soubor statistickými testy. Soubor je testován upravenou implementací nástroje Randomness_testsuite, Python implementací testovací sady NIST Statistical Test Suite. V kapitole 7 jsou poté diskutovány výsledky otestovaných karet.

1. ČIPOVÉ KARTY

Čipové karty jsou plastové karty, které obsahují integrovaný obvod. Integrovaný obvod neboli čip slouží pro příjem a návrat dat. Čipové karty se dělí na paměťové karty a karty s mikrokontrolerem. Paměťové karty umožňují ukládat data pomocí paměťového čipu, ale neobsahují mikrokontroler, a proto s přijatými daty nemohou dále pracovat. Tato bakalářská práce věnuje kartám s mikrokontrolerem, kterým se říká mikroprocesorové.

Mezi čipové karty neboli „smart cards“ nepatří karty s magnetickým proužkem, které neobsahují čip a ze kterých se čipové karty vyvinuly. Karty s magnetickým proužkem jsou schopné ukládat data, ale paměť je nepřepisovatelná a neumí data zpracovávat. Toto dělení karet je znázorněno na obrázku 1.1 níže.



Obrázek 1.1 Dělení karet s čipem (integrovaným obvodem) a bez čipu

Pro zamezení opakování a snadnější čitelnost jsou čipové karty s mikroprocesorem dále v této bakalářské práci označovány pouze jako čipové karty. Popis nezahrnuje karty paměťové, pokud to není výslovně uvedeno.

Čipové karty slouží k identifikaci uživatelů a jejich autorizaci v rámci služeb. Umožňují ukládání dat a jejich zabezpečení pomocí kryptografických algoritmů.

Mezi čipové karty patří například SIM karty v telefonech, kreditní a debetní karty, věrností karty obchodních řetězců, karty pro řízení přístupu do chráněných místností či areálů, eObčanky nebo karty pro vedení docházky zaměstnanců.

Čtečkou čipových karet může být počítač, bankovní terminál, terminál pro čtení elektronických jízdenek nebo telefon. Čtečka dála může být napojena na další zařízení například počítač, s tímto zařízením potom může komunikovat jiným protokolem než se samotnou kartou.

Provedení funkce nebo přístup k datům na kartě může být podmíněno prokázáním znalosti PINu jeho zadáním do terminálu, nebo autentizací terminálu čipovou kartou. Metody autentizace jsou více rozebrány v kapitole 1.3, která se věnuje architektuře

programovatelných čipových karet. I paměťové karty mohou před čtením nebo zápisem dat požadovat splnění nějaké podmínky, typicky zadání PINu do terminálu, tyto podmínky přístupu jsou ale neměnné.

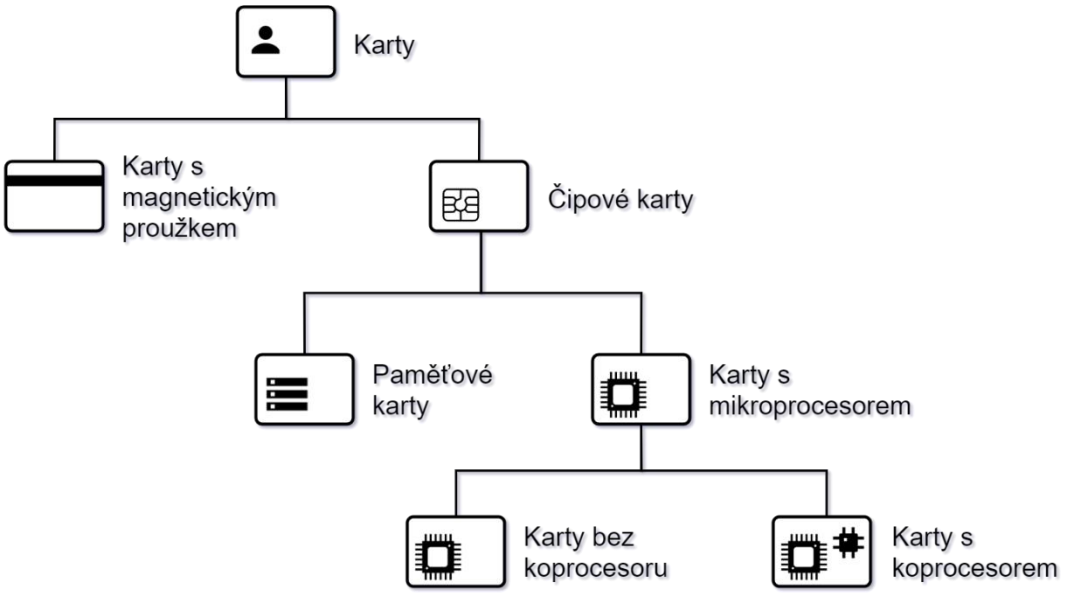
Čipové karty obsahují, kromě čipu s mikroprocesorem, ROM, RAM a EEPROM paměť. Mikroprocesor obsahuje vstupní a výstupní rozhraní. Více o architektuře v kapitole 1.3.

Z důvodu uložení kryptografických klíčů a jiných citlivých dat, je nutné zajistit, aby obsah karet nebylo možné zkopírovat. Tuto ochranu určuje standard ISO 7816-6. Více informací o čipových kartách [1] a zde [2]. Více o standardech ISO 7816 zde [3].

1.1 Druhy čipových karet

Jednotlivé čipové karty se různí v množství aplikací, jejich hardwaru, paměťových parametrech, kryptografických protokolech, způsobu komunikace s okolím i ve fyzických rozměrech. V této kapitole je uvedeno několik způsobů dělení karet a bližší popis skupin, které tímto dělením vznikají.

Další dělení karet podle jejich hardwaru je znázorněno na obrázku 1.2 níže. Z obrázku lze vidět, že se mikroprocesorové čipové karty dělí podle toho, jestli mají nebo nemají koprocesor. Koprocesor slouží mimo jiné k asymetrickému šifrování a umožňuje rychlejší kryptografické výpočty. Čipové karty obsahující kryptografické moduly pro asymetrickou kryptografii se nazývají PKI karty a často poskytují generování veřejného a soukromého klíče i výpočet otisku. Soukromí klíč se uloží do paměti karty a není přístupný. Toto umožňuje podepisování elektronickým podpisem, jenž provádí sama karta.

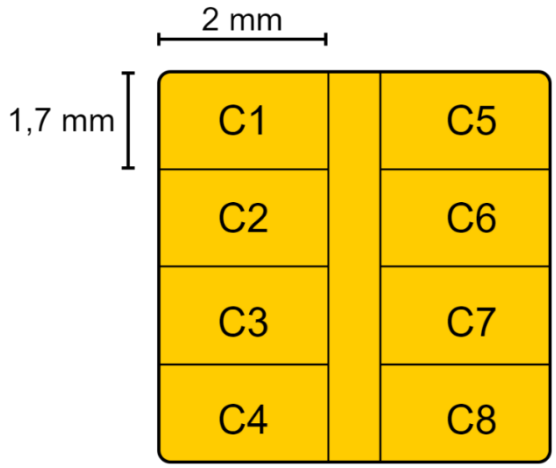


Obrázek 1.2 Podrobnější dělení čipových karet

Do tohoto dělení nebyly zahrnuty zařízení jako například USB tokeny nebo „super smart cards“. Tyto zařízení používají stejnou technologii, ale čipovými kartami nejsou. USB tokeny neboli „mini klíče“ se k počítači nepřipojují pomocí čtečky, ale jak název napovídá pomocí USB portu. „Super smart cards“ mají kromě mikroprocesoru zabudované uživatelské rozhraní, jako tlačítka nebo display [1].

Dalším parametrem pro rozdělení čipových karet může být způsob, kterým karty komunikují s čtečkami čipových karet. Tímto způsobem se dají rozdělit na karty kontaktní, bezkontaktní, hybridní a duální.

Kontaktní karty komunikují s terminálem pomocí kontaktní pozlacené oblasti, nejčastěji oválného nebo čtvercového tvaru, která umožňuje elektrické propojení terminálu s kartou. Zároveň je kontaktní oblast používána pro napájení. Kontaktní oblast se skládá z osmi kontaktů, z toho dva zatím nejsou využívané a slouží jako rezerva. Rozložení a účel jednotlivých kontaktů stanovuje norma ISO 7816-2. Pro komunikaci musí být karty vloženy do terminálu. Komunikace probíhá na principu half-duplex (poloviční duplex), to znamená, že dvě komunikující strany nemohou vysílat současně. V jednom okamžiku jedna strana vysílá a druhá přijímá. Komunikaci vždy zahajuje terminál. Na obrázku 1.3 níže je znázorněno, jak může vypadat ploška s kontakty a jejich stanovené minimální rozměry. Více o kontaktech čipových karet zde [3].



Obrázek 1.3 Možná podoba kontaktní plošky s minimálními rozměry kontaktů

Bezkontaktní karty nemusí být do terminálu vloženy, vyžadují pouze blízkou vzdálenost od terminálu. Tyto karty komunikují s terminálem pomocí elektromagnetického signálu a stejným způsobem jsou napájeny. Aby komunikace byla možná, musí být součástí karty anténa, která by signál přijímala a odesílala. Anténa bývá zabudovaná dovnitř plastového pouzdra karty. Tento způsob komunikace může být uživatelsky pohodlnější než u kontaktních karet, ale způsobuje několik problémů, které musí být ošetřeny při implementaci. Například je nutné vyřešit případné kolize, které mohou nastat, pokud je v blízkosti terminálu více karet. Dalším možným problémem je odstranění karty z požadovaného rozsahu dříve, než byla komunikace řádně ukončena,

což může vést k poškození některých souborů, do kterých terminál zapisoval data. Jedním z možných řešení tohoto problému jsou atomic procesy, které zajišťují, že pokud data nejsou kompletní, pak nebudou zapsána do trvalé (nepřechodné) paměti. Atomic procesy tak chrání soubory před poškozením, ale jejich použití výrazně prodlužuje dobu potřebnou pro zápis dat na kartu.

Hybridní karty umožňují komunikovat s terminálem kontaktně i bezkontaktně a to proto, že obsahují dva čipy, pro každý způsob komunikace jeden.

Duální karty stejně jako karty hybridní umožňují komunikovat oběma způsoby, ale na rozdíl od nich obsahují pouze jeden čip. Dvojí komunikace je možná díky dvěma nebo více typům vstupně-výstupního rozhraní. Existují také duální karty, které obsahují například dvě kontaktní rozhraní a bezkontaktně komunikovat neumí.

Podle množství aplikací, které podporuje čipová karta se karty dají dělit na jednoaplikační a multiaplikační.

Jedna aplikace, kterou jednoaplikační karty obsahují, může být po jejím nahrání neměnná nebo dynamická, jejíž funkce je možné měnit. Výroba jednoaplikačních karet je levnější než u multiaplikačních. Karta se statickou aplikací může být používána i jinak než pro funkce, pro které byla vyrobena, protože rozšíření funkcí může být provedeno na terminálu. V takovémto případě nová aplikace běží na terminálu a karta zůstává nezměněná.

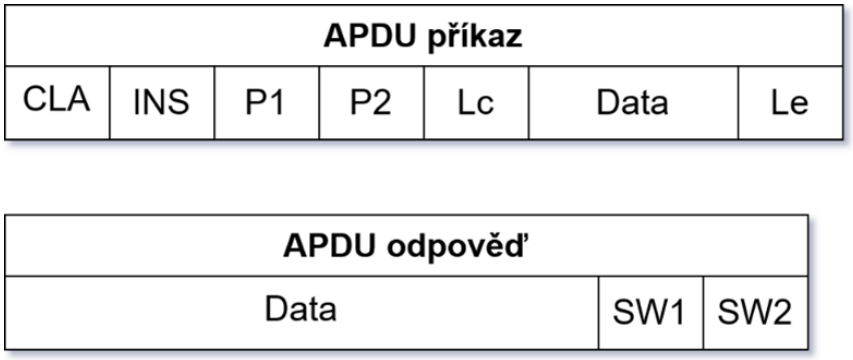
Multiaplikační karty, jak název napovídá, podporují více aplikací. Na rozdíl od karet jednoaplikačních musí mít operační systém. Tyto karty se dále dají rozdělit na statické karty s nahranými aplikacemi a karty dynamické, u kterých je možné aplikace měnit nebo nahrát nové. Z tohoto důvodu se jim také říká programovatelné čipové karty. Mezi nejznámější programovatelné čipové karty patří technologie JavaCard, MultOS a BasicCard. Architektura multiaplikačních dynamických čipových karet je blíže popsána v dalších kapitolách.

Čipové karty se dají dělit do skupin i podle fyzických rozměrů. Čipové karty jsou většinou kapesní velikosti, jejich rozměry se ale velmi různí. Jediný rozměr, který se málokdy mění, je tloušťka karty, ta bývá nejčastěji 0,76 mm. Velikosti karet se řídí standardy. Ty jsou důležité z toho důvodu, že kontaktní karty musí mít přesnou velikost, aby vyhovovaly velikosti otvoru v terminálu a aby se kontakt terminálu propojil s kontaktní ploškou karty. Časté jsou karty velikosti ID-1, šířka těchto karet bývá 8,6 centimetrů, což odpovídá velikosti klasické platební karty. Dalším častým rozměrem je šířka 2,5 centimetrů u D-000/Plug-in nebo 1,5 centimetrů u Mini-UICC karet, tyto rozměry jsou známé především kvůli SIM kartám pro GSM síť. Velikost kontaktní plošky se s velikostí karty nemění. Detailnější dělení čipových karet je uvedeno v literatuře [1] a [4].

1.2 Application Protocol Data Unit

Komunikační protokol APDU (Application Protocol Data Unit) se používá k přenosu dat mezi čipovou kartou a terminálem. Pomocí APDU příkazů karta dostává instrukce a APDU odpovědi informují terminál o tom, zda instrukce byla provedena úspěšně nebo jestli došlo k nějaké chybě a také může obsahovat data zaslaná kartou. Tyto příkazy jsou specifikovány v ISO/IEC 7816 standardu. V této kapitole bude popsáno, jakou podobu mají zprávy APDU a jejich význam. Níže na obrázku 1.4 je zobrazeno, z čeho se skládají APDU příkazy a APDU odpovědi.

Před prvním zasláním APDU příkazu kartě musí být sestaveno spojení s kartou. Při sestavování spojení karta zašle ATR (Answer To Reset), ve kterém poskytne terminálu informace o tom, který protokol má být použit pro komunikaci.



Obrázek 1.4 Struktura APDU zpráv

APDU příkazy jsou zprávy, které terminál zasílá kartě. Příkaz se skládá z povinné hlavičky a dat. Maximální délka příkazu je 260 bytů, z čehož 0 až 255 bytů může být využito pro data. Hlavička obsahuje byty:

- **CLA** – určuje třídu nebo typ příkazu
- **INS** – specifikuje konkrétní příkaz, který má být proveden
- **P1** a **P2** – slouží k předání parametrů příkazu

Nepovinné jsou byty:

- **Lc** – určuje délku dat, které následují (může mít délku 0-3 byty)
- **Data[]** – data, která mají být předána kartě
- **Le** – určuje maximální délku dat, které může karta poslat v APDU odpovědi

Karta na příchozí APDU příkazy posílá terminálu APDU odpovědi. Jejich maximální délka je 258 bytů. Odpovědi vždy musí obsahovat stavové byty. Stavové byty jsou zpětnou vazbou terminálu a oznamují, jak karta zpracovala APDU příkaz. Stavové byty jsou následující:

- **SW1** – informace o stavu operace
- **SW2** – podrobnější informace o chybách

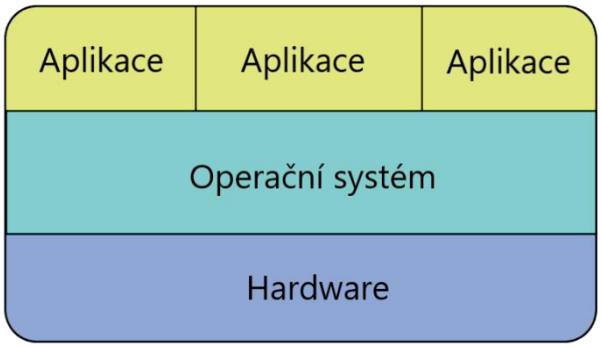
SW1 = 90 a SW2 = 00 znamená, že operace byla úspěšně dokončena bez chyb. Datová část APDU odpovědi obsahuje data, která jsou vrácena z karty:

- **Data[]** – maximálně 256 bytů

Více o APDU zprávách, komunikačních protokolech a ATR zde [1] a [4]. Více o ISO/IEC 7816 standardu zde [3].

1.3 Architektura programovatelných čipových karet

Architektura se pro různé čipové karty značně liší. Všechny programovatelné čipové karty ale musí obsahovat tyto tři prvky: hardware, operační systém a aplikace. Podle kvality a funkce programovatelné čipové karty také mohou obsahovat virtuální stroj, firewall, interpret a běhové prostředí. Na obrázku 1.5 jsou zobrazeny základní vrstvy architektury programovatelných čipových karet.



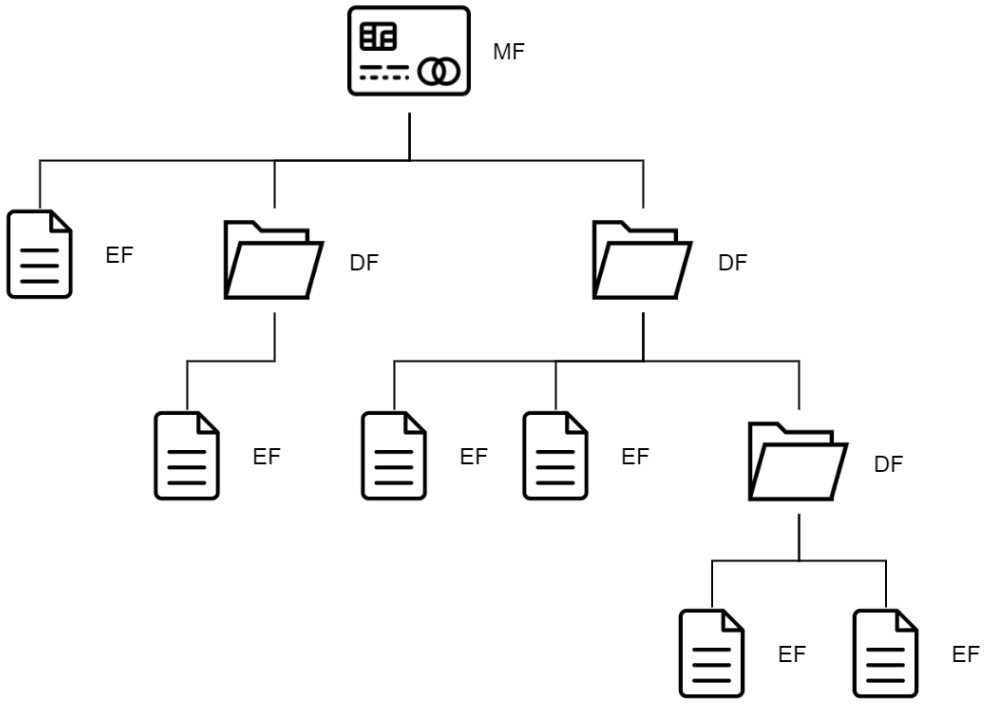
Obrázek 1.5 Základní architektura dynamických multiaplikačních karet

Hardware je fyzické vybavení karty, vykonává instrukce operačního systému pomocí assembleru, překladače symbolických instrukcí a adres. Karty můžou mít dodatečný hardware, například generátor hodinového signálu pro lepší komunikaci s terminálem v případě bezkontaktního přenosu, nebo hardware podporující další kryptografické funkce, tím může být generátor náhodných čísel.

Operační systém umožňuje nahrávání a mazání aplikací z karty, spravuje souborový systém a řídí reakce na příchozí ADPU zprávy z terminálu. Operační systém také slouží jako rozhraní mezi hardwarem a samotnými aplikacemi. To do jisté míry umožňuje vytváření aplikací pro programovatelné čipové karty bez toho, aby developer dopředu věděl přesné parametry hardwaru dané karty. Díky operačnímu systému totiž aplikace nemusí pracovat přímo s hardwarem. Operační systém bývá uložen v paměti ROM (Read Only Memory).

Přesný popis operačních systémů je složitý, protože existuje mnoho společností, které vytváří operační systémy pro čipové karty a tyto operační systémy se navzájem značně

liší. Jako příklad se dají uvést karety, jejichž operační systém je napsán v assembleru a mají 8bitový mikroprocesor. Toto můžou být karty, které řídí přístup do budov. Oproti tomu existují karty s 32bitový mikroprocesorem, jejichž operační systém je napsán v jazyce Java nebo C++, tyto karty se používají v telekomunikaci, kde se požaduje, aby operační systém poskytoval komplexnější funkce. V příštích kapitolách budou detailněji rozebrány tři různé technologie programovatelných karet.



Obrázek 1.6 Struktura dat na čipové kartě s kořenovým adresář MF, podadresáři DF a soubory s daty EF

Fyzická struktura dat na kartě je zachycená na obrázku 1.6. Kořenový adresář, který je u čipových karet nazývaný Mastery file (MF), je vždy pouze jeden. Přímo v kořenovém adresáři data nikdy nemohou být uložena, místo toho obsahuje soubory. Jedním typem souborů jsou podadresáře, takzvané Dedicated files (DF). Druhým typem jsou soubory obsahující samotná data, těm se říká Elementary files (EF).

Datové soubory musí vždy patřit do podadresáře nebo kořenového adresáře, není možné, aby soubor byl uložen mimo stromové struktury. Podadresáře můžou obsahovat několik datových souborů nebo další podadresáře. Podadresář často obsahuje soubory, které ukládají data pouze k jedné určité aplikaci. Jednotlivé datové soubory obsahují data k operačnímu systému a k aplikacím a ukládají se do nich klíče a certifikáty. Některé datové soubory slouží pro běh operačního systému a není možné k nim přistupovat, to jsou například uložené klíče. Jiné soubory jsou přístupné pomocí příkazů. Jestli daný datový soubor bude přístupný nebo ne určuje atribut souboru nebo podadresáře. Každý soubor a adresář má přiřazen dvoubytový identifikační kód, který operačnímu systému umožňuje pracovat se soubory a nahrazuje klasické pojmenovávání souborů.

Přístupová práva určují, jaké podmínky musí být splněny pro čtení a zápis dat na kartu. Tyto podmínky jsou nejčastěji autentizace pomocí PINu nebo autentizace terminálu čipovou kartou. Existují dva možné způsoby uplatňování podmínek přístupu. První metoda je podmínit oprávnění číst a oprávnění zápisu každé zvlášť. Druhá metoda je založena na tom, že každý datový soubor a podadresář má vlastní přístupová práva, která jsou všechna uložena spolu s identifikačním kódem souboru ve speciálním datovém souboru. Soubor, ve kterém jsou uloženy pravidla přístupu, musí mít správně nastaveny oprávnění pro zápis, aby neautorizovaní subjekt nemohl tyto pravidla změnit.

Logická struktura umožňuje jednodušší správu dat, například těch kryptografických. Klíče jsou uloženy každý zvlášť v datovém souboru v adresáři. Pro držitele karty je výhodnější a uživatelsky pohodlnější mít k sobě patřící soubory na jednom místě. Logické adresování sdružuje dvojice soukromý-veřejný klíč nebo celý certifikát do kontejnerů. Součástí softwaru k administraci čipových karet je většinou i utilita, která umožňuje zobrazovat a spravovat jednotlivé kontejnery.

Programovatelné čipové karty mohou obsahovat interpret. Ten slouží pro překlad stáhnutého kódu, díku tomu je možné, po úspěšné bezpečnostní kontrole, doplnit na kartu dodatečné aplikace bez toho, aby musel být upravován samotný operační systém.

Aplikace pro svůj běh využívají paměť RAM (Random Access Memory) pro uložení dočasných dat během operací. Samotné aplikace jsou uloženy v paměti EEROM, tato paměť je sice přepisovatelná, ale přepis dat je omezený rychlostí a také tím, že smazání a zápis dat není možné provádět donekonečna, protože počet těchto operací je omezený. Více o architektuře zde čipových karet zde [1] a [4].

1.3.1 JavaCard

JavaCard je velmi rozšířená technologie, která se používá pro čipové karty nebo jiné autentizační tokeny. Vývoj provádí společnost Oracle Corporation. JavaCard funguje jako otevřená platforma, to znamená, že na čipovou kartu mohou být nahrány aplikace třetích stran, proto existuje velké množství verzí JavaCard technologie. Aplikace pro čipové karty s touto technologií jsou psané v jazyku Java.

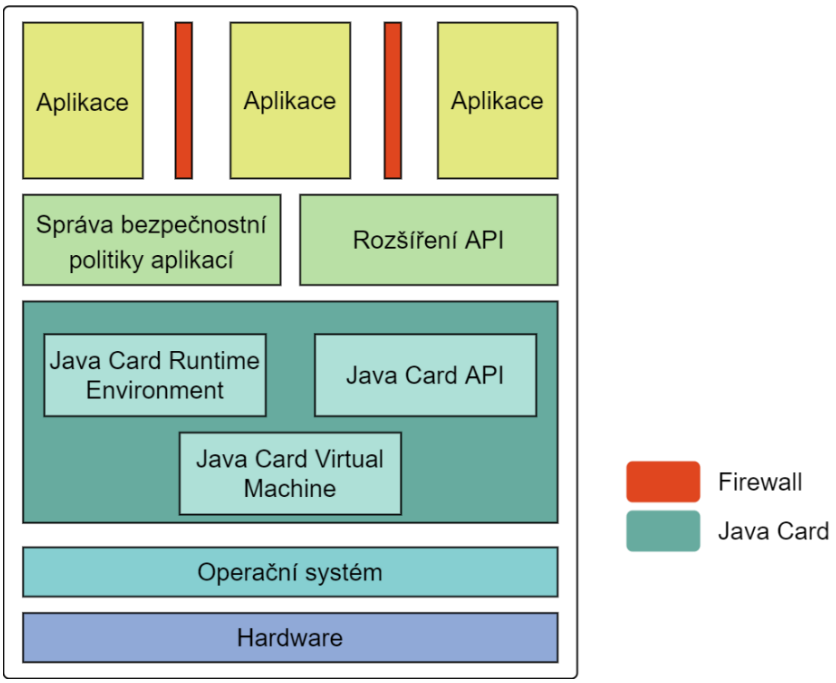
Java je pro čipové karty vhodný jazyk, protože byla vyvinutá také pro elektrospotřebiče jako chytré topinkovače nebo kávovary, v těchto zařízeních bývají mikroprocesory stejně jako na čipových kartách. Programy psané v Javě jsou nezávislé na hardwaru, a tak je jejich implementace na různé čipové karty snazší. Java vyžaduje poměrně velkou operační paměť, proto se v čipových kartách nenachází veškeré balíčky a knihovny, ale pouze část jazyku Java, která je využitelná na této platformě. Z tohoto důvodu na kartě není možné využívat například více rozměrných polí, vláken a velké datové typy. V případě potřeby určité funkce, která není podporovaná základním API (Application Programming Interface) JavaCard technologie umožňuje stáhnutí přídavných balíčků a knihoven.

Po vytvoření aplikace musí kód projít kontrolou, než je nahrán na samotnou kartu. Aplikacím na JavaCard kartách se říká applety. Applety jsou z bezpečnostních důvodů od sebe zvlášť odděleny appletem firewall, aby jedna nemohla zasáhnout do funkcí druhé a aby nemohly ovlivnit funkci operačního systému. Díky tomuto oddělení je možné, aby na jedné kartě společně fungovaly aplikace z různých zdrojů.

Pro vývoj appletů a pro implementaci JavaCard technologie Oracle poskytuje JavaCard Development Kit (JCDK), sadu nástrojů (toolkit) pro testování a debugging nových appletů pomocí simulátoru a nástroje pro ověřování appletů před nahráním na kartu.

Tři hlavní prvky JavaCard technologie jsou JavaCard Runtime Environment, JavaCard Virtual Machine a JavaCard Application Programing Interface. Jejich zařazení do architektury karet je znázorněno na obrázku 1.7.

JavaCard Runtime Environment (JCRE) definuje chování běhového prostředí při spouštění appletů, oddělení aplikací firewallem, správu paměti a oddělení a mazání souborů vytvořených terminálem.



Obrázek 1.7 Architektura čipové karty s JavaCard technologií

JavaCard Virtual Machine (JCVM) je interpret, který slouží jako virtuální procesor. Podobně jako skutečný procesor má vlastní sadu instrukcí, čítač instrukcí a zásobník. Překládá Java bytecode do strojového kódu. JCVM také zodpovídá za dohled nad objekty. Pokud odhalí porušení stanovených bezpečnostních pravidel, vyvolá výjimku a zastaví zpracovávání kódu, ve kterém nastal problém.

JavaCard Application Programming Interface (JCAPI) definuje balíčky a třídy potřebné pro základní operace nebo užitečné při programování aplikací pro čipové karty

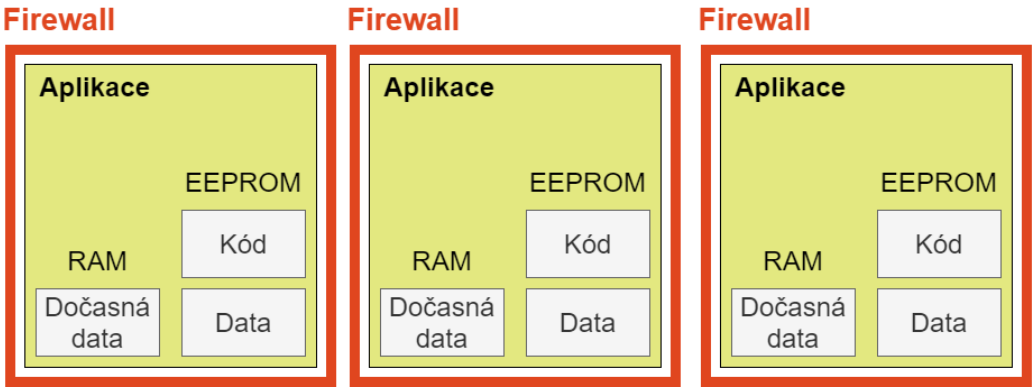
s JavaCard technologií. Tyto základní balíčky můžou být dále rozšířeny nebo přidány nové podle potřeby.

JavaCard Development Kit je možné zdarma stáhnout na stránkách společnosti Oracle. Více informací o technologii JavaCard zde [4], [5], [6] a [7].

1.3.2 MultOS

MultOS je operační systém, který byl vytvořen převážně pro čipové karty. Aplikace pro MultOS lze vytvářet v programovacím jazyku MEL, C, nebo v Javě. Kód je poté přeložen pomocí kompilátoru do MEL (MultOS Executable Language), jazyka nezávislého na použitém hardwaru. Vydavatel specifikací, licencí a certifikačních služeb pro MultOS sytém je Maosco Consortium. MultOS byl vyvinut ze systému pro elektronické peněženky Modex.

Než je aplikace přesunuta na kartu, musí být podepsaná pomocí licencované MultOS certifikační služby. Přítomnost certifikačního klíče při nahrávání aplikace kontroluje sama karta. Podobný postup je vyžadován při odstraňování aplikací z karty.



Obrázek 1.8 Aplikace na MultOS

Aby karty mohly obsahovat aplikace od různých společností bez narušení bezpečnosti, jsou aplikace od sebe odděleny firewallem podobně jako u JavaCard technologie. Kód aplikace je uložen ve statické paměti, do které aplikace nemůže zapisovat. Data potřebná pro běh aplikace jsou také uložena ve statické paměti, ale aplikace tyto data může upravovat a zapisovat do paměti. Dočasná data aplikace, například hodnoty proměnných, jsou uložena v relační paměti RAM. Aplikace nemůžou do sebe navzájem zasahovat, protože každá má přidělený svůj datový a kódový prostor v paměti. Virtuální stroj (Application Abstract Machine) aplikaci ukončí, pokud detekuje přístup jedné aplikace do datového prostoru jiné. Oddělení aplikací a jejich datové prostory jsou znázorněny na obrázku 1.8.

Virtuální stroj, nezávislý na dostupném hardwaru, přistupuje k funkcím operačního systému, psanému ve strojovém kódu, a poskytuje instrukce API. Operační systém se stará o virtuální stroj, řídí komunikaci mezi procesy a spravuje paměť, aplikace a příchozí příkazy od terminálu.

MultOS podporuje množství kryptografických algoritmů, například hashovací funkce SHA1 a SHA2, symetrické algoritmy DES, 3DES, AES a asymetrické algoritmy RSA a algoritmy využívající kryptografii nad eliptickými křivkami. Více informací o MultOS a kartách s tímto operačním systémem zde [4], [8] a [9].

1.3.3 BasicCard

BasicCard je operační systém pro čipové karty, který poskytuje německá ZeitControl, což je oproti ostatním zmíněným vývojářům operačních systémů pro čipové karty, malá společnost. BasicCard nabízí od roku 1992, ale jejich popularita v posledních letech klesá. BasicCard karty jsou programovatelné v jazyku ZC-Basic, který je poté kompilátorem převeden na P-code.

Do jazyku Basic bylo doplněno několik speciálních funkcí zvlášť pro čipové karty, například komunikační protokoly a rozhraní pro přístup do souborového systému. Kód napsaný v jazyku ZC-Basic je na čipových kartách v porovnání s ostatními jazyky krátký a rychle přehratelný, protože neobsahuje složité bezpečnostní mechanismy a je jednoduše interpretovatelný.

Pro bezpečnou komunikaci BasicCard poskytuje kryptografické algoritmy DES, 3DES, IDEA, AES, RSA, podporuje eliptické křivky s délkou klíče 167 bitů a hashovací funkci SHA1.

BasicCard karty se řadí do tří skupin. Enhanced, Professional a MultiApplication. První dvě skupiny karet jsou jednoaplikační, MultiApplication karty mohou aplikací obsahovat několik, jsou tedy multiaplikační.

BasicCard operační systém je vhodný pro potřebu jednoduše a rychle programovatelných karet, které nepotřebují složitější funkce. BasicCard operační systém je dostupný ve více variantách, které se liší ve svých funkcích, kryptografických algoritmech, komunikačních rozhraních a také podle toho, jaké vlastnosti se nejvíce hodí pro daný hardware a paměť.

Stejně jako Java Card a MultOS, BasicCard karty umožňují stahování programových kódů, které poskytují třetí strany. Různé varianty BasicCard můžou být zdarma stáhnuté z webových stránek společnosti ZeitControl spolu s nástroji pro vývojáře aplikací. Více o BasicCard zde [4], [10] a [11].

2. NÁHODNÁ ČÍSLA

Náhodné číslo je číslo získané pomocí procesu nebo jevu, jehož výsledek se nedá předpovědět. Opakováním tohoto procesu nedojde k získání stejného výsledku s větší pravděpodobností než jakéhokoli jiného výsledku, je tedy nereprodukovatelný. Náhodná čísla mají dvě základní vlastnosti, nepredikovatelnost a rovnoměrné rozdělení. Typickým příkladem získání náhodného čísla je hod kostkou. Číslo, které padne, není možné předpokládat a všechny čísla na kostce padají se stejnou pravděpodobností.

Zdrojem náhodných čísel jsou generátory náhodných čísel neboli RNG (Random Number Generator). Na základně procesu generování náhodného čísla je možné zdroje náhodných čísel rozdělit na fyzikální a algoritmické. Generátory náhodných čísel a jejich dělení i s příklady je popsáno v dalších podkapitolách.

Pravděpodobnost a získávání náhodných čísel začalo být zkoumáno kvůli hře v kostky v 17. století francouzským matematikem Pierre de Fermatem. Získávání náhodných čísel bylo studováno z části pro uplatnění v hazardních hrách. Například roku 1957 britská vláda zvolila pro losování státní loterie počítač. Oproti předchozím loteriím, tato měla velký počet menších výher a možnost až 100 milionů losů, z tohoto důvodu bylo místo klasického losování tedy vybrán počítač přímo určený pro generování náhodných čísel ERNIE (Electronic Random Number Indicator Equipment) [12].

Užití náhodných čísel je věnována podkapitola 2.2. Více o náhodných číslech zde [13], [14] a [15].

2.1 Generátory náhodných čísel

Už bylo uvedeno, že generátorem náhodných čísel (RNG) je proces, kterým se získávají náhodná čísla. Generátorem může být fyzikální jev, na jehož základě se určitým způsobem získá číslo, nebo algoritmus. Protože pro počítače jsou čísla posloupnost bitů, generátorem náhodných čísel je často generátor náhodných bitů (RBG).

Po vygenerování jednoho čísla není možné určit, jestli je toto číslo náhodné. Při zkoumání generátoru se proto pracuje s větším množstvím vygenerovaných čísel nazývaným posloupnost náhodných čísel.

Kvalitu čísel, které generuje RNG, a tedy kvalitu samotného generátoru zkoumají testy náhodnosti. Jednotlivé testy jsou často seskupeny do sad testů pro ověření více vlastností generátoru. Několik sad testů je blíže rozebráno v kapitolách 2.4.1 až 2.4.4.

Aby se proces generování dal označit jako generátor náhodných čísel, generovaná posloupnost musí splňovat několik požadavků:

- Rovnoměrné rozdělení hodnot
- Nepředvídatelnost
- Nezávislost jednotlivých hodnot na jiných hodnotách

Generátory se hodnotí podle jejich hodnoty entropie. Entropie popisuje průměrné množství informace, které nese posloupnost či číslo. Entropie udává míru náhodnosti, tedy jak nepředvídatelná hodnota je. Čím vyšší je hodnota entropie, tím je náhodnost generátoru vyšší. Výpočet entropie H se provádí vztahem

$$H = -\sum_{i=1}^{q} p_i \log_2 p_i \quad \text{[Sh/symbol]}, \tag{2.1}$$

kde p_i značí pravděpodobnost prvku i a q představuje počet všech možných prvků.

Maximální entropie je možné dosáhnout pouze pokud jsou všechna čísla generována se stejnou pravděpodobností. Po dosazení stejné pravděpodobnosti p u všech prvků do vzorce (2.1) je získán vztah

$$H_{max} = \log_2 p \quad \text{[Sh/symbol]} \tag{2.2}$$

pro maximální entropii generátoru.

Jako příklad maximální entropie může být RNG, který generuje 0 nebo 1 se stejnou pravděpodobností, tedy ve vygenerované posloupnosti je počet nul polovinou délky posloupnosti. Entropie tohoto generátoru by byla 1 Shannon na symbol.

U generátorů založených na matematických algoritmech, které budou blíže popsány v podkapitole 2.1.2, se při hodnocení kvality zkoumá perioda. Perioda u RNG vyjadřuje, jak často se čísla mohou opakovat, je tedy žádoucí, aby perioda byla co nejvyšší. U generátorů, které čísla generují na základě fyzikálního jevu, se hodnota periody pokládá za nekonečno.

Dalšími parametry pro zvolení typu generátoru pro určitý účel může být rychlost generování a parametry dostupného procesoru, tedy schopnost generovat čísla pomocí určitého jevu nebo procesu.

Podle potřeby se generátory mohou dělit ještě na ty, které generují spojité posloupnosti a ty, které generují diskrétní. Více informací o RNG zde [13], [14] a [15].

2.1.1 Fyzikální generátory

Fyzikální generátory generují true-random posloupnosti (opravdu náhodné). Jejich zkratka je TRNG (True Random Number Generator). Generování náhodných čísel je založeno na určitém fyzikálním jevu, u kterého není možné předpovědět výsledek. Samotné fyzikální jevy jsou popsané jako náhodné fyzikálními zákony. Těmito fyzikálními jevy můžou být tepelný šum, chování fotonů (kvantové generátory), poločas rozpadu radioaktivních částic, nebo šum spojený s elektronickými obvody.

Mezi ne zcela, ale téměř náhodné, těžko popsatelné parametry, patří generátory založené na meteorických jevech nebo na sledování lidského chování, například pohyb myši, čas úderu do klávesy, nebo losovací zařízení. Jako zdroj též může sloužit část lidského těla, například duhovka nebo otisk prstu. Tyto systémy generování jsou deterministické, jsou ale považovány za náhodné, protože sdílí vlastnosti s TRNG.

TRNG mají tu výhodu, že generování je neopakovatelné, to může být ale žádoucí v případě využívání generátorů pro simulace. Další výhodou je větší bezpečnost než

u algoritmických, které generují pseudo-náhodná čísla. Nevýhodou je obtížná realizace na některých zařízeních. Fyzikální generátory musí být součástí hardwaru, jinak je nutné použít generátory algoritmické. Výraznou nevýhodou je delší čas generování, to může být, podle požadavků na generátor, rozhodující faktor pro volbu rychlejších algoritmických.

U fyzikálních generátorů se sleduje jejich nezávislost na okolí a to, jestli nebyly poškozeny. Je důležité, aby generování neovlivňovali například změny teploty nebo magnetického pole. U generátorů, u kterých by hrozilo, že jsou nestabilní, se provádí odstínění.

Čísla vygenerované pomocí fyzikálních generátorů jsou někdy využívané jako vstupní parametry pro generátory algoritmické. Tento postup se volí v případě, že na čísla jsou kladeny další požadavky, například v kryptografii, nebo když daný hardware není uzpůsobený ke generování true-random čísel.

2.1.2 Algoritmické generátory

Algoritmické generátory, jinak PRNG (PseudoRandom Number Generator), generují pseudonáhodná čísla, tedy čísla, která se zdají být náhodná, ale ve skutečnosti jsou získávány deterministickým způsobem. Proces generování je založen na matematických funkcích. Generátory jsou tedy programy, které pomocí matematických funkcí vracejí výsledky. Použité funkce jsou často jednosměrné a malá změna parametrů se projeví výraznou změnou výsledku. PRNG vyžaduje jeden nebo více vstupních parametrů, anglicky nazývané seeds, které jsou použity v matematických funkcích. Aby výstup byl skutečně náhodný, musí být vstup také náhodný. Proto se často používá jako vstup čísla vygenerované pomocí TRNG. Pokud je seed známo, generování posloupnosti je opakovatelné.

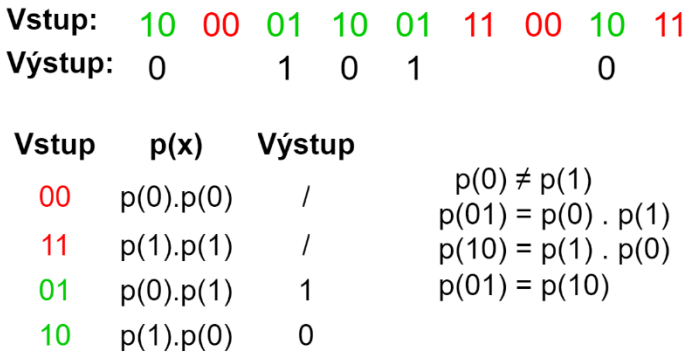
Generování je možná zřetězit tak, že výstupní hodnota je použita jako vstupní parametr pro další funkci. Použité rekurentní algoritmy jsou velice rychlé a generují náhodná čísla v reálném čase.

Už bylo uvedeno, že na rozdíl od generátorů fyzikálních, u algoritmických je sledována hodnota periody. Ta by měla být co nejvyšší.

Vlastnosti generátoru mohou být vylepšovány pomocí extraktoru. Extraktory jsou algoritmy, pomocí kterých je možné dosáhnout téměř rovnoměrné rozložení tím, že některé méně náhodné hodnoty jsou zahozeny či upraveny. To samozřejmě způsobuje to, že výsledná posloupnost je výrazně kratší než ta původní. Mezi známé extraktory patří extraktor pomocí XOR operace a von Neumannův extraktor.

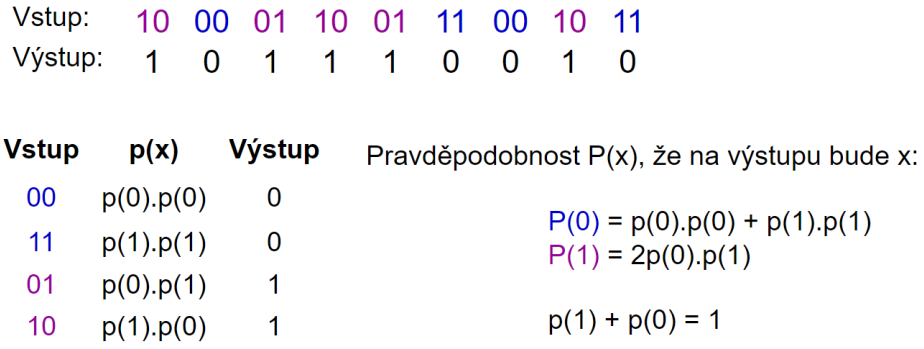
Von Neumannův extraktor funguje na principu eliminace dvojic bitů, které obsahují stejné prvky. Zbylé dvojice jsou změněny na jeden bit, a to podle toho, v jakém pořadí se ve dvojici objevují prvky. Tento postup řeší situace, kdy prvky nejsou generovány se stejnou pravděpodobností, protože pravděpodobnost vygenerování dvojice 01 je stejná,

jako vygenerování 10. Schéma fungování von Neumannova extraktoru je zobrazeno na obrázku 2.1.



Obrázek 2.1 Vyrovnání pravděpodobnosti bitů von Neumannovou metodou

Upravování vlastností a opravování chyb XOR operací se provádí také u generátorů fyzikálních. XOR operace je logická funkce založená na exkluzivním součtu. Stejně jako von Neumannův extraktor, tento extraktor upravuje generování bitů s rozdílnou pravděpodobností. Dvojice bitů vytvoří po XOR operaci jeden výstupní bit. Po provedení XOR operace se dvojice 00 a 11 převede na 0 a dvojice 01 a 10 vytvoří 1. Z tohoto je zřejmé, že je nutné vygenerovat dvakrát více bitů, než je potřeba ve výsledné posloupnosti. Převádění dvojic bitů XOR metodou a vliv na pravděpodobnost výstupních bitů je uvedeno na obrázku 2.2.



Obrázek 2.2 Zlepšení pravděpodobnosti bitů pomocí XOR operace

Výhodou algoritmických generátorů je rychlost a snadná realizace. Pro implementaci PRNG není potřebný speciální hardware, tak jak je tomu u TRNG, pouze program pro generování. Nevýhodou ovšem je nízká periodicita, která u TRNG není zkoumaná a považuje se za nekonečnou.

2.1.3 Kombinované

Už bylo zmíněno, že je běžné kombinovat fyzikální a algoritmické generátory. Zkombinováním těchto dvou typů je možné částečně eliminovat nevýhody a získat bitové

posloupnosti, které mají rovnoměrnou četnost jedniček a nul. Pro některá užití je nutné, aby generované posloupnosti měly určité vlastnosti, toho lze dosáhnout právě kombinovanými generátory.

Algoritmické generátory potřebují výchozí hodnoty, které by měly pocházet z TRNG. Fyzikální generátory generují pravá náhodná čísla, ale jsou náročné na realizaci, generování trvá dlouho a vygenerovaná čísla nemusí mít ty vlastnosti, jaké jsou potřeba. Algoritmické jsou oproti nim rychlé a jednouché na implementaci, ale je u nich problém s periodicitou.

Kombinované generátory fungují na principu XOR operace mezi true-random posloupností z fyzikálního generátoru a pseudonáhodnou posloupností, kterou vygeneroval program. Kvůli tomu, že pro posloupnost získanou z kombinovaného generátoru je nutné vygenerovat náhodnou posloupnost z fyzikálního generátoru, je zřejmé, že i tyto generátory budou pomalé.

2.2 Užití generátorů

Jedno z nejčastějších využití nacházejí generátory náhodných čísel v kryptografii. Náhodná čísla jsou stěžejní při generování kryptografických klíčů a jejich kvalita se výrazně odráží v bezpečnosti samotné šifry. Užití generátorů v kryptografii a na čipových kartách je věnována podkapitola 2.2.1. Náhodná čísla jsou také využívána v hazardních hrách a matematických simulacích.

Rozšířená je matematická metoda Monte Carlo, která využívá pseudonáhodná čísla a díky které je možné řešit určitý typ integrálů a používá se při optimalizačních úlohách.

Podle toho, k čemu jsou náhodná čísla použita, může být zvolen typ generátoru. Například pro matematické simulace se často používají generátory pseudonáhodných čísel, protože je žádoucí, aby generování posloupnosti bylo opakovatelné. To je u algoritmických generátorů možné, protože při stejných vstupních parametrech seeds, je vygenerována stejná posloupnost.

2.2.1 Užití generátorů v kryptografii a na čipových kartách

Kryptografie umožňuje dvěma stranám komunikovat bez toho, aby někdo jiný předávané informace odposlouchával nebo podvrhoval, pomocí šifrování. Čipové karty využívají kryptografii při komunikaci s terminálem, protože kdyby veškerá komunikace nebyla šifrovaná, představovalo by to bezpečnostní slabinu. Útočníkovi by to umožnilo odposlouchat komunikaci, zkopírovat soubory nebo podvrhnout identitu uživatele karty nebo terminálu. V kryptografii jsou náhodná čísla velmi důležitá, jsou potřebná ke generování kryptografických klíčů. Kromě procesu generování klíčů, jsou náhodná čísla využívána také pro kryptografickou sůl, padding a nonce. Co jednotlivé termíny znamenají bude vysvětleno v příštích odstavcích.

Kryptografii lze rozdělit na symetrickou a asymetrickou. V symetrické je používán pro šifrování a dešifrování stejný, nebo od sebe lehce odvoditelný klíč. Asymetrická

kryptografie oproti tomu používá páry klíčů, veřejný a soukromý. Veřejný klíč umožňuje zprávu zašifrovat, ale dešifrování tímto klíčem není možné, k tomuto účelu je potřebný klíč soukromý.

Šifrovací algoritmy jsou známé, bezpečnost šifry stojí na utajení klíčů. Symetrické šifrování používá transpozici a substituci znaků, tedy záměnu pořadí znaků a nahrazení znaku jiným znakem. Asymetrické šifrování využívá různé matematické problémy jako faktorizace čísel, výpočet diskrétního logaritmu a eliptické křivky. Tyto matematické problémy fungují na takovém principu, že i když je snadné z výchozích hodnot vypočítat výslednou hodnoty, z výsledné hodnoty není možné získat vstupní hodnoty.

Symetrické proudové šifry jako klíč používají náhodné nebo pseudonáhodné číslo, jehož bity jsou generovány na základě inicializačního klíče. Bezpečnost Vernamovy šifry, symetrické proudové šifry, o které bylo matematicky dokázané, že je neprolomitelná, spočívá ve vygenerování kvalitního náhodného klíče, stejně dlouhého jako zpráva, která má být zašifrována. Tento klíč může být použit pouze jinou a generovaná čísla musí mýt vysokou entropii, při nesplnění těchto podmínek hrozí rozšifrování zprávy.

Pro příklad asymetrického šifrování, šifra RSA využívá problém faktorizace velkých čísel. Ke stanovení dvojice veřejný a soukromý klíč je zapotřebí vygenerování dvou velkých prvočísel. Šifrování se provádí veřejným klíčem, dešifrování klíčem soukromým. Pro generování klíčů je potřebný kryptografický generátor náhodných čísel. Požadavky na tyto generátory jsou popsány v podkapitole 2.3. Více o principech kryptografie a generování klíčů zde [16], [17] a [18].

Sůl, anglicky salt, je náhodné číslo, které se někdy používá při generování hashe. Hash je výsledek jednosměrné funkce, která jakýkoliv vstup, obvykle heslo, přemění v posloupnost přesně stanovené délky. Různé vstupy by vždy měly vést k rozdílným hashům a výpočet vstupu z hashe by měl být nemožný. Podle těchto kritérií se hodnotí, jestli hashovací funkce je bezpečná. Kolize, tedy zjištění vstupu z hashe nebo nalezení dvou vstupů se stejnými hashy, se vždy vyskytují, porovnává se ale, jak obtížné je jejich objevení. Hashování hesel se provádí po zadání hesla do přihlašovacího formuláře. Hashovací funkce se používá například pro to, aby útočník při odposlouchávání komunikace nemohl nezachytil heslo. Také kvůli bezpečnosti by se v databázích s uživatelskými údaji neměly uvádět hesla, pouze jejich hashe.

Pro vyšší bezpečnost se někdy při hashování používá sůl. Sůl je náhodné číslo, je to další vstup pro hashovací funkci. V praxi ale část hash funkcí přijímá jenom jeden vstup, v tomto případě se jednoduše přičte sůl k heslu. Pokud je k uživatelským heslům přičítán hash, je kromě hashe soli a hesla uložena i samotná sůl. Pro různé uživatele by měla být použitá jiná hodnota soli, tím je zajištěno, že i když uživatelé nudou mít shodná hesla, jejich hashe nebudou stejné. Pokud útočník zjistí hash a chce zjistit heslo, sůl poskytuje určitou ochranu proti slovníkovým útokům. Protože hashovací funkce jsou dostupné, může si ze slovníků častých hesel vypočítat slovníky jejich hashů. Při použití soli ale musí

slovníky hashů počítat pro každého uživatele zvlášť, to zabraňuje útočit na více uživatelských účtů zároveň a výrazně zpomaluje útočníka.

Padding znamená přidávání bitů ke zprávě, před jejím zašifrováním. Padding je používán například v případě, že délka zprávy, kterou je potřeba zašifrovat blokovou šifrou, není dělitelná délkou bloku. Tedy v případě, že po rozdělení zprávy na bloky, by jeden blok nebyl dostatečně dlouhý. Pro správné fungování blokového schématu šifrování by stačilo tento neúplný blok doplnit chybějícím počtem nul. To by ale snížilo bezpečnost, protože v případě zachycení šifrovaného textu, útočník by mohl předpokládat část bitů před zašifrováním. Proto se při doplnění do délky bloku používá vždy jiná náhodná hodnota padding. Padding je možný využít ve všech blokových šifrách, které využívají nějakou inicializační hodnotu.

Nonce se přidává ke zprávě před šifrování, je to hodnota, která zajišťuje, že i když je dvakrát šifrován stejný text stejným klíčem, výsledná šifrovaná zpráva nebude stejná. Z tohoto důvodu stejné číslo nonce nemůže být použito vícekrát. Tento požadavek je i v jeho názvu nonce (number used only once), tedy číslo použité jen jednou. Nonce je vždy předáno tak, aby při dešifrování mohlo být odečteno ze zprávy. Nonce nutně nemusí být náhodné. Více o kryptografické soli, paddingu a číslu nonce zde [16] a [17].

2.3 Požadavky na kryptograficky bezpečné generátory

Protože kryptografie je závislá na kvalitních náhodných číslech, generátory náhodných čísel pro kryptografické účely musí splňovat určité podmínky. Ne všechny generátory tyto podmínky splňují, existují proto různé standardy pro kryptografické generátory, které kladou požadavky pro bezpečnost generátorů a nutnost testování a také vedou seznam schválených generátorů.

Jednou z vlastností, kterou mají kryptografické generátory splnit je, že není možné předvídat další bit ani při znalosti dosavadní posloupnosti bitů. To znamená, že při znalosti k bitů není možné s přesností více než 0,5 určit bit k+1.

Druhá vlastnost je, že při odhalení vnitřního stavu generátoru není možné zpětně dopočítat vygenerovanou posloupnost. Pokud do generátoru vstupuje další entropie, nemělo by odhalení vnitřního stavu vést ani k předpovězení vnitřního stavu generátoru v příštím kroku generování.

Kryptograficky bezpečné generátory by tyto vlastnosti měli splňovat, u většiny algoritmických generátorů to však platí jen za určitých podmínek. Generátory pseudonáhodných čísel vyžadují náhodné vstupy seeds a kvalita výsledné posloupnosti se odráží právě v kvalitě náhodnosti seeds.

Příkladem kryptograficky bezpečného generování může být Blum-Blum-Shub generátor. Ten pro generování využívá funkce modulo součinu dvou velkých náhodných prvočísel. Dalším příkladem může být hashovací funkce, jejíž popis je uveden v podkapitole 2.2.1, která může sloužit jako funkce bezpečná pro generování v kryptografii, pokud hashované počáteční hodnotě je přičteno v každé iteraci číslo 1.

Počáteční hodnota musí zůstat utajena. Pro generování čísel počítačem je možné využití snímání pohybu myší uživatelem a přesný čas jednotlivých úderů do klávesnice. Tyto data jsou využity jako zdroj entropie a slouží jako vstupy pro hashovací funkci.

Jedním ze standardů bezpečnostních požadavků pro kryptografické moduly je FIPS PUB 140-3, který definuje 4 úrovně bezpečnosti a mimo jiné i vyžadované vlastnosti generátoru jako je nezávislost na okolí a způsob implementace. Více informací o tomto standardu zde [19].

Dalším standardem je PKCS (Public Key Cryptography Standards), je to balíček standardů PKCS#1 až PKCS#15 pro kryptografii s veřejným klíčem. Tuto sadu standardů vydává společnost RSA Security. Standardy definují základní matematické vlastnosti veřejných a soukromých klíčů, funkce pro šifrování a podepisování a bezpečné kryptografické systémy. Další informace o standardech PKCS zde [20]. Další informace o generátorech náhodných čísel v kryptografii zde [14], [16], [17] a [18].

2.4 Testy náhodnosti

Pro ověření kvality generátoru je nutné testovat vygenerované posloupnosti. Pro testování je definována řada testů, některé z nich jsou popsány v této podkapitole. Jednotlivé testy jsou často zařazeny do sad testů, díky tomu je možné otestovat více vlastností generátoru. Popisem čtyři z těchto sad testů, Diehard, Dieharder, NIST Statistical Test Suite a TestU01, se věnují následující podkapitoly. Tyto sady testů jsou v závěru kapitoly porovnány v tabulce.

Statistické testy zkoumají, jestli testovaná posloupnost má ty vlastnosti, které by skutečně náhodná posloupnost vykazovala. Pokud posloupnost neuspěje v jakémkoliv statistickém testu, je generátor označen za nespolehlivý, nebo je dále testován. Pokud posloupnost uspěje, výsledek testu ukazuje, že generátor vygeneroval posloupnost, která má určité charakteristiky skutečně náhodné posloupnosti.

Testování vychází ze tří principů:

- **Rovnoměrné rozložení** – Vygenerování nuly a jedničky musí mít stejnou pravděpodobnost, tedy pravděpodobnost 0,5. To znamená, že nuly tvoří polovinu délky posloupnosti.
- **Škálovatelnost** – Pokud posloupnost úspěšně prošla testem náhodnosti, náhodně vybraná část posloupnosti musí být také náhodná a projít stejným testem.
- **Konzistence** – PRNG by měly být testovány pro více výchozích parametrů seeds. Na základě posloupnosti vygenerované z jedné hodnoty seed není možné posoudit kvalitu generátoru.

U algoritmických generátorů se také musí zkoumat postupy, kterými jsou čísla generovaná, aby bylo ověřeno, že není chybný. To se provádí pomocí teorie čísel a matematickou analýzou metody generování daného generátoru.

Při testování je zvolena nulová hypotéza, že testovaná posloupnost je náhodná. Pokud je hypotéza vyvrácena, generátor, který tuto posloupnost vygeneroval, není vyhovující. Pokud hypotéza není vyvrácena, není to důkaz, že posloupnost je náhodná, pouze se zvyšuje důvěra v tuto posloupnost a generátor. Předpokládá se, že testování nemuselo odhalit chyby nebo slabá místa generátoru. Náhodnost posloupnosti je popisována pravděpodobnostními hodnotami. Výsledek testu je *p*-hodnota, vyjadřuje sílu důkazů pro potvrzení nulové hypotézy. Jestliže je *p*-hodnota rovna jedné, vygenerovaná posloupnost je dokonale náhodná. Jestliže je rovná nule, posloupnost je dokonale nenáhodná.

P-hodnota je porovnávána s hladinou významnosti α. Pokud je *p*-hodnota vyšší nebo rovna α, je nulová hypotéza přijata. Pokud je *p*-hodnota menší, je posloupnost zamítnuta.

Hladina významnosti α je pravděpodobnost milného zamítnutí nulové hypotézy. Tento problém je označován jako chyba typu I. Pokud je α vysoká hodnota, je možné, že bude zamítnut generátor skutečně náhodných čísel. Pokud by naopak hodnota α byla příliš nízká, v testu by mohli uspět i nenáhodné posloupnosti. To je označováno za chybu typu II. Hlavním cílem je zvolit hladinu významnosti α tak, aby nedocházelo k chybě typu II. Hodnota α je volena v rozmezí od 0,001 do 0,01.

Nyní budou krátce představeny některé statistické testy. Tyto testy jsou často zahrnuty v sadách testů, kterým se věnují příští podkapitoly.

- **Frekvenční test** – Cílem tohoto testu je určit, jestli jsou jedničky a nuly v posloupnosti zastoupeny přibližně tak, jak by se očekávalo od náhodné posloupnosti. Sleduje se tedy rovnoměrné rozdělení.
- **Sériový test** – Tento test zkoumá dvojice bitů. Cílem je zjistit, zda všechny možné dvojice bitů jsou zastoupeny tak, jak se očekává v náhodné posloupnosti.
- **Poker test** – V tomto testu je posloupnost rozdělena na stejně dlouhé části, u kterých se testuje zastoupení jednotlivých čísel.
- **Runs test** – Testuje počet a délku řetězců po sobě jdoucích čísel (jedniček nebo nul).
- **Autokorelační tes**t – Cílem testu, je prověřit korelaci mezi posloupností a její posunutou verzí pomocí operace XOR.
- **Spektrální test** – Cílem testu je odhalit pravidelné řetězce, které nejsou blízko u sebe v testované posloupnosti. Využívá se diskrétní Fourierova transformace.

Více o testech náhodnosti zde [14], [15], [17].

2.4.1 Diehard

Sada testů Diehard pro testování náhodnosti čísel obsahuje 15 statistických testů. Diehard vyžaduje jako vstup posloupnosti v 32bitovém formátu. Nejvhodnější velikost posloupností pro testování nástrojem Diehard je 10 až 100 megabytů, testování větších posloupností je pomalé a méně přesné. Výstupem každého testu je výsledná *p*-hodnota.

Nástroj byl původně napsán v jazyku C a Fortran. Sada testů Diehard byla zveřejněná roku 1995 a disk spolu s testy obsahoval i posloupnosti, které byly připravené k testování.

Diehard byl použit jako základ pro novější nástroj Dieharder, této sadě testů je věnována následující podkapitola. Více informací o Diehard sadě testů zde [21] a [22].

2.4.2 Nástroj Dieharder

Dieharder je nástroj, který nyní obsahuje přibližně 30 testů pro testování vlastností a kvality náhodných čísel. Tyto testy rozšiřují sadu testů Diehard a zapracovává i některé testy z NIST Statistical Test Suit, kterým je věnována podkapitola 2.4.3. Dieharder je propracovanější než nástroj Diehard a odstranil několik slabin, které Diehard obsahuje.

Nástroj Dieharder je otevřený software pod GNU Public Licencí. To znamená že je ho možné stáhnout, je možné k nástroji přidat další testy, nebo samotný kód upravovat podle potřeby. Poskytuje jednoduché uživatelské rozhraní.

Dieharder umožňuje testovat nezpracované řetězce bitů, soubory obsahující formátované ASCII znaky nebo čísla v desítkové soustavě. V případě, že testovaná posloupnost je vyhodnocena jako nenáhodná, Dieharder umožňuje prozkoumat, u kterého z testů došlo k selhání a proč.

Dieharder umožňuje testovat spolehlivěji a rychleji delší posloupnosti, než je možné s nástrojem Diehard. Dieharder by měl být schopný testovat vstup až do velikosti 2 gigabytů. Více o Dieharder [23] a [24].

2.4.3 NIST Statistical Test Suite

NIST Statistical Test Suite je soubor statistických testů k ověření náhodných a pseudonáhodných generátorů pro kryptografické využití, který vydala společnost NIST (National Institute of Standards and Technology). NIST STS je nejčastěji používaná sada pro ověření kvality generátorů.

Sada obsahuje 15 statistických testů, které testují posloupnosti bitů generovaných kryptografickými hardwarovými či softwarovými generátory. Testy se soustředí na odhalení různých druhů porušení náhodnosti, které se můžou v posloupnosti objevit.

Některé z 15 zvolených testů v sobě obsahují další menší testy. Více o souboru testů NIST STS zde [15].

2.4.4 TestU01

TestU01 je knihovna implementovaná v jazyku ANSI C. Nabízí 6 sad testů pro statistické testování generátorů náhodných čísel. Knihovna byla poprvé zveřejněná roku 2007. Její autoři jsou Pierre L'Ecuyer a Richard Simard z kanadské Université de Montréal. Knihovna testuje posloupnosti bitů a také rovnoměrné rozdělení čísel v intervalu (0;1).

Ze 6 sad testů jsou 3 určené pro testování posloupností náhodných čísel v intervalu (0;1). Mají název SmallCrush, Crush a BigCrush. Liší se zejména v době běhu testu, počtu použitých náhodných čísel a použitých testech. Výsledky SmallCrush testů jsou obvykle dostupné za několik vteřin, u BigCrush je nutné počkat několik hodin.

Zbylé 3 sady testů jsou pro bitové posloupnosti například generované kryptografickým generátorem. Sady se nazývají Rabbit, Alphabit a BlockAplhabit. Před spuštěním testů je nutné specifikovat, kolik bitů je dostupných pro každý test. Rabbit obsahuje 38 a Alpabit 17 různých statistických testů. BlockAlphabit používá Alphabit sadu testů opakovaně na generátor nebo binární soubor tak, že bity jsou rozděleny do bloků různých velikostí a pořadí těchto bloků je měněno.

Knihovna obsahuje známé statistické testy pro RNG, testy, které byly teoreticky navrhnuté v literatuře, a taktéž několik úplně nových testů.

Testy z knihovny TestU01 je možné testovat knihovnou předdefinované generátory, generátory definované uživatelem, stream čísel ze zařízení i na čísla uložená v souborech. Více o knihovně zde [25].

2.4.5 Srovnání testových sad

Vybrané sady testů, které slouží k testování více druhů nenáhodnosti, které se mohou ve vygenerovaných posloupnostech objevit, jsou popsány v předchozích kapitolách. V tabulce 2.1 jsou tyto sady testů shrnuty. Tabulka obsahuje informace o tom, jakou formou jim můžou být předána čísla, která mají být otestovaná. Zároveň je zde uveden rok prvního publikování, programovací jazyk, ve kterém jsou napsány a počet testů, které obsahují.

Tabulka 2.1 Porovnání sad testů

Název	**Počet testů**	**Rok vydání**	**Jazyk**	**Vstup**
Diehard	15	1995	**Fortran, C**	Binární soubor
Dieharder	31	2003	**C**	Binární soubor, soubor s ASCII znaky, standartní výstup programu
NIST STS	15	2010	**C**	Binární soubor, soubor s ASCII znaky 0 a 1
TestU01	95, 106 *(Crush, BigCrush)* 17, 38 *(Alphabit, Rabbit)*	2007	**C**	Binární soubor, soubor s ASCII znaky 0 a 1, standartní výstup programu

2.5 Možnosti generování náhodných čísel na čipových kartách

Pro bezpečnou komunikaci s terminálem čipové karty potřebují kryptografii. K tomu, aby komunikace mohla být šifrovaná, je nutné, aby čipové karty generovaly čísla s vysokou náhodností. Čipové karty mohou generovat čísla náhodná nebo pseudonáhodná, v závislosti na tom, jestli jejich hardware obsahuje generátor náhodných čísel nebo jsou čísla generována z výchozí hodnoty seed, která je vložena do algoritmického generátoru. V následujících podkapitolách jsou blíže popsány specifikace platforem a generování čísel na nich.

2.5.1 Náhodná čísla na JavaCard

Karty s JavaCard technologií nevyrábí vývojáři této technologie, proto je náročné souhrnně popsat generátory náhodných čísel na JavaCard kartách. Každý výrobce podporuje jiné kryptografické funkce a jiný způsob generování. Lze ale říct, že pro generování čísel JavaCard využívá třídu RandomData. Tato třída je abstraktní, implementace záleží na výrobci karty. RandomData může být použita pro generování pseudonáhodných i náhodných čísel.

Všechny implementace RandomData třídy musí implementovat všechny abstraktní metody. Mezi tyto metody patří setSeed, metoda, která upravuje vstupní seed generátoru náhodných čísel. Seed bývá dlouhé 48 bitů a je měněno lineární kongruencí. Metoda nextBytes generuje náhodná data, nextByte nahradila od verze 3.0.5 metodu generateData. K tomu, aby metoda nextByte produkovala nedeterministický výstup, vstupní parametry seed musí být náhodné.

Čísla vhodná pro použití v kryptografických klíčích a v číslu nonce jsou generována algoritmem ALG_KEYGENERATION. ALG_PRESEEDED_DRBG je algoritmus pro generování deterministických náhodných bitů. Od verze 3.0.5 JavaCard skutečně náhodná čísla generuje algoritmem ALG_TRNG, který generuje rozdílná čísla i při stejném seed. Před touto verzí byly pro generování použity algoritmy ALG_PSEUDO_RANDOM a ALG_SECURE_RANDOM. Více o generování náhodných čísel na Java Card platformě zde [26] a [27].

2.5.2 Náhodná čísla na MultOS

MultOS specifikace uvádějí, jaké funkce musí nebo můžou být implementovány na dané verzi MultOS systému a které funkce není možné implementovat. Takovýmto vestavěným funkcím se říká primitivní funkce. MultOS ke generování definuje primitivní funkci GetRandomNumber. Tato funkce generuje 8bytové číslo do zásobníku.

GetRandomNumber funkce musí být podporovaná na všech kartách od verze MultOS 4.2. Způsob generování závisí na implementaci, kterou zvolí výrobce. Funkce GetRandomNumber tedy může vracet náhodná čísla z hardwarového generátoru nebo pseudonáhodná z algoritmu a nějaké hodnoty seed.

Primitivní funkce GenerateRandomPrime generuje náhodná prvočísla. Implementování funkce je povinné pro verzi MultOS 4.3.1, volitelné pro MultOS 4.2, Multos 4.3.2 a výše. Pro verzi MultOS 4 tato funkce není dostupná. Poté co je číslo vygenerováno, je považováno za kandidáta na prvočíslo. Pokud úspěšně projde prvočíselným testem, funkce vrátí toto prvočíslo, pokud ne, nový kandidát na prvočíslo je vygenerován. Více o generování náhodných čísel a MultOS specifikaci zde [28].

2.5.3 Náhodná čísla na BasicCard

Generování náhodných čísel na platformě BasicCard zajišťuje funkce Rnd, která vrací 4bytové číslo. Princip generování náhodných čísel se liší pro různé verze BasicCard operačního systému a pro terminál.

BasicCard ve verzi Enhanced generuje pseudonáhodná čísla. Tyto karty nemají hardwarový generátor, proto není generování na základě fyzikálních jevů možné. Každá karta má v paměti unikátní sériové číslo. Při prvním generování je sériové číslo použito jako seed. Poté je toto číslo uloženo do paměti EEPROM. Unikátní sériové číslo zajišťuje, že žádné dvě karty negenerují stejné posloupnosti. Uložení vstupní hodnoty seed do EEPROM znamená, že karta po resetu nebude generovat stejné posloupnosti. Enhanced katy nemají žádný zdroj entropie, pokud je nutné vygenerovat kryptograficky bezpečná náhodná čísla, terminál musí kartě nějaká náhodná data poslat.

Terminál generuje náhodná čísla pomocí vstupního parametru seed. Seed je odvozeno od času ze systémových hodin, datumu a času zpracování procesu. BasicCard umožňuje pro generování zvolit jiné seed, pokud je vyžadovaná vyšší bezpečnost, nebo naopak, pokud je při vývoji žádoucí generovat předvídatelné hodnoty.

Verze karet Professional a MultiApplication mají zabudované hardwarové generátory, a tedy generují opravdu náhodné posloupnosti.

Pro generování kryptografických klíčů na všech verzích je používán program KeyGen.exe. Program vyzve uživatele k zadání náhodného vstupu do terminálu. Vstup z tlačítek terminálu je pak použit do hashovacího algoritmu MD5. BasicCard poskytuje funkce pro testování prvočíselnosti vygenerovaných čísel. Více o generování náhodných čísel a klíčů na BasicCard kartách zde [29].

2.5.4 Srovnání generátorů čipových karet

Všechny tři vybrané platformy, tedy JavaCard, BasicCard a MultOS karty, mohou generovat čísla pomocí hardwarových generátorů i softwarových generátorů. Jaký generátor je použit na kartě záleží na verzi a specifikaci operačního systému a někdy také na výrobci karty.

Tabulka 2.2 Možnosti generování náhodných čísel na platformách JavaCard, MultOS a BasicCard

Technologie	**JavaCard**	**MultOS**	**BasicCard**
Generátor	PRNG nebo TRNG *(dle implementace)*	PRNG nebo TRNG *(dle implementace)*	PRNG *(Enhanced)* TRNG (*Professional, MultiApplication)*

U karet s operačním systémem MultOS a JavaCard záleží na implementaci této technologie na kartu. Specifikace těchto technologií uvádějí, že generátor náhodných

čísel na kartách musí být dostupný, ale způsob generování není stanoven. Specifikace MultOS definují, jaké funkce musí nebo můžou být implementovány na karty určité verze MultOS systému. JavaCard technologie definuje abstraktní třídu, jejíž implementace a implementace jejich metod je na kartu povinné. Způsob implementace ale definován není. Určité verze BasicCard karet obsahují hardwarový generátor, ostatní generují čísla pseudonáhodné. Stručný přehled možného generování na platformách je naznačen v tabulce 2.2.

3. ANALÝZA NÁSTROJŮ PRO TESTOVÁNÍ NÁHODNOSTI

V této kapitole budou popsány testovací nástroje, které byly vyzkoušeny v rámci této bakalářské práce. Jednotlivé podkapitoly rozebírají způsob užití těchto nástrojů, jejich silné stránky a také se zaměřují na možnost jejich spuštění na operačním systému Windows. Podkapitola 3.5 je věnována nástroji, který má za cíl sjednotit vstup, konfiguraci a formu výsledků několika různých testovacích sad. Podkapitola 3.6 je věnována několika implementacím testovací sady NIST STS (viz 2.4.3). Tyto implementace jsou v podkapitolách porovnány a je posouzena jejich vhodnost užití pro tvorbu aplikace, která je popsaná v kapitole 6. Výsledky testů analyzovaných nástrojů popsaných v této kapitole jsou rozebrány v kapitole 6.3.

Převážná většina nástrojů k testování kvality generátorů náhodných čísel je psaná v jazyce C nebo C++. Často přijímaný datový vstup je přímo binární výstup generátoru, soubor s binárními daty, kde každý byte obsahuje 8 bitů z generátoru. Tento formát je dále označován jako raw binary a znamená, že soubor neobsahuje informace o datovém typu ukládaných čísel ani jiné informace. Některé nástroje také přijímají jako vstup soubor s ASCII reprezentací binárních dat, tedy znaky 0 a 1. Protože nástroje přijímají různé soubory, je před zpuštěním testování vhodné ověřit formát souboru s přiloženými testovacími daty, které jsou většinou součástí vybraného nástroje.

Pro zvolení nástroje, který bude použit, je zapotřebí zohlednit jednak spolehlivost nástroje, tedy to, že odhalí, pokud posloupnost vykazuje vlastnosti, které neodpovídají vlastnostem skutečně náhodné posloupnosti. Také je potřebné zohlednit rychlost testování. Pro tuhle práci byl také kladen důraz na jednoduchost instalace nástroje na počítač s operačním systémem Windows, a to proto, že nástrojů a projektů zaměřených na testování RNG na platformě Linux je poměrně velké množství a jejich zpuštění na jiné platformě často vyžaduje doplňkový software (například Cygwin).

3.1 NIST Statistical Test Suit

Pro spuštění byl použit Cygwin. Implementace „Fast NIST STS[1]“ obsahuje i binární soubory pro Windows, což u dřívějších kompilací byl problém, protože ve Windows knihovna math neobsahuje některé funkce, například funkce erf() a erfc(), využívané v NIST STS nástroji. Vstup může být raw binární soubor nebo soubor s ASCII znaky 0 a 1, na jednom řádku nebo i s novým řádkem.

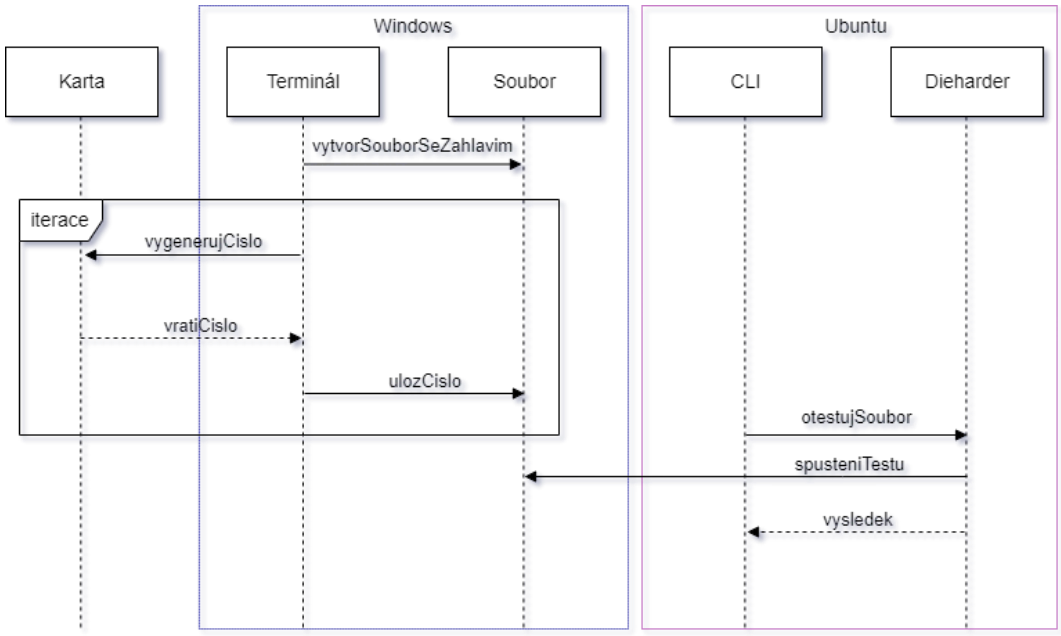
Faster randomness testing, projekt Masarykovy Univerzity, díky kterému tahle verze NIST STS testů vznikla, umožňuje také online testování nahraných souborů. Výsledky těchto testů jsou přehledné a ukládají se. Drobným nedostatkem je ale nedostatek

[1] https://randomness-tests.fi.muni.cz/

informací pro uživatele, který NIST STS nikdy nepoužíval. Před spuštěním testu je nutné vyplnit povinné parametry “length” a “streams”. Chybí ale poznámka, co tyto parametry znamenají a po vložení čísla, které nevyhovuje je zobrazena hláška “Error in Frequency: Length is not in allowed range for this test”, která nenapoví, jaká hodnota by mohla být vložena. Výsledky všech proběhlých testů uživatele jsou ukládány a přehledné. Takováto možnost online testování má samozřejmě to omezení, že výhoda rychlého testování je částečně snížena z důvodu pomalého nahrávání velkých souborů. Online testování a odkaz ke stažení NIST STS testů zde [31].

3.2 Dieharder

Tato sada testů je popsána v kapitole 2.4.2. Nástroj Dieharder[2] byl spuštěn ve virtuálním stroji Ubuntu. Pro testování kvality náhodných čísel bylo kartou BasicCard 7.6 REV D, tedy duální kartou z řady Professional, vygenerováno 18 miliónů čísel. Terminálem byla na hostujícím počítači vytvořena složka a v ní textový soubor, do kterého terminál generovaná čísla průběžně ukládal. Soubor s vygenerovanými čísly byl předán nástroji Dieharder v příkazovém řádku ve virtuálním stroji Ubuntu. Postup testování je zobrazen na obrázku 3.1.



Obrázek 3.1 Schéma testování pomocí ukládání čísel do souboru

Nástroj Dieharder ve verzi 3.31.1 umožňuje testování souborů s řetězci bitů, formátovanými ASCII znaky nebo hodnotami typu float. Protože byla čísla uložena do textového souboru, pro testování byl použit mód pro testování souborů s formátovanými

[2] https://webhome.phy.duke.edu/~rgb/General/dieharder.php

ASCII znaky. Soubor obsahoval záhlaví vyobrazené na obrázku 3.2 a čtyřbytová vygenerovaná čísla, každé na novém řádku.

```
#=============================================================
# generator = BasicCard 7.6 seed = unknown
#=============================================================
type: d
count 18000000
numbit: 32
```

Obrázek 3.2 Záhlaví textového souboru s vygenerovanými čísly

Numbit hodnota udává kolik bitů mají testovaná čísla, *count* udává počet čísel v souboru a volba *typu d* znamená, že jsou čísla v decimálním tvaru.

Dieharder umožňuje nastavit hranici, při které je *p*-hodnota vyhodnocena jako příliš nízká pro potvrzení nulové hypotézy, tedy to, že generátor produkuje náhodná čísla, nebo nízká natolik, že je nulová hypotéza zamítnuta. Vyhodnocení *weak* značí, že pro potvrzení nulové hypotézy byly získány pouze slabé důkazy. *Failed* znamená, že nulová hypotéza byla vyvrácena.

Pro zpuštění testování byl napsán skript, který slouží pro automatické postupné spouštění jednotlivých testů na stejný soubor. Pro každý test byl vytvořen textový soubor s výsledky.

```
#!/bin/bash

for ((i = 0 ; i <= 209 ; i++)); do
        Dieharder -d $i -g 202 -f random18m.txt >>
                        results/resultsTest"$i".txt
        if [ $i == 13 ]
        then i=14
        elif [ $i == 17 ]
        then i=99
        elif [ $i == 102 ]
        then i=200
             dieharder -d $i -n 4 -g 202 -f random18m.txt >>
                             results/resultsTest"$i".txt
        else continue
        fi
done
```

Jednotlivé testy jsou volány parametrem `-d`. Dieharder poskytuje celkem 31 tesů, test *diehard_sums* ale autor nedoporučuje používat a další tři testy jsou označeny za podezřelé. Pro testování byly použity všechny testy, kromě testu *diehard_sums*. Vypsání všech možných testů za sady Dieharder je možné zadáním příkazu `dieharder -l`, příkaz také vypíše číslo testu pomocí kterého je volán.

Výsledky jsou zapsány v tabulce 3.1. Popisy jednotlivých testů jsou dostupné ve zdrojovém kódu nástroje Dieharder [30]. Ze 30 Dieharder testů bylo 7 testů vyhodnoceno jako *passed*. Jeden test byl vyhodnocen jako *weak*. Dalších 21 testů bylo vyhodnoceno

jako *failed*, tedy že hypotéza byla vyvrácena. Zbylý test, který zahrnoval 30 podtestů s různými parametry, byl vyhodnocen pro každý podtest zvlášť, 16krát jako *failed* a 14krát jako *passed*.

Tabulka 3.1 Výsledky jednotlivých testů

Název testu	Ntup	Počet přehrání	Výsledek	Podrobněji
diehard_birthdays	0	1	**Passed**	
diehard_operm5	0	6	**Weak**	
diehard_rank_32x32	0	7	**Failed**	
diehard_rank_6x8	0	3	**Passed**	
diehard_bitstream	0	2	**Failed**	
diehard_opso	0	12	**Passed**	
diehard_oqso	0	8	**Passed**	
diehard_dna	0	4	**Failed**	
diehard_count_1s_str	0	0	**Failed**	
diehard_count_1s_byt	0	7	**Failed**	
diehard_parking_lot	0	0	**Failed**	
diehard_2dsphere	2	0	**Failed**	
diehard_3dsphere	3	0	**Failed**	
diehard_squeeze	0	8	**Failed**	
diehard_runs	0	1	**Passed**	Passed 2 ze 2
diehard_craps	0	9	**Failed**	Failed 2 ze 2
marsaglia_tsang_gcd	0	111	**Failed**	Failed 2 ze 2
sts_monobit	1	1	**Failed**	
sts_runs	2	1	**Failed**	
sts_serial	1-16	1	*****	Failed 16 testů ze 30
rgb_bitdist	4	5	**Failed**	Ntup (0-12)
rgb_minimum_distance	4	2	**Failed**	Ntup (2-5)
rgb_permutations	4	2	**Passed**	Ntup (2-5)
rgb_lagged_sum	4	28	**Failed**	Ntup (0-32)
rgb_kstest_test	0	1	**Failed**	
dab_bytedistrib	0	0	**Failed**	
dab_dct	256	0	**Failed**	
dab_filltree	32	2	**Passed***	Záleží na ntup (0-43)
dab_filltree2	0; 1	0	**Failed**	Failed 2 ze 2
dab_monobit2	12	0	**Failed**	

Při postupném spouštění jednotlivých testů bylo možné z terminálu vyčíst, kolikrát pro daný test musel být soubor přehrán. To umožňuje bližší odhad, které testy byly vyhodnoceny jako failed kvůli malému počtu dat. Pokud soubor musel být přehrán vícekrát pro jeden test, je zřejmé, že například test založený na porovnání výskytu

stejných čísel v testované množině může odhalit nesoulad s nulovou hypotézou, protože může vyhodnotit číslo, které se v souboru vyskytuje pouze jednou za příliš časté.

V tabulce výsledků je pro každý test zapsán přibližný počet přehrání. Toto číslo bylo zaokrouhleno dolů, proto některé testy mají počet přehrání nula.

Čím více dat je poskytnuto nástroji Dieharder tím přesnější jsou výsledky jednotlivých testů. Při nedostatečné velikosti vstupních dat pro test jsou čísla vyhodnocená testem jako nevyhovující nebo slabá. Pro testování byl zvolen počet čísel 18 miliónů. Více vygenerovaných čísel by blíže vypovídalo o kvalitě jejich náhodnosti.

3.3 PractRand

PractRand[3] je primárně nástroj pro generování náhodných čísel pomocí několika algoritmických generátorů náhodných čísel. Poskytuje přehled jejich rychlostí a také umožňuje jejich testování.

Je možné ho použít na vlastní PRNG. Umožňuje testovat velmi dlouhé posloupnosti (tera až exabajty). Ukazuje průběžné výsledky i v průběhu běhu programu. Testování probíhá pomocí více vláknových procesů pro větší rychlost. Obsahuje několik originálních testů, které nejsou zahrnuty v jiných sadách a nástrojích. Je to vhodný nástroj, pokud uživatel potřebuje testovat výstupy nově implementovaných generátorů. Data k testování do nástroje PractRand vstupují ze stdin (a také stdin8, stdin16, stdin32 a stdin64).

Autor uvádí, že nástroj dokáže odhalit bias u více druhů generátorů než většina ostatních nástrojů, ale potřebuje k tomu více bitů.

Neumožňuje spouštět testy na soubor s vygenerovanými daty. To se ale dá obejít tak, že data ze souboru (raw_binary_file.bin) budou přesunuty na standartní výstup:

Pro Windows:

```
type raw_binary_file.bin | RNG_test stdin -tlmin 1KB
```

Pro Linux:

```
cat raw_binary_file.bin | ./RNG_test stdin -tlmin 1KB
```

Pomocí nástroje není možné testovat textový soubor s ASCII znaky nebo ASCII znaky 0 a 1, pouze raw binárních data.

Parametr `-tlmin 1KB` je dobré přidat pro případ, že před koncem běhu testu přestane generátor posílat další data nebo dojdou data z testovaného souboru. I při tomto nastavení bude zobrazen error “error reading standard input”, ale průběžné výsledky budou zobrazeny. Tento nástroj vyžaduje velké množství dat, proto je nevhodný pro testování ze souboru a je vhodný pro při testování nově vytvořených algoritmických generátorů, kdy je možné jejich výstup vkládat jako standartní vstup do nástroje. Více o nástroji PractRand zde [32] a [33].

[3] https://github.com/MartyMacGyver/PractRand

PractRand je možné bez obtíží instalovat a používat i na operačním systému Windows. Pro vyzkoušení nástroje a porovnání s jinými nástroji byl PractRand instalován pomocí programu MinGW a Visual Studio Code s rozšířením pro C a C++.

3.4 RaBiGeTe

Aplikace k testování posloupností uložených v souboru s uzavřeným kódem. RaBiGeTe[4] aplikace obsahuje GUI s grafickým zobrazením *p*-hodnot. Všechny parametry je možné konfigurovat. V nástroji je použito několik testů ze sady NIST STS.

Nástroj je pouze pro Windows. Podporuje více vláknové procesy, množství vláken je odvozeno, od toho, kolik testů je zvoleno pro spuštění. V rámci této práce bylo odzkoušeno, že vstupem může být pouze raw binary ne ASCII znaky. Více o RaBiGeTe aplikaci zde [34].

3.5 Randomness Testing Toolkit

Nástroj Randomnes Testing Toolkit[5] je pouze pro Linux. Sjednocuje vstupy pro sady NIST STS, Dieharder a TestU01. K nástroji je poskytnutá velmi dobrá dokumentace a možnost konfigurace. Konfigurační soubor musí být upraven podle velikosti testovaného souboru. Jsou přiloženy soubory s doporučené nastavením pro různé délky jako vzor.

Tento nástroj byl vyzkoušen na operačním systému Ubuntu. Po jeho instalaci a upravení konfiguračního souboru, kde bylo zapotřebí přidat cestu k instalovaným sadám testů bylo možné spouštět testy ze sady NIST STS, Dieharder i různé sady testů TestU01, předáváním parametrů v příkazovém řádku. V příkazu musí být také zadán konfigurační soubor, jehož nastavení záleží na délce testovaných dat. Tímto způsobem bylo možné otestovat soubory s náhodnými čísly, která byly kvůli správnému vyhodnocování přeloženy do raw binary formátu. Více o tomto nástroji zde [35] a [36].

Podobný cíl má projekt Random-quality[6], který poskytuje nix-shell pro jednodušší použití Dieharder, TestU01 a PractRand. Více o Random-quality zde [37].

Oba tyto nástroje ale nejsou určeny pro Windows, a proto nebyly využity v této práci pro tvorbu aplikace.

3.6 Implementace NIST Statistical Test Suite v jazyce Python

Knihovna NIST Statistical Test Suite, která je psaná v jazyce C, jako většina ostatních nástrojů pro testování kvality náhodnosti generátorů, byla implementována v jazyce Python několika různými tvůrci. Implementace této knihovny v jiném jazyce je vhodná

[4] http://cristianopi.altervista.org/RaBiGeTe_MT/#download
[5] https://github.com/crocs-muni/randomness-testing-toolkit
[6] https://github.com/tweag/random-quality

kvůli komptabilitě s jinými programy a jednoduššímu spuštění pro uživatele, kteří používají Windows a nevyužívají Cygwin nebo MinGW a Visual Studio s rozšířením pro C a C++.

V této kapitole bude popsáno v čem se jednotlivé implementace liší, jaké vstupy přijímají. Srovnání jejich výsledků při použití stejné posloupnosti bude představeno v kapitole 6.3.5.

3.6.1 Randomness_testsuite

Implementace NIST STS v jazyce Python od Stevna Kho Anga zveřejněná pod licencí MIT. Randomness_testsuite[7] obsahuje grafické uživatelské rozhraní, umožňuje testovat ručně zadanou posloupnost 0 a 1, ale také testuje data souboru, a to buď řetězce ASCII znaků 0 a 1 (v aplikaci nazývané binary data file), nebo textový řetězec.

Aplikace umožňuje spustit některé nebo všechny NIST STS testy a výsledek je zobrazen v GUI u každého testu. Výsledek je také možné exportovat jako textový dokument.

K počítání *p*-hodnoty jsou používané knihovny NumPy a SciPy, které je nutné před spuštěním instalovat.

```
pip install numpy
pip install scipy
```

Knihovna NumPy pak také může být využita ke generování náhodných posloupností. Hladina významnosti pro testy je nastavena na hodnotu 0,01. Na začátku každého testu je v kódu uveden účel daného testu a je také uveden význam přijímaných parametrů funkcí.

Autor jako motivaci k sepsání této aplikace uvádí nekonzistentnost ve výsledcích jiných Python NIST STS implementacích. Více zde [38].

3.6.2 The NIST testsuite in Python

Autorem implementace[8] je Ilja Gerhardt. Aplikace byla nejdřív napsána v jazyce Wolfram (dříve Mathematica) a poté kvůli dlouhému běhu programu také v jazyce Python v roce 2011. Pro běh jsou potřebné knihovny NumPy a SciPy.

Aplikace v příkazovém řádku přijímá jak standartní vstup, soubor s raw binárními daty, soubor s ASCII znaky 0 a 1 a také vstup posloupnosti ASCII 0 a 1 přímo v příkazovém řádku:

```
$ echo "0001001010010111" | ./testrandom.py -t 1
```

Tento příkaz spustí první NIST STS test. Všechny testy je možné spustit jednoduše smazáním parametru `-t 1`.

Uživatel si může zvolit, jestli má být výsledkem *p*-hodnota nebo pouze zda byla posloupnost testem vyhodnocena jako náhodná nebo nenáhodná. Výsledky také mohou

[7] https://github.com/stevenang/randomness_testsuite
[8] https://gerhardt.ch/random.php

být uloženy do souboru pomocí `--outfile`. Parametry pro testování není možné nakonfigurovat. Více o této implementaci zde [39].

3.6.3 Knihovna NistRng

Knihovna[9] vytvořená profesorem Lucou Pasqualini z Univerzity Sieny. Vychází ze Sp800_22_tests, nástroje, kterému je věnována následující podkapitola. Knihovna je zveřejněná s licencí BSD 3. NistRng poskytuje možnost spustit všechny testy z NIST STS baterie nebo pouze vybraný test. Knihovna stejně jako předchozí popsané implementace sady NIST STS používá knihovnu NumPy a SciPy, která je také využitá ke generování posloupnosti, na které může být knihovna odzkoušena. Knihovnu je možné stáhnout příkazem `pip install nistrng`.

Funkcím, které spouštějí testy mají být předány data v polích numpy(ndarray)[10]. NistRng umožňuje převést numpy pole čísel se znaménky na numpy pole jejich 8bitové reprezentace a naopak. Každý test vrací objekt, který obsahuje název testu, float průměr *p*-hodnot, která značí skóre, které posloupnost testem získala a boolean hodnotou, jestli posloupnost byla testem vyhodnocena jako úspěšná. Ta je získána porovnáním *p*-hodnoty s hodnotou hladiny významnosti α, která je nastavená pro všechny testy na hodnotu 0,01, ale uživatel má možnost si repositář s nástrojem stáhnout a hladiny významnosti pro každý test nakonfigurovat. Více o této knihovně zde [40].

3.6.4 Sp800_22_tests

Implementace[11] z roku 2017 od Davida Johnstona. Zveřejněna pod licencí BSD 3, stejně jako předchozí implementace. Taktéž používá NumPy a SciPy, ale také pro některé výpočty implementuje několik vlastních neúplných gama funkcí.

Čísla k testování přijímá ze souboru s raw binárními daty. Testování probíhá pomocí příkazu `./sp800_22_tests.py raw_binary_file.bin`

Do terminálu jsou vypsány *p*-hodnoty pro každý test, posouzení, zda test prošel nebo neprošel a také během výpočtů doplňující informace o výsledcích každého testu. Více o implementaci zde [41].

[9] https://github.com/InsaneMonster/NistRng
[10] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html
[11] https://github.com/dj-on-github/sp800_22_tests

4. ANALÝZA SOUČASNÉHO STAVU TESTOVÁNÍ RNG NA ČIPOVÝCH KARTÁCH

Generátorům náhodných čísel a testování daných generátorů bylo už věnováno několik studií. Převažují studie, kde je implementován na kartu nový TRNG nebo PRNG, který je následně testován. Popis generátorů, které jsou implementovány výrobcem karet, není uveřejněn kvůli konkurenceschopnosti. Proto je možné pouze testovat výstupy těchto generátorů. V této kapitole je představeno několik prací a je popsán přístup, který byl ve studii zvolen k testování generátorů. V závěru je v podkapitole 4.1 shrnuto, jak je nejčastěji přistupováno k testování.

Akram, Markantionos a spol. [42] na čipové karty s JavaCard technologií implementovali 6 generátorů pseudonáhodných čísel, které následně byly testovány sadou NIST Statistical Test Suite, která je popsána v podkapitole 2.4.3. Každý PRNG měl jiný generující algoritmus. Autoři upozorňují, že tyto generátory jsou méně efektivní než generátory implementovány výrobcem, protože nejsou psány v nativním kódu, ale běží na JavaCard Virtual Machine viz 1.3.1. Tento projekt je zde zmíněný, protože zvolený postup testování generátorů umožňuje velmi dobré srovnání jednotlivých generujících algoritmů. Při testování byl každému algoritmu předán stejný soubor s výchozími hodnotami (seeds). Každý generátor vygeneroval 1 048 578 pseudonáhodných 128bitových posloupností. Výstup generátorů byl ukládán do binárních souborů, který byly poté použity jako vstup pro statistické testy.

Ing. Šimka ve své práci [43] zdůrazňuje, že pro důkladné otestování bezpečnosti TRNG, by měl být znám princip, na kterém jsou náhodné čísla generována. TRNG po porušení generuje předvídatelná nebo konstantní čísla a bezpečnost kryptografického systému může být ohrožena. Druh útoku na hardwarový generátor se často odvíjí od způsobu generování. Součástí implementace TRNG jsou často testy, které kontrolují, že TRNG nebyl poškozen. Tyto testy Ing. Šimka rozdělil do 3 kategorií a uvádí, že z důvodu omezených paměťových schopností na čipových kartách tyto testy někdy musí být schopny vyhodnocovat kvalitu náhodných čísel i v případě dostupnosti pouze krátkých posloupností. V práci upozorňuje, že přesto, že je důležité odhalit poruchu generátoru, je nežádoucí, aby čipová karta hlásila falešné poplachy, protože selhání TRNG v některém druhu testu může vést k zablokování dané čipové karty. Práce také upozorňuje, že uspění jednoho TRNG při statistickém testování, neznamená, že všechny TRNG stejného typu budou vykazovat tytéž vlastnosti, a proto by se každá implementace měla testovat zvlášť.

Boorghany a spol. [44] měli za cíl otestovat náhodná čísla a RSA klíče generované tokeny a čipovými kartami. Pro testování náhodných čísel byla použita sada NIST STS. Proběhlo testování dvou různých čipových karet s technologií JavaCard a jednou PKCS #11 čipovou kartou. U všech karet bylo možné upravit vstupní hodnoty generátorů, ale tato možnost nebyla využita, protože autoři nechtěli do zařízení vnášet žádnou externí

entropii. Pro testování generátoru náhodných čísel kaženého tokenu a čipové karty, bylo vygenerováno 10 milionů bitů. Získané výsledky byly porovnávány s výsledky generujícího algoritmu Blum-Blum-Shub. V práci byla hodnocena nejen náhodnost generovaných čísel, ale také průměrná doba generování náhodného bitu. Jedna testovaná čipová karta s JavaCard technologií generovala čísla výrazněji rychleji než druhá, ale jejich náhodnost byla nedostatečná.

Prototyp TRNG, založeném na elektronickém oscilátoru, vytvořený Marco Buccim a spol. [45] byl testován a úspěšně prošel požadavky NIST FIPS. V rámci studie byla testována posloupnost délky 1,6 miliónů bitů a proběhlo několik opakování testování generátoru.

TRNG na principu elektrickém oscilátoru byl také navrhnut Dongshengem Liu a spolem [46]. Byl navržen tak, aby čísla generoval rychle a mohl být použit v čipových kartách. Byl implementován na SD čipovou kartu. Testy ze sady NIST STS byly použity na skupiny 2Mb posloupností před i po jejich upravení. Na ověření hodnoty náhodnosti byla také použita testovací sada Diehard, testovány byly proudy bitů o délce několika miliónů. TRNG byl ve studii také testován při extrémních teplotních a napěťových podmínkách.

Dichtl a Janssen [47] testovali generátor Infineon, který používá kombinaci pseudonáhodných a skutečně náhodných čísel, sadou statistických testů RIPE. Statistické vlastnosti generovaných čísel byly kvůli možnosti útoku na generátor testovány v různých podmínkách, které zahrnovali i zchlazení Infineon kontroléru na teplotu tekutého dusíku. Testovány byla i generována data před jejich úpravou, která slouží k dosáhnutí větší náhodnosti a spočívá ve zkombinování čísel vygenerovaných hardwarovým generátorem s pseudonáhodnými čísly.

Erozan a spol. navrhnuli TRNG [48] pro chytré senzory, čipové karty a nositelné zařízení. Ve jejich práci byl testován NIST Statistical Test Suite. Bylo testováno vždy 1096 bitů, které byly vygenerovány 100 různými TRNG instancemi.

Weigl a Anheier [49] hardwarově do čipových karet implementovali 7 statistický testů pro RNG. Jejich práce zkoumala převážně časové zpoždění, spotřeba energie a plocha potřebná pro jednotlivé testy. Práce měla za cíl zkoumat, jestli je možné a výhodné testovat náhodná čísla přímo kartou. Každý z vybraných 7 testů měl možné hardwarové řešení, ale práce ukázala, že některé testy byly pro použití na čipových kartách vhodnější než jiné. Poker test byl v práci vyhodnocen jako nejnáročnější ve všech 3 měřených parametrech. Přítomnost obvodů pro všechny zkoumané testy na jedné čipové kartě by bylo možné, ale vyžadovalo by přibližně 5-6% plochy procesoru. Spotřeba energie je přijatelná i pro méně výkonné čipové karty a časové zpoždění pro jednotlivé testy je uspokojivé.

Suresh a spol. [50] se také věnovali implementací testů náhodnosti tak, aby paměťovými a energetickými požadavky vyhovovala čipovým kartám. V práci bylo uzpůsobeno 6 testů z testové sady NIST Statistical Test Suite. Uzpůsobené testy testují

příchozí bity sériově a většina nevyžaduje ukládání předchozích bitů a díky tomu je šetřena paměť zařízení. Různé testy sdílejí proměnné kvůli optimalizaci paměti a napájení. Navrhnuté testy umožňují měnění hodnotu hladiny významnosti α pro každý test, podle požadavků kladených na danou platformu.

Nástroj Java Card Algorithm Tester neboli JCAlgTest, vyvinutý CRoCS laboratoři, má za cíl testovat karty s platformou JavaCard a automaticky získávat informace o podporovaných algoritmech, paměťových schopnostech karty a doba běhu algoritmů a operací. Součástí nástroje je applet, který je nutné nahrát na kartu a klientská aplikace JCAlgTestClient, která běží na hostujícím počítači, komunikuje s čipovou kartou s nahraným appletem a ukládá data získaná z karty. Tento nástroj nebyl napsán za účelem testovat kvalitu náhodnosti generátorů, ale testuje dobu, za kterou je karta schopna vygenerovat náhodná čísla. V rámci projektu bylo otestováno velké množství různých karet a porovnány rychlosti generátorů. Více o JCAlgTest zde [51] a srovnání rychlosti generování zde [52].

4.1 Shrnutí analýzy

Z výše uvedených studií je zřejmé, že nejčastější metoda testování nově implementovaných generátorů je pomocí testovací sady NIST Statistical Test Suite. Někdy je použita také kombinace testů NIST STS a Diehard.

Výstup generátoru je ukládán do binárního souboru, který je poté použit jako vstupní data pro testy. V některých případech je přímo testován bitový výstup generátoru. V případě implementování nového TRNG je někdy testován jeho výstup před upravením jeho výstupů pomocí extraktoru (viz 2.1.1).

TRNG by se měly také testovat během celé doby jejich užívání z důvodu možné poruchy nebo úmyslného narušení. Mělo by se testovat, jestli s postupem času neklesá entropie čísel, které jsou generovány. Proto jsou ve studii někdy tyto generátory testovány v různých podmínkách, které mají simulovat útok na generátor. Těmito podmínkami může být vysoká nebo nízká okolní teplota nebo různé napětí.

5. UKLÁDÁNÍ RNG VÝSTUPŮ Z ČIPOVÝCH KARET

K tomu, aby generátory náhodných čísel na čipových kartách mohly být otestovány, jejich výstupy musí být předány systému, ve kterém budou spouštěny testy náhodnosti. V této kapitole bude popsán vývoj aplikací pro ukládání výstupů generátorů z čipových karet do souborů na hostujícím počítači a případné problémy, které se vyskytly při jejich tvorbě.

5.1 BasicCard

V této kapitole bude popsán způsob ukládání dat z generátoru náhodných čísel na BasicCard kartě do souboru. Pro tento účel vznikl v rámci této práce BasicCard projekt RNGintoTxt. Pro jeho spuštění je nejdřív nutné nahrát na čipovou kartu soubor Card.bas zkompilovaný do Card.dbg a poté může být spuštěn program terminálu Term.zct, který data uloží do souboru na hostujícím počítači.

5.1.1 BasicCard Development Environment

BasicCard Development Environment[12] slouží k tvorbě aplikací pro karty i terminál, umožňuje kompilaci zdrojového kódu, nahrání vytvořených souborů na kartu a spuštění terminálového programu, který komunikuje s čipovou kartou.

Software umožňuje nahrávání aplikací na kartu i kompilaci pomocí GUI i příkazového řádku. V příkazovém řádku je kompilace možná provést ZCMBasic, pokud je v příkazu uvedena cesta k potřebným souborům a souboru pro danou verzi karty. Příkaz pro kompilaci souboru Card.bas pro kartu verze 7.6 rev D by vypadal následovně:

```
ZCMBasic -OICard Card.bas -CFPro\ZC76_D.zcf -I"Inc";"Lib"
```

Více o BasicCard Development Environment a kompilaci zde [53].

5.1.2 Aplikace na kartě

Soubor s aplikací, který bude nahrán na kartu, může obsahovat příkaz (command), který může být volán terminálem. V definici tohoto příkazu je uvedeno, jakými hodnotami bytů CLA a INS je volán (viz 1.2). U definice příkazu je nutné specifikovat, jaký typ hodnot příkaz vrací terminálu a zda příkaz přijímá nějaká data. Vytvořenou aplikaci je nutné před nahráním na kartu zkompilovat pro příslušný typ BasicCard karty pomocí BasicCard Development Environment a následně ji pomocí stejného nástroje nahrát na kartu.

Pro tuto práci, byla napsána aplikace Card.zcc, která může být nahrána na BasicCard kartu typu Enhanced, Professional i MultiApplication. Aplikace vrací ID aplikace a po zavolání příkazu generuje náhodná čísla voláním funkce Rnd, která byla popsána v kapitole 2.5.3.

[12] http://basiccard.com/index.html?download.htm

Příkazy na kartě mohou terminálu vrátit data, která karta může uložit do souboru nebo s ním jinak dál pracovat. Karta ale nemůže vracet v parametru pole, proto je v této práci předávání náhodných čísel řešeno řešit předáváním více parametrů, protože je to rychlejší než volat příkaz vícekrát a vracet pouze jeden parament.

Každé z čísel získaných voláním funkce Rnd má délku 4 byty, tedy přesně délku typu long. Kvůli maximální délce APDU odpovědi je nejoptimálnější z karty vracet 64 parametrů typu long. Tento způsob vracení velkého počtu parametrů není elegantní, ale značně to urychluje vracení více náhodných čísel. Dalším možným přístupem by bylo, aby karta zapsala data do souboru, který poté terminál přečte a sám uloží do souboru na hostujícím zařízení.

Některé typy BasicCard karet, například Professional ZC7.5 revision B, neumožňují vracet tak vysoký počet parametrů. Pro tyto karty byla napsána obdobná aplikace Card_shorter_allowed_parameters.zcc, která po zavolání funkce pomocí stejných bytů INS a CLA jako v Card.zcc, navrací terminálu namísto 64 parametrů pouze 30.

Pro karty verze Enhanced byla vytvořena podobná aplikace Card_enhanced.zcc, obsahuje funkci, která vrací 28 parametrů typu long, a neobsahuje některé definice příkazů, kvůli nižší kapacitě paměti RAM na těchto kartách.

5.1.3 Aplikace terminálu BasicCard

Terminálová aplikace v jazyce ZC-Basic slouží ke komunikaci s kartou. Zasílané APDU zprávy mezi terminálem a kartou lze zobrazit v BasicCard Development Environment před spuštěním terminálové aplikace zobrazením okna I/O.

ZC-Basic programovací jazyk umožňuje v terminálu ukládání do binárních souborů. Dokumentace k BasicCard Development Environment [53] uvádí, že binární data jsou ukládána do „ZC-basic specifického formátu“. Testovací nástroje, které testují raw binary soubory, ale potřebují data, ve kterých každých 8 bitů představuje 1 byte dat, tedy žádné informace o formátu ukládaných dat ani jiná doplňující data.

Aby generované posloupnosti byly otestovány přesně, tato možnost využita nebyla, protože je zřejmé, že tento způsob ukládání by mohl vést ke zkreslení výsledků, a i kvalitní posloupnost by mohla být vyhodnocena za málo náhodnou. Z tohoto důvodu byla komunikace s kartou v aplikaci popsané v kapitole 6 řešena pomocí Python modulu pyscard. Tento způsob je více rozebrán u popisu komunikace s JavaCard kartami v kapitole 5.2.3.

Nicméně pro testovací účely v rámci práce byla napsána terminálová aplikace Term, která opakovaně volá příslušný příkaz aplikace nahrané na kartě a ukládá vracená čísla do souboru na hostujícím počítači. Uživatel je v průběhu generování informován o tom, kolik procent čísel bylo vygenerováno a také je vytvořen logovací soubor s údaji o času generování, souborech s vygenerovanými daty a poznámce uživatele k dané kartě.

Vygenerovaná čísla kartou terminál ukládá do textového souboru v decimální podobě, každé vygenerované číslo na nový řádek. Tento textový soubor je poté možný převést,

pomocí vytvořených funkcí v nástroji Translator.py, na soubor s ASCII znaky 0 a 1 nebo raw binární soubor, kde každý byte obsahuje byte vygenerovaných dat

5.2 JavaCard

V této kapitole bude popsáno, jak a pomocí jakých nástrojů je možné tvořit, kompilovat a nahrávat applety na kartu s JavaCard platformou a jakým způsobem lze zasílat APDU příkazy kartě a přijímat data získaná z funkce z nahraného appletu.

5.2.1 Tvorba appletu

Vývojové prostředí NetBeans umožňuje tvorbu appletů a jejich testování pomocí simulovaných Java karet. Uživatel ale musí přizpůsobit JDK použité v NetBeans podle JCDK, to znamená, že často musí být použita výrazně starší verze, jinak applet nebude možné sestavit (pro JCDK 2.2.2 nejvýše verze JDK 8). Přidání nového JDK pro NetBeans je možná v GUI, ale změna defaultního se musí provést v souboru /etc/netbeans.conf.

Apache NetBeans IDE má plnit stejnou funkci jako NetBeans IDE. Ve vývojovém prostředí byl vytvořen zkušební applet a byl sestaven, ale při pokusech nahrát applet na simulovanou kartu zobrazil výjimku, že nebylo možné se připojit ke kartě (`java.net.ConnectException: Connection refused: connect`) a simulované karty nebylo možné ani ručně spustit.

Další vyzkoušený způsob vygenerování .cap souboru, kterým by bylo možné nahrát na čipovou kartu byl proveden pomocí nástroje converter, který je součástí JavaCard Development Kit. Nástroj by měl po správném nastavení proměnných prostředí systému JC_HOME a JAVA_HOME, ze souboru .class vygenerovat soubor .cap. Detailnější popis tohoto nastavení zde [54].

Converter se ale v rámci této práce nepodařilo spustit tak, aby byly vykonány všechny příkazy v converter.bat souboru, protože jeden z příkazů nebylo možné ani manuální spustit v příkazovém řádku. Chyba `'C:\Program' is not recognized as an internal or external command, operable program or batch file`. se zobrazila při pokusu o spuštění jak ve Windows 10, tak ve virtualizovaném Windows 7.

Vygenerování .cap souboru, který bylo možné nahrát na fyzickou kartu, se zdařilo až pomocí Eclipse SDK v kombinaci s pluginy Eclipse JCDE a JCDK (JavaCard Development Kit). Eclipse automaticky provádí kompilaci, tedy generuje .class soubory. Pro úspěšné vytvoření .cap je potřebné nastavit AID appletu i balíčku, ve kterém se applet nachází. V Eclipse SDK je poté možné vytvořit .cap soubor zvolením appletu a vybrání možnosti Java Card a poté Convert.

Vytvoření dalších .cap souborů pro různé verze JavaCard byly vytvořeny kompilátorem v nástroji JCIDE[13].

[13] https://www.javacardos.com/tools

5.2.2 Nahrání appletu na kartu

Vytvořený .cap soubor byl poté nahraný na fyzickou čipovou kartu nástrojem GlobalPlatformPro[14] příkazem:

```
gp -install helloWord.cap -r OMNIKEY
```

kde parametr `-r` specifikuje, která čtečka má být použita. Seznam appletů nahraných na čipovou je možné získat příkazem:

```
gp -l -r OMNIKEY
```

Odstranění appletu a balíčku z karty probíhá podobně pomocí parametru `-delete`, ale je nutné specifikovat AID konkrétního appletu a balíčku, který má být odstraněn Pro usnadnění orientace v nahraných appletech a jejich AID lze použít přepínače `-v` a `-i`. Applet vytvořený pro tuto práci je poté zobrazen jako "xsucho08a" a balíček jako "xsucho08p". Více o GlobalPlatformPro zde [55].

5.2.3 Ukládání do souboru

Komunikace s kartou, na které už je nahrán applet, probíhá pomocí rozšíření programovacího jazyka Python o modul Pyscard, který doplňuje Python o podporu čipových karet. Pomocí Pyscard se s kartou sestavuje spojení, jsou jí posílány APDU příkazy a přijímány APDU odpovědi karty.

Spojení s kartou ve čtečce probíhá pomocí:

```
from smartcard.System import readers

readers = readers()
choosenReader = input("reader:")
connection = reader[int(choosenReader)].createConnection()
```

Nejdříve je nutné kartě sdělit, se kterým appletem má pracovat. To se provádí posláním APDU, které obsahuje byty, které znamenají příkaz „select" a AID daného appletu.

```
select=[0x00,0xA4,0x04,0x00,0x09]
appletAID = [0x78,0x73,0x75,0x63,0x68,0x6f,0x30,0x38,0x61]
selectAPDU= select + appletAID
data, sw1, sw2 = connection.transmit(selectAPDU)
```

Poté co je s kartou navázané spojení a applet je vybrán, mohou být kartě posílány data, která budou zpracovány kartou podle nahraného appletu. Karta zpátky pošle vždy byty SW1 a SW2, které znamenají, zda vše proběhlo v pořádku nebo jestli při zpracovávání obdrženého APDU došlo k nějaké chybě. Pokud karta odpoví byty 0x90 a 0x00, vše proběhlo v pořádku. Ale ani případě, že karta neodpoví předpokládanými byty 0x90 a 0x00, neznamená, že náhodná čísla nebyla vygenerována.

Pokud karta odpoví byty 0x61 a nenulovou hodnotou SW2, znamená to, že by chtěla poslat zpátky další odpověď o délce bytu SW2. Tyto data karta pošle, pokud k tomu obdrží APDU příkaz:

```
GET_RESPONSE_APDU = [0XA0, 0XC0, 00, 00 ] + sw2
```

[14] https://github.com/martinpaljak/GlobalPlatformPro

Ve vytvořeném Python kódu je implementováno zasílání příkazu pro odeslání dodatečných dat karty. Tento mechanismus se aktivuje v situaci, kdy karta odpoví na dotaz SW1 bytem 0x61.

Karta na základě obdrženého příkazu vykonává funkci appletu a zpětně posílá výstup generátoru. Tento výstup se skládá z bytů, které jsou následně ukládány v binární podobě do souboru. Více o modulu Pyscard zde [56].

6. APLIKACE PRO TESTOVÁNÍ KVALITY RNG ČIPOVÝCH KARET

V této kapitole bude popsána vytvořená Python aplikace pro testování výstupů generátorů náhodných čísel čipových karet. Před otestováním karty, je nutné na kartu nahrát zkompilovaný JavaCard applet nebo BasicCard aplikaci, jejíž funkce je aplikací volána. Aplikace slouží k ukládání výstupu karet do souboru a testování kvality posloupnosti uložené v souboru. Aplikaci lze také využít pro převedení souboru s daty v raw binární podobě do souboru s ASCII znaky 0 a 1.

6.1 Popis aplikace

Aplikace má vlastní grafické uživatelské rozhraní, ale je ji možné taky používat jako terminálovou aplikaci. Aplikace využívá upravenou Python implementaci baterie NIST STS Randomness_testsuite.

Randomness_testsuite bylo v aplikaci využito kvůli kvalitě implementace jejich jednotlivých testů. Randomness_testsuite je popsán v kapitole 3.6.1 a dále v kapitole 6.1.3 je popsána její úprava a zdůvodnění těchto změn. Její použití pro tuto práci i její pozměnění je umožněno, protože její kód je uveřejněn pod licencí MIT. Srovnání výsledků s NIST STS nástrojem je provedeno v kapitole 6.3.

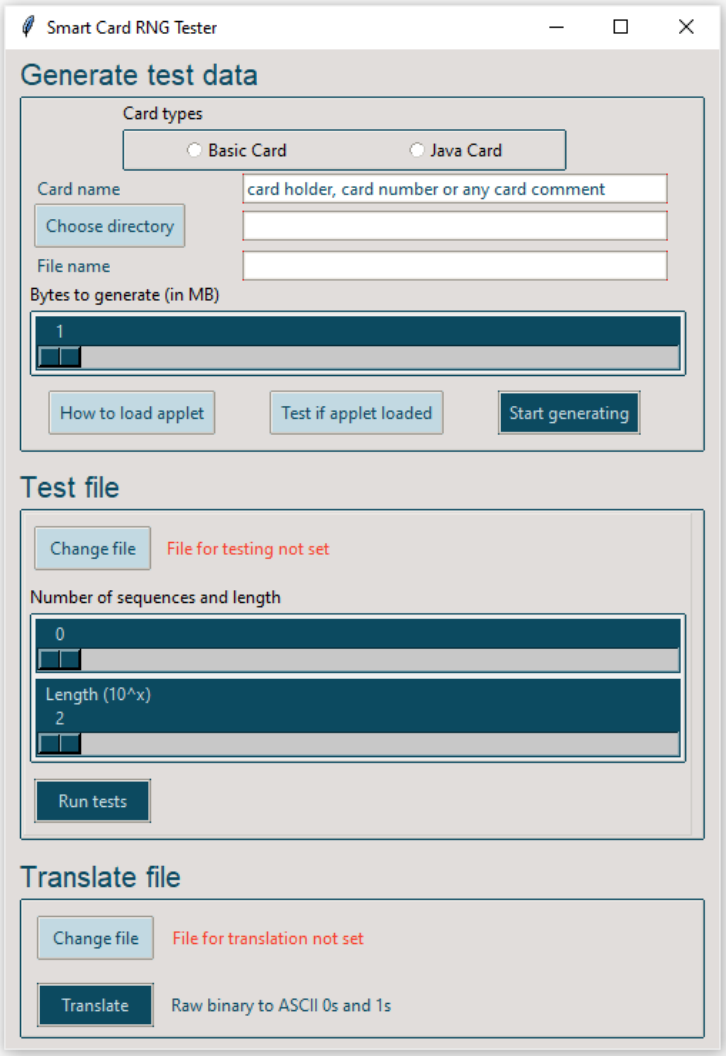
6.1.1 GUI

Pro tvorbu grafického uživatelského rozhraní byl využit Python modul Tkinter. GUI aplikace slouží k testování a získávání dat z karty a také převedení raw binárního souboru do souboru kompatibilního s testováním. GUI obsahuje 3 sekce, které jsou vidět na obrázku 6.1.

První sekce slouží ke specifikování, do jaké složky by měl být vytvořen soubor s daty z RNG karty, komentář ke kartě pro lepší orientaci při testování více karet, specifikaci, kolik dat by mělo být vygenerováno v megabytech za předpokladu, že 1 megabyt je roven 8 miliónům bitům. Před spuštěním je nutné nahrát zkompilované aplikace ke generování na kartu. K popisu, jak to udělat slouží tlačítko „How to load applet“, které zobrazí příkazy, které v příkazovém řádku umožní nahrát aplikaci pro zvolený typ karet a verzi. Nahrávání se neprovádí automaticky v aplikaci, protože při opakování neúspěšných pokusů nahrát aplikaci hrozí zablokování dané karty.

Pokud byl soubor vytvořen a generování skončí, je soubor s uloženými daty nastaven v druhé sekci jako soubor, na který se mají spustit testy. To, na který soubor budou aplikovány testy, je možné změnit tlačítkem „Change file“. V této sekci se dá také nastavit délka posloupností a jejich množství. Délka se nastavuje jako exponent, kvůli lepší ovladatelnosti. Proč testovat více než jednu posloupnost je vysvětleno v kapitole 6.1.3.

Třetí sekce slouží k převedení souboru s daty v raw binární podobě do souboru s daty ve formátu, které aplikace dokáže testovat.

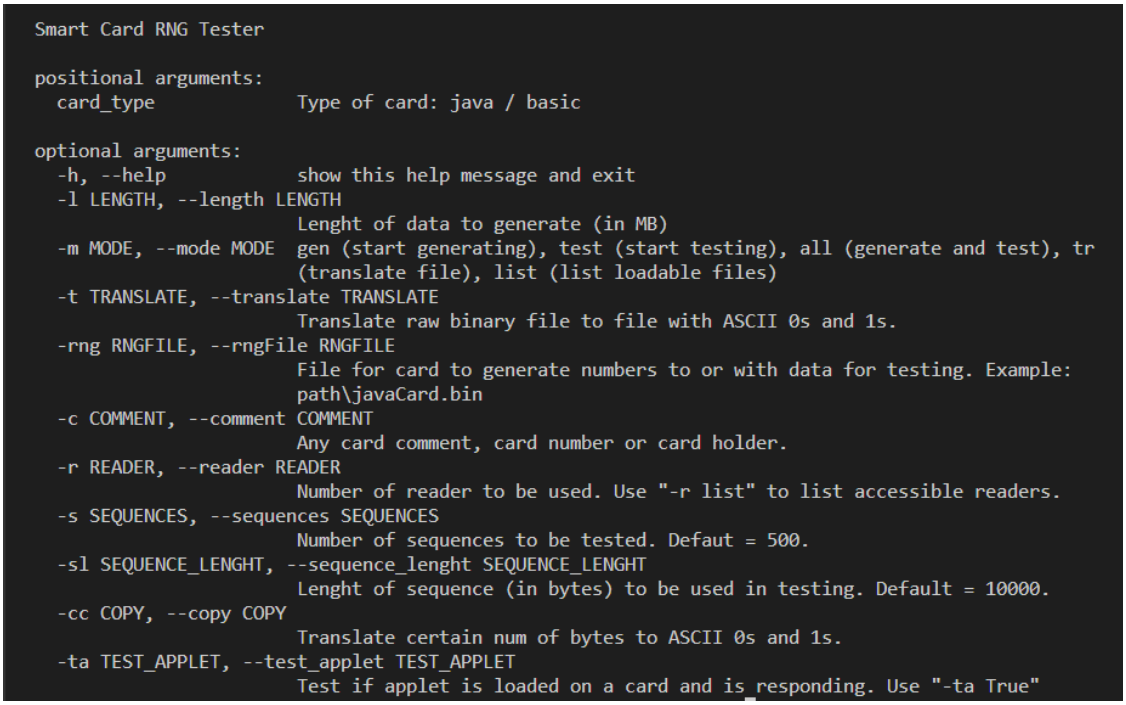


Obrázek 6.1 GUI aplikace pro testování RNG čipových karet

6.1.2 Terminálová aplikace

Terminálová aplikace využívá knihovnu argparse pro příjem parametrů od uživatele. Přijímané parametry lze zjistit pomocí `-help`, výpis lze vidět na obrázku 6.2.

Aplikace má několik módů, mód „gen“ slouží k získávání dat z generátoru na čipové kartě a jejich ukládání do souboru. Název tohoto souboru lze nastavit parametrem `-rng`. Mód „test“ slouží k spuštění všech testů na soubor s posloupností ve formátu ASCII znaků 0 a 1, kde každý byte je na novém řádku. Mód „all“ je pro spuštění sběru dat z karty do souboru a poté spuštění testů na soubor s těmito daty. Pokud je nutné testovat soubor s raw binárními znaky, lze ho přeložit do této podoby pomocí módu „tr“.



Obrázek 6.2 Příkaz help aplikace

Při spouštění módu pro testování je možné specifikovat počet a délku posloupností. Defaultně je počet nastaven na 500 posloupností o délce 10 tisíc bytů.

6.1.3 Testování a interpretace výsledků

Otestováním jedné posloupnosti pomocí testů z NIST STS není možné zjistit, jestli generátor produkuje skutečně náhodná čísla. Výsledkem každého testu nebo podtestu je *p*-hodnota, která znamená pravděpodobnost, že by generátor skutečně náhodných čísel vygeneroval posloupnost méně náhodnou než tu, která byla testována. Z toho je zřejmé, že pravděpodobnost, že posloupnost vygenerovaná skutečně náhodným generátorem bude vyhodnocena jako nenáhodná není nulová. Z toho důvodu dokumentace NIST Statistical Test Suite [57] doporučuje testovat velké množství posloupností pro ověření kvality generátoru.

Proto byla využitá Python implementace Randomness_testsuite upravena tak, aby uživatel mohl navolit, kolik posloupností má být testováno a jak mají být dlouhé. V aplikaci je předaný soubor s daty k testování rozdělen na posloupnosti dané délky a každý test je tedy proveden několikrát.

Po skončení testování jsou vytvořeny dva soubory s výsledky. Jeden soubor obsahuje výsledné *p*-hodnoty jednotlivých testů první posloupnosti. Tato hodnota je porovnána s hladinou významnosti, která je v aplikaci nastavena na 0,01. Pokud je *p*-hodnota větší než hladina významnosti, pak posloupnost testem úspěšně prošla.

Druhý soubor obsahuje pro každý test poměr posloupností, které byly vyhodnoceny jako náhodné. Toto získané číslo se porovnává s intervalem spolehlivosti, jenž se vypočítává vztahem

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}, \tag{6.1}$$

kde m je množství posloupností a $\hat{p}$ je rovno 1 – hladina významnosti [57].

Pokud je poměr posloupností, které testem prošly vyšší než tato minimální přípustná hodnota, tedy spadají do intervalu, je test vyhodnocen jako úspěšný. Informace o minimální přípustném poměru posloupností, které testem prošly, pro daný počet posloupností, je zapsána na konci souboru s výsledy. Na obrázku 6.3 je zobrazena část souboru s výsledky, který je výstupem aplikace.

Toto je jeden ze dvou možných přístupů vyhodnocování výsledků pro několik posloupností doporučený tvůrci NIST STS testů. Pro více informací o způsobu vyhodnocování výsledků pro několik posloupností zde [57].

```
------------------------------------------------------------------------------
RESULTS FOR THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
Test Data File: C:/Users/radka/SmartCardTester/Cards/data/bc86revD_50.bin
Basic Card MultiApp 8.6 rev D with marker
Number of sequences:    500
Sequences length:   100000

Type of Test                                    Proportion          Conclusion
01. Frequency Test (Monobit)                    0.992               Random
02. Frequency Test within a Block               0.992               Random
03. Run Test                                    0.984               Random
04. Longest Run of Ones in a Block              0.992               Random
05. Binary Matrix Rank Test                     0.99                Random
06. Discrete Fourier Transform (Spectral) Test  0.984               Random
07. Non-Overlapping Template Matching Test      0.994               Random
08. Overlapping Template Matching Test          0.99                Random
09. Maurer's Universal Statistical test         0.988               Random
10. Linear Complexity Test                      0.994               Random
11. Serial test:
                                                0.992               Random
                                                0.994               Random
12. Approximate Entropy Test                    0.986               Random
13. Cummulative Sums (Forward) Test             0.988               Random
```

Obrázek 6.3 Ukázka souboru s výsledky poměru úspěšných posloupností

6.2 Potřebný software

V této kapitole jsou popsány knihovny a nástroje potřebné pro běh aplikace. Jde o Python knihovny, které jsou využívány k testování a komunikací s kartou s JavaCard technologií, nástroje určené k nahrávání appletů na kartu.

Knihovny NumPy[15] a SciPy[16] jsou nutné ke spouštění testů ze sady NIST STS, tak jak bylo popsáno v kapitole 3.6.1. Jejich instalace pomocí správce balíčků Pip je možná příkazem

```
python pip install numpy
python pip install scipy
```

Dále ke komunikací s kartami potřebný Python modul Pyscard[17], který Python rozšiřuje o podporu čipových karet. Modul je také možné instalovat pomocí Pip a to:

```
python -m pip install pyscard
```

Pomocí Pyscard je možné s kartou sestavit spojení, vybrat příslušný applet a zasílat APDU příkazy. Modul Pyscard je použitý v aplikaci ve třídě *SmartCard* v souboru *CommunicationWithCard*.

Pro nahrání appletů, které volají generování čísel a jejich zaslání v APDU odpovědi, na kartu, je zapotřebí software, který dokáže poslat a zpravovat applety na dané kartě. Pro BasicCard karty je to BasicCard Development Software[18], jenž byl popsán blíže v kapitole 6.1.1. Tento nástroj kompiluje aplikace a nahrává je na kartu.

Pro karty s JavaCard technologií je k nahrání už dříve zkompilovaných appletů do .cap souboru potřebný nástroj GlobalPlatformPro[19][55]. Jeho použití bylo popsáno v kapitole 5.2.2. Pro získání informací o appletech nahraných na kartě v příkazovém řádku pomocí GlobalPlatformPro je možné použít příkaz `gp -list -v -i` a ověřit si tak, že .cap soubor byl nahrán.

6.3 Porovnání výsledků

Pro ověření správnosti výsledků byly pro stejná data (v raw binárním formátu nebo ASCII řetězci nul a jedniček) spuštěny testy různých nástrojů. Výsledky testování pomocí upravené implementace Randomness_testsuite byly porovnány s výsledky nástrojů NIST STS, Dieharder a TestU01 a jiné Python implementace testové sady NIST STS NistRng. V této kapitole jsou rozebrány a srovnány výsledky pro jednotlivé nástroje.

Porovnání výsledků by nemělo smysl na datech, která jsou skutečně náhodná, protože to by mohlo vést k závěru, že nástroj je spolehlivý, i kdyby nebyl schopný odhalit vlastnosti nenáhodné posloupnosti. Proto byly k testování nástrojů vygenerovány náhodné byty pomocí Python knihovny NumPy. K zanešení nenáhodnosti do dat byly byty při ukládání do souboru pozměněny. Většina bytů byla uložena správně, ale pokud byl byt roven určité hodnotě, potom buď nebyl uložen, byl uložen dvakrát, nebo místo něho byl uložený jiný předem definovaný byt. Funkce createBiasFiles() pro ukládání takto zkreslených bytů z RNG je v souboru aplikace Tools.py.

[15] https://numpy.org/install/
[16] https://scipy.org/install/
[17] https://sourceforge.net/projects/pyscard/
[18] http://basiccard.com/index.html?download.htm
[19] https://github.com/martinpaljak/GlobalPlatformPro

Soubory s výsledky jednotlivých testovacích nástrojů, tohoto souboru, je zahrnut v přílohách práce. Výsledky byly ověřeny i pro jiný počet a délek posloupností.

Pro testování nástrojem NIST byla použita implementace Fast NIST STS verze 6.0.1 založené na NIST STS 2.1.2 (viz 3.1) Testování sadou Dieharder a TestU1 bylo provedeno na virtuálním počítači s operačním systémem Linux nástrojem Randomness Testing Toolkit (viz kapitola 3.5).

6.3.1 Výsledky aplikace

Pro potvrzení výsledků byly na 50 posloupností, dlouhých 100 tisíc bytů, ze souboru spuštěny všechny testy s defaultním nastavením. Pro tuto práci upravená implementace nástroje Randomness_suite vyhodnotila 10 testů ze 16 jako neúspěšné. Pro vyhodnocení, jestli je pro posloupnosti test úspěšný, je použit poměr posloupností, které jim úspěšně prošly (více o tomto vyhodnocování v kapitole 6.1.3).

Testy vyhodnoceny jako úspěšné:

- Binary matrix rank
- Discrete Forier Transform test
- Non Overlaping Template Matching test
- Linear Complexity
- Random Excursions test – všechny podtesty
- Random Excursions Variant test – všechny podtesty

6.3.2 Výsledky nástroje NIST Statistical Test Suite

V této kapitole budou shrnuty výsledky testů souboru ze zkreslenými náhodnými byty pomocí implementace Fast NIST STS.

Pro testování bylo také použito 50 posloupností o délce 100 tisíc bytů. Nástroj byl spuštěn pomocí Cygwin terminálu a příkazu

```
./assess 800000 -file data/numpy/biased50.bin -streams 50 -ascii -fileoutput -defaultpar
```

Kladně byly vyhodnoceny testy:

- Binary matrix rank
- Discrete Forier Transform test
- Linear Complexity

U těchto testů byla stejná proporce posloupností, které byly vyhodnoceny jako úspěšné jako u výsledků aplikace.

Test s různými výsledky pro různé podtesty:

- Non Overlaping Template Matching – neúspěšné 34 ze 148

Tento test byl navrženou aplikací vyhodnocen úspěšně, ale proběhl pouze jeden test, nebyl rozdělen na podtesty.

Nástroj některé testy vůbec neprovede, pokud jiné testy nebudou vyhodnoceny kladně. Testy, které nebyly použity, nebo byly použity jen pro malé procento posloupností, takže jejich výsledky není možné interpretovat:

- Approximate Enthropy
- Runs – použit jen když je úspěšný test Frequency block
- Serial
- Random Exersions
- Random Exercuisions Variant

Pro testy, které byly použity pro všechny posloupnosti, je proporce úspěšnosti testů shodná s výsledky navržené aplikace. Při porovnání těchto výsledků s výsledky aplikace je zřejmé, že aplikace odhalila nenáhodnost dat stejně spolehlivě, jen spustila a vyhodnotila některé testy, které NIST STS nástroj nespustil, kvůli tomu, že posloupnost nesplnila některé předchozí testy.

6.3.3 Výsledky Dieharder a TestU01

Testování sadami Dieharder, TestU01 Rabbit a TestU01 SmallCrush bylo provedeno nástrojem Randomness Testing Toolkit ve virtualizovaném stroji Ubuntu. Nástroj přijímá raw binární soubory, proto už při vytváření souboru s málo náhodnými daty byly byty uloženy také v raw binárním formátu. Pro testování byl použit konfigurační soubor nástroje 50MB.json.

Ze sady Dieharder bylo 7 testů z 21 vyhodnoceno jako úspěšných. Stejně jako při testování v předchozích případech bylo počítáno s hladinou významnosti 0,01. Mezi úspěšné testy patří například *Diehard 32x32 Binary Rank Test*, který stejně jako test *Binary matrix rank* používá matici 32 x 32 a byl vyhodnocen jako úspěšný v navržené aplikaci i NIST STS. Naopak *STS Runs Test* neuspěl, stejně jako u předchozích nástrojů.

Sadou Rabbit z TestU01 byl soubor vyhodnocen jako dostačující 4 testy ze 16. SmallCrush z TestU01 obsahuje 5 testů a soubor byl vyhodnocen jako dostačující pouze jedním, a to opět testem *Matrix Rank*.

6.3.4 Výsledky nástroje NistRng

Tato Python implementace sady NIST STS, která je popsána v kapitole 3.6.3, byla také pro ověření výsledků aplikace poupravena tak, aby každý test provedla několikrát a testovaný soubor rozdělila na posloupnosti o dané délce.

Opět proběhlo testování souboru s nedostatečně náhodnými daty, který byl rozdělen na 50 posloupností s délkou 100 tisíc bytů. Úspěšné byly testy:

- Binary Matrix Rank
- Maurer's Universal Statistical test

Testy *Runs* a *Linear Complexity* nebyly použity, protože tento nástroj před spuštěním testu ověří, jestli by měl být daný test použit nebo ne, na základě délky posloupnosti a délky bloku daného testu.

Tato implementace vyhodnocovala testem *Random Excursion* posloupnosti jako nenáhodné, i když všechny jiné nástroje tyto posloupnosti hodnotily jako náhodné.

6.3.5 Přehled výsledků NIST STS implementací

V této kapitole jsou porovnány v tabulkách získané výsledky dvou Python implementací NIST STS a implementace Fast NIST STS. Testování stejného a nedostatečně náhodného souboru každým z těchto nástrojů probíhalo tak, že byl soubor rozdělen na 50 posloupností o délce 100 tisíc bytů, proto by výsledky všech nástrojů pro jednotlivé testy měly být shodné.

V tabulce 6.1 je zobrazen celkové hodnocení prošel/neprošel pro každý test. Šedá barva v tabulce znamená, že test neproběhl, nebo proběhl pro příliš malé procento posloupností na to, aby mohl být spolehlivě vyhodnocen. Žlutá barva znamená, že pouze část podtestů bylo vyhodnoceno úspěšně. V tomto případě je v dané buňce uvedeno, kolik a z jakého celkového počtu testů bylo vyhodnoceno jako náhodné.

Z tabulky je zřejmé, že nástroj NistRng několik testů vyhodnotil jinak, než nástroj NIST Statistical Test Suite a pro tuto aplikaci upravená implementace Randomness_testsuite. NistRng některé testy nespouštěl a také NIST STS nespustil některé testy pro posloupnosti, které neprošly úspěšně některým z testů.

Tabulka 6.1 Porovnání výsledků pro NIST STS implementace

	Navržená aplikace	**NIST STS**	**NistRng**
Frequency Test (Monobit)	failed	failed	failed
Frequency Test within a Block	failed	failed	failed
Runs	failed	-	-
Longest Run of Ones in a Block	failed	failed	failed
Binary Matrix Rank	passed	passed	passed
Discrete Fourier Transform (Spectral)	passed	passed	failed
Non-Overlapping Template Matching	passed	114/148	failed
Overlapping Template Matching	failed	failed	-
Maurer's Universal Statistical	failed	failed	passed
Linear Complexity	passed	passed	-
Serial	failed	-	failed
Approximate Entropy	failed	-	failed
Cummulative Sums (Forward)	failed	failed	failed
Cummulative Sums (Reverse)	failed	failed	failed
Random Excursions	passed	-	failed
Random Excursions Variant	passed	-	failed

Navržená aplikace, na rozdíl od NistRng, nikdy nevyhodnotila test jako úspěšný v případě, že NIST STS testem posloupnost označila za nenáhodnou. Několik testů bylo označeno aplikací za úspěšné v případě, že test nebyl NIST STS spuštěn. Test *Non-Overlapping Template Matching* byl nástrojem NIST STS spuštěn s několika podtesty, který každý byl vyhodnocen zvlášť, Randomness_testsuite tento test vyhodnocoval jen jednou jako úspěšný.

V tabulce 6.2 je možné porovnat, jaký poměr posloupností bylo vyhodnoceno jako náhodné každým testem pro upravený Randomness_testsuite a NIST STS. Pro testy, které byly NIST STS spuštěny, jsou tyto hodnoty shodné. Barva buněk má stejný význam jako v předchozí tabulce.

NIST STS poměr úspěšných posloupností pro test někdy vypočítal i v případě, že nebylo otestováno dostatečné množství posloupností, ale vždy bylo označeno, že tento výsledek není úspěšný, a proto jsou tyto hodnoty označeny šedou barvou.

Tabulka 6.2 Porovnání podílů úspěšných posloupností

	Navržená aplikace	**NIST STS**
Frequency Test (Monobit)	0,02	0,02
Frequency Test within a Block	0,26	0,26
Runs	0,16	1,0
Longest Run of Ones in a Block	0,0	0,0
Binary Matrix Rank Test	1,0	1,0
Discrete Fourier Transform (Spectral)	0,98	0,98
Non-Overlapping Template Matching	1,0	0,0 - 1,0
Overlapping Template Matching	0,0	0,0
Maurer's Universal Statistical	0,02	0,02
Linear Complexity	1,0	1,0
Serial	0,0	-
Approximate Entropy	0,0	-
Cummulative Sums (Forward)	0,02	0,02
Cummulative Sums (Reverse)	0,02	0,02
Random Excursions	0,96 - 1,0	1,0
Random Excursions Variant	0,96 - 1,0	1,0

7. VÝSLEDKY TESTOVÁNÍ

V rámci této práce bylo vytvořenou aplikací (viz kapitola 6) otestováno 7 čipových karet. Každou kartou byla aplikací do souboru vygenerována posloupnost o délce 50 miliónů bytů. Tyto soubory poté byly pomocí aplikací testovány, a to s různými nastavením počtu a délkou posloupnosti.
Každý soubor byl testován 3krát. Jednou při nastavení délky posloupnosti na 50 tisíc bytů při 1 tisíc testovaných posloupností, podruhé při dvakrát tak dlouhých posloupnostech, ale polovině počtu posloupností a potřetí bylo otestováno 50 posloupností o délce 1 miliónu bytů.

V následujících podkapitolách 7.1 a 7.2 jsou diskutovány výsledky pro jednotlivé karty, v kapitolách jsou výsledky pro daný typ karet shrnuty do tabulky. V těchto tabulkách zelená buňka znamená, že test byl celkově vyhodnocen jako úspěšný, červená jako neúspěšný a žlutá značí, že nějaký podtest v tomto testu byl vyhodnocen jako neúspěšný. V buňce je v tomto případě zapsáno kolik podtestů bylo v testu vyhodnoceno jako neúspěšných. Celkový počet obsažených podtestů je zapsán v závorce u názvu testu. Pokud poměr úspěšných posloupností v testu byl velice nízký, je odpovídající buňka označena sytě červenou. Bledě červenou jsou označeny testy, které byly vyhodnoceny jako neúspěšné, ale pouze o několik setin.

V kapitole 7.3 budou představeny testy, které byly některou z karet vyhodnoceny jako nenáhodné, jaký je účel tohoto testu a je přiblíženo, co vyhodnocení tohoto testu za neúspěšný může znamenat pro generátor, který danou posloupnost vygeneroval.

Kromě souboru s 50 milióny byty byly také vytvořeny soubory s kratší vygenerovanou posloupností, a to s 1, 4 a 10 milióny bytů pro každou kartu. Vytvoření těchto souborů proběhlo z důvodu ověření, že byty v hlavním testovaném souboru nevykazují jiné vlastnosti než jiné posloupnosti generované danou kartou.

Generování kartami JavaCard bylo výrazně pomalejší než generování stejného množství bytů kartami BasicCard. Důvodem může být způsob komunikace s terminálem, způsob šifrování zpráv nebo implementace aplikací, které na kartách volají generování náhodných čísel, nebo samotná délka generování náhodných bytů. Pro porovnání délky komunikace bylo posláno kartám 500 testovacích příkazů, na která karta měla odpovědět pouze několika byty a nijak je nezpracovávat. BasicCard kartám toto trvalo maximálně 2,1 sekundy, některým JavaCard kartám až 18 sekund.

7.1 BasicCard

Otestovány byly celkem 4 karty s platformou BasicCard. Tři z nich byly z řady Professional a jedna MultiApplication. Dvě byly úplně stejné, verze 7.6 rev D, další byla verze 7.5 rev B, tato karta byla duální, měla jak bezkontaktní, tak kontaktní rozhraní. Multiaplikační karta byla verze 8.6 rev D. Všechny tyto karty by měly mít hardwarový

generátor. Výsledky karet a případný počet neúspěšných podtestů v jednom testu je zapsaný v tabulce 7.1.

Multiaplikační karta měla pro každé nastavení délky a množství posloupností některý test nebo podtest označený jako neúspěšný. Jednou šlo o test *Runs* a jeden podtest testu *Serial*. Pro jiné délky byl vyhodnocen jako nespěšný některý podtest z testu *Random excursions*. U testu *Runs,* při délce posloupnosti 1 milión bytů, bylo úspěšných pouze 90 % posloupností.

Jedna z karet verze 7.6 rev D byla při testování vyhodnocena v *Runs* vyhodnocena dvakrát jako neúspěšná. Při délce posloupností 1 milión bytů byla úspěšně vyhodnoceno v tomto testu pouze 60 % posloupností. Ostatními testy až na výjimku 3 podtestů testu *Random excursions* pro jinou délku posloupností, byly vyhodnoceny úspěšně.

Druhá karta stejné verze měla úspěšné všechny testy pro všechny nastavení parametrů testování až na test *Overlapping Template Matching*, který pro nejkratší délku posloupností byl úspěšný v 98 % případů, což už je vyhodnoceno jako neúspěch.

Karta Professional 7.5 rev B byla při délce posloupnosti 1 milión bytů vyhodnocena jako náhodná testem *Runs* pouze u 52 % posloupností, při opakování testování jiných posloupností vygenerovaných touto kartou byly výsledky podobné. Při délce posloupnosti 100 tisíc bytů, byl podíl úspěšných posloupností u tohoto testu 96,4 %, to už je pro počet 500 posloupností vyhodnoceno, jako neúspěšné. Kromě tohoto testu byla posloupnost označena jako nenáhodná dohromady pro všechny nastavení 3 podtesty *Random Excursions* a 2 *Random Excursions Variant* podtesty.

Tabulka 7.1 Výsledky pro BasicCard karty a různá nastavení délky posloupnosti

	BC 8.6D			**BC 7.6D - 1**			**BC 7.6D - 2**			**BC 7.5B**		
	1M	100k	50k	1M	100k	50k	1M	100k	50k	1M	100k	50k
Frequency (Monobit)												
Frequency within a Block												
Runs												
Longest Run of Ones in a Block												
Binary Matrix Rank												
Discrete Fourier Transform (Spectral)												
Non-Overlapping Template Matching												
Overlapping Template Matching												
Maurer's Universal Statistical												
Linear Complexity												
Serial (2)	1											
Approximate Entropy												
Cummulative Sums (Forward)												
Cummulative Sums (Reverse)												
Random Excursions (8)		1	2		1	2					1	2
Random Excursions Variant (18)										1		1

7.2 JavaCard

Otestovány byly 3 karty s platformou JavaCard. První dvě podporují JCAPI 2.2.2, třetí karta podporuje verzi API JCAPI 2.2.1. Tabulka 7.2 ukazuje výsledky pro různé délku posloupností a výsledné hodnocení. V případě neúspěšných podtestů je číslo neúspěšných zapsáno v tabulce.

První JavaCard karta, JCOP31, měla test *Overlapping Template Matching* vyhodnocený jako nenáhodný s 97,6 % úspěšných posloupností pro jejich délku 100 tisíc bytů. Také pro kratší délky posloupností byly testy *Random Excursions* a *Random Excursions Variant* vyhodnoceny jako nenáhodné některým podtestem.

Druhá karta, karta J2A080, prošla všemi testy úspěšně, až na jeden podtest testu *Random Excursions* a test *Discrete Fourier Transform* neboli *Spectral* pro délku posloupností 10 tisíc bytů, kdy testem prošlo 97,6 % posloupností.

Třetí karta, NXP JCO31, byla vyhodnocena kladně pro všechny délky a počet testovaných posloupností, kromě 2 podtestů testu *Random Excursions* a 1 podtestu *Random Excursions Variant*, a to pro délky posloupnosti 50 a 100 tisíc bytů.

Tabulka 7.2 Výsledky pro JavaCard karty a různá nastavení délky posloupnosti

	JCOP			J2A080			NXP		
	1M	100k	50k	1M	100k	50k	1M	100k	50k
Frequency (Monobit)									
Frequency within a Block									
Runs									
Longest Run of Ones in a Block									
Binary Matrix Rank									
Discrete Fourier Transform (Spectral)									
Non-Overlapping Template Matching									
Overlapping Template Matching									
Maurer's Universal Statistical									
Linear Complexity									
Seriál (2)									
Approximate Entropy									
Cummulative Sums (Forward)									
Cummulative Sums (Reverse)									
Random Excursions (8)			1			1		2	2
Random Excursions Variant (18)		1	3					1	1

7.3 Vysvětlení testů

V předchozích kapitolách bylo popsáno, které testy byly aplikací vyhodnoceny jako neúspěšné. V této kapitole jsou tyto testy popsány a je zde vysvětleno, kterou vlastnost testují.

Test *Runs* má odhalit, jestli posloupnost neobsahuje jiný počet posloupností délky *k*, které jsou složené z identických bitů, než by se očekávalo od skutečně náhodné posloupnosti. Testuje se pro různé hodnoty *k*. Když je test vyhodnocen jako neúspěšný, může to znamenat, že generátor osciluje mezi bity 0 a 1 příliš rychle nebo moc pomalu.

Účel *Discrete Fourier Transform* neboli *Spectral* testu, je zjistit, jestli posloupnost neobsahuje stejné vlastnosti nebo vzory opakovaně blízko u sebe. To se provádí výpočtem vrcholů Diskrétní Fourierově transformaci posloupnosti.

Overlapping Template Matching testuje, jestli se předem definovaný string vyskytuje v posloupnosti tolikrát, jak by se čekalo od náhodné posloupnosti.

Serial test zkoumá počet zastoupení všech možných posloupností o délce *m* bitů v testované posloupnosti. Každá *m*-bitů dlouhá posloupnost musí být zastoupena přibližně stejně často. V Randomness_testsuite je nastavená délka *m* na 16 bitů. Jednotlivé posloupnosti se můžou překrývat. Test obsahuje 2 podtesty, jeden pro délku posloupností *m* a druhý pro $m - 1$.

Random Excursions se skládá z 8 podtestů, pro různé stavy jeden, *Random Excursions Variant* má stavů, a tedy i podtestů 18. Cílem první skupiny testů je zjistit počet cyklů, které měli přesně *K* výskytů určitého stavu při průběžném součtu jednorozměrné náhodné procházky (random walk). Cílem druhé skupiny testů je zjistit, kolikrát nastal určitý stav a porovnat toto číslo s očekávanou hodnotou. Více o těchto testech zde [15] a [38].

7.4 Porovnání karet

Každá otestovaná karta vygenerovala posloupnost, kterou minimálně jeden test vyhodnotil jako nenáhodnou. V této kapitole budou 2 platformy testovaných karet subjektivně porovnány.

Jen jedna BasicCard karta ze 4 prošla testem *Runs*, ostatní měly pro posloupnosti délky 1 milión bytů velice nízké procento posloupností, které byly vyhodnoceny jako náhodné.

I karty JavaCard některými testy neprošly, ale tyto testy měly poměr neúspěšných posloupností pouze těsně pod přijatelnou hodnotou. Tedy jimi vygenerované posloupnosti celkově vykazovaly vlastnosti, které by skutečně náhodná posloupnost nevykazovala, ale ne v takové míře, jako posloupnosti vygenerované většinou testovaných BasicCard karet.

8. ZÁVĚR

V rámci této práce proběhlo seznámení s čipovými kartami, jejich možnostech generování náhodných čísel, postupy a nástroji pro testování kvality generátorů náhodných čísel a rozšíření Python aplikace Randomness_testsuite. Ta byla upravena z toho důvodu, aby bylo možné získat finální výsledek každého testu z testovací baterie NIST Statistical Test Suite při rozdělení souboru s daty k testování na několik posloupností, protože je pravděpodobné, že některá i skutečně náhodná posloupnost bude vyhodnocena testem jako nenáhodná. V práci proběhlo ověření správnosti této úpravy porovnáním výsledku testů tohoto nástroje a výsledku testu implementace NIST Statistical Test Suite při testování stejného souboru.

V práci byla popsána vytvořená Python aplikace pro automatické testování výstupů generátorů náhodných čísel čipových karet, která k testování používá rozšířený nástroj Randomness_testsuite. Aplikace má vlastní grafické uživatelské rozhraní a je ji možné užívat i jako terminálovou aplikaci. V rámci práce byl vytvořen JavaCard applet a BasicCard aplikace pro čipové karty, která zasílá výstup generátoru terminálu. Tyto aplikace byly zkompilované a mohou být nahrány na různé verze těchto karet. Aplikace pro testování RNG čipových karet uživatele instruuje, jak tyto aplikace nahrát na kartu, dále s čipovými kartami a těmito nahranými aplikacemi komunikuje a ukládá výstup generátorů náhodných čísel do souboru, který je poté otestovaný.

Výsledkem práce je 7 otestovaných programovatelných čipových karet, kdy u některých karet s platformou BasicCard je pozorovatelná vlastnost generátoru oscilovat mezi bity 0 a 1 s jinou rychlostí, než by se očekávalo od generátoru skutečně náhodných čísel. Tato vlastnost ale nebyla pozorována u všech karet, i když měly identickou verzi platformy a stejný generátor, tudíž by se dalo předpokládat, že tato vlastnost může vzniknout po až po čase používání karty nebo záleží na samotné implementaci stejného hardwarového generátoru na kartu. Testované JavaCard karty tuto vlastnost neprojevovaly, ale i ony měly u některých testů vyhodnoceno jako nenáhodné více procent posloupností, než by bylo statisticky očekávané od generátorů skutečně náhodných čísel.

LITERATURA

[1] RANKL, Wolfgang. *Smart card applications: design models for using and programming smart cards*. 4rd edition. Přeložil Kenneth COX. Chichester: John Wiley, 2007, 1-50,. ISBN 9780470058824.

[2] DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. *Velký průvodce infrastrukturou PKI a technologií elektronického podpisu*. 2. aktualizované vydání. Brno: Computer Press, 2009. 38-48 ISBN 978-80-251-2619-6.

[3] ISO7816 Standard Overview. *Smart Card Supply - Your Source for ID Badge and Smart Card Supplies - Evolis, Ediguard and Persona Printers* [online]. Dostupné z: http://www.smartcardsupply.com/Content/Cards/7816standard.htm

[4] RANKL, Wolfgang, Wolfgang EFFING a Kenneth COX. *Smart Card Handbook*. 4th ed. Přeložil Kenneth COX. Chichester: John Wiley, 2010, 441-444, 499- 521. ISBN 978-0-470-74367-6.

[5] Development Kit User Guide: Java Card Platform Architecture. *Oracle.com* [online]. March 2021 [cit. 2021-10-29]. Dostupné z: https://docs.oracle.com/en/java/javacard/3.1/guide/java-card-3-platform-architecture.html

[6] ORTIZ, Enrique C. An Introduction to Java Card Technology – Part 1. *Oracle.com* [online]. May 29, 2003 [cit. 2021-10-29]. Dostupné z: https://www.oracle.com/java/technologies/java-card/javacard1.html

[7] YAKKUNDI, Ashwin Arvind. *Security Implications of Memory Use on Java Card Platform*. Chennai, 2017. Master's thesis. Masarykova univerzita. Vedoucí práce RNDr. Petr Švenda, Ph.D.

[8] MultOS Technology: Multos SmartCard Technology. *Multos.com* [online]. [cit. 2021-11-01]. Dostupné z: https://multos.com/technology/multos-smartcard-technology/

[9] *MULTOS Developer's Guide* [online]. Berkshire: MULTOS Limited, 2021 [cit. 2021-11-02]. Dostupné z: https://multos.com/wp-content/uploads/2021/06/MDG.pdf

[10] BasicCard. *Basiccard.com* [online]. [cit. 2021-10-30]. Dostupné z: http://basiccard.com/index.html

[11] MILLIER, Brian. BasicCards 101 (Part 1): Program Your First Smartcard. *Circuit Cellar* [online]. **2004** (164) [cit. 2021-10-30]. Dostupné z: http://basiccard.com/circuit_cellar.pdf

[12] FAIRHEAD, Harry. ERNIE - A Random Number Generator. *I Programmer* [online]. 2013 [cit. 2021-11-13]. Dostupné z: https://www.i-programmer.info/history/machines/6317-ernie-a-random-number-generator.html

[13] JÍRA, R. *Generování náhodných čísel pomocí magnetických nanostruktur*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 82 s. Vedoucí Ing. Michal Urbánek, Ph.D. 33-37 [cit. 2021-11-13] Dostupné také z: https://core.ac.uk/download/pdf/30309652.pdf

[14] Náhodná čísla: Prezentace pro předmět Aplikovaná kryptografie. VUT FEKT, 2020/21.

[15] RUKHIN, Andrew, Juan SOTO, James NECHVATAL, et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications: NIST Special Publication 800-22 [online]. Revision 1a. National Institute of Standards and Technology, 1.1 - 1.1.6 [cit. 2021-11-13]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[16] ROSULEK, Mike. *The Joy of Cryptography: Undergraduate textbook* [online]. Draft of January 3, 2021. Corvallis, Oregon: School of Electrical Engineering & Computer Science, 2021 [cit. 2021-11-11]. Dostupné z: https://web.engr.oregonstate.edu/~rosulekm/crypto/crypto.pdf

[17] MENEZES, Alfred J., Paul C.van OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1997, 39-41, 171-187. ISBN 978-0-84-938523-0

[18] FERGUSON, Niels, Bruce SCHNEIER a Tadayoshi KOHNO. *Cryptography Engineering: Design Principles and Practical Applications* [online]. Indianapolis: Wiley, 2010, s. 137-145 [cit. 2021-11-11]. ISBN 978-0-470-47424-2. Dostupné z: https://www.schneier.com/wp-content/uploads/2015/12/fortuna.pdf

[19] FIPS 140-3: Security Requirements for Cryptographic Modules. *NIST: Computer Security Resource Center* [online]. NIST, 2019 [cit. 2021-11-12]. Dostupné z: https://csrc.nist.gov/publications/detail/fips/140/3/final

[20] BHATTACHARYA, Anish. PKCS: Public-Key Cryptography Standards. *Encryption Consulting* [online]. 2021 [cit. 2021-11-12]. Dostupné z: https://www.encryptionconsulting.com/public-key-cryptography-standards/

[21] RAVINDRAN, Vish. The Diehard Tests. *The urban engine* [online]. [cit. 2021-11-13]. Dostupné z: http://theurbanengine.com/blog//the-diehard-tests

[22] The tests - A study of entropy. [online]. Dostupné z: https://sites.google.com/site/astudyofentropy/background-information/the-tests

[23] BROWN, Robert G., Dirk EDDELBUETTEL a David BAUER. Dieharder: A Random Number Test Suite. *Robert G. Brown's General Tools Page* [online]. Durham, NC 27708-0305: Duke University Physics Department, 2021 [cit. 2021-11-13]. Dostupné z: https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[24] dieharder(1) - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/1/dieharder

[25] L'ECUYER, Pierre a Richard SIMARD. TestU01. *ACM Transactions on Mathematical Software* [online]. 2007, 33(4), 1-40 [cit. 2021-11-13]. ISSN 0098-3500. Dostupné z: doi:10.1145/1268776.1268777

[26] *Java Card: Development Kit User Guide* [online]. Version 3.1.0u5. Oracle, 2021 [cit. 2021-11-16]. Dostupné z: https://docs.oracle.com/en/java/javacard/3.1/guide/java-card-development-kit-user-guide.pdf
[27] Class RandomData. *Oracle* [online]. [cit. 2021-11-16]. Dostupné z: https://docs.oracle.com/javacard/3.0.5/api/javacard/security/RandomData.html
[28] *MDRM: MULTOS Developer's Reference Manual* [online]. MAO-DOC-TEC-006 v1.58. Berkshire: MULTOS Limited, 2021, 145-146, 175-176 [cit. 2021-11-16]. Dostupné z: https://multos.com/wp-content/uploads/2021/06/MDRM.pdf
[29] GUILFOYLE, Tony. *The ZeitControl BasicCard Family* [online]. Document version 8.15. Minden Germany: ZeitControl cardsystems, 2012, 48,132-134, 142-143, 183-184, 267 [cit. 2021-11-16]. Dostupné z: http://209.68.36.204/downloads/bc_pdf.zip
[30] Dieharder. *Github* [online]. [cit. 2021-12-04]. Dostupné z: https://github.com/eddelbuettel/dieharder
[31] ŘÍHA, Zděněk a Marek SÝS. *Faster randomness testing* [online]. [cit. 2023-03-16]. Dostupné z: https://randomness-tests.fi.muni.cz/
[32] *PractRand* [online]. [cit. 2023-03-18]. Dostupné z: https://pracrand.sourceforge.net/
[33] PractRand. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/MartyMacGyver/PractRand
[34] *RaBiGeTe MT: Multi-threaded version of RaBiGeTe with GUI (Windows only)* [online], 2011. [cit. 2023-03-18]. Dostupné z: http://cristianopi.altervista.org/RaBiGeTe_MT/
[35] OBRÁTIL, Ľubomír, 2017. *Masaryk University Faculty of Informatics The automated testing of randomness with multiple statistical batteries*. Brno. Dostupné také z: https://is.muni.cz/th/uepbs/thesis. Diplomová práce. Masaryk University Faculty of Informatics. Vedoucí práce RNDr. Petr Švenda, Ph.D.
[36] Randomness Testing Toolkit. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/crocs-muni/randomness-testing-toolkit
[37] Random-quality. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/tweag/random-quality
[38] NIST Randomness Testsuit. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/stevenang/randomness_testsuite
[39] GERHARDT, Ilja. Random Number Testing. *Ilja Gerhardt* [online]. [cit. 2023-03-18]. Dostupné z: https://gerhardt.ch/random.php
[40] PASQUALINI, Luca. NistRng. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/InsaneMonster/NistRng
[41] Sp800_22_tests. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/dj-on-github/sp800_22_tests
[42] AKRAM, Raja Naeem, Konstantinos MARKANTONAKIS a Keith MAYES. Pseudorandom Number Generation in Smart Cards: An Implementation,

Performance and Randomness Analysis. *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2012, 2012, 1-7. ISBN 978-1-4673-0229-6. Dostupné z: doi:10.1109/NTMS.2012.6208760

[43] ŠIMKA, Martin. *Testing of true random number generators used in cryptography*. Košice, Slovakia: Department of Electronics and Multimedia Communications, Technical University of Košice, 2. Dostupné z: doi:10.1.1.117.1949

[44] BOORGHANY, Ahmad, Siavash Bayat SARMADI, Pamian YOUSEFI, Pouneh GORJI a Rasool JALILI. Random data and key generation evaluation of some commercial tokens and smart cards. *2014 11th International ISC Conference on Information Security and Cryptology*. IEEE, 2014, 2014, 49-54. ISBN 978-1-4799-5383-7. Dostupné z: doi:10.1109/ISCISC.2014.6994021

[45] BUCCI, M., L. GERMANI, R. LUZZI, A. TRIFILETTI a M. VARANONUOVO. A high-speed oscillator-based truly random number source for cryptographic applications on a smartcard IC. *IEEE Transactions on Computers*. 2003, **52**(4), 403-409. ISSN 0018-9340. Dostupné z: doi:10.1109/TC.2003.1190581

[46] LIU, Dongsheng, Zilong LIU, Lun LI a Xuecheng ZOU. A Low-Cost Low-Power Ring Oscillator-Based Truly Random Number Generator for Encryption on Smart Cards. *IEEE Transactions on Circuits and Systems II: Express Briefs*. 2016, **63**(6), 608-612. ISSN 1549-7747. Dostupné z: doi:10.1109/TCSII.2016.2530800

[47] DICHTL, Markus a Norbert JANSSEN. A high quality physical random number generator. 2000.

[48] EROZAN, Ahmet Turan, Rajendra BISHNOI, Jasmin AGHASSI-HAGMANN a Mehdi B. TAHOORI. *Inkjet-Printed True Random Number Generator based on Additive Resistor Tuning*. IEEE, 2019, 2019, 1361-1366. ISBN 978-3-9819263-2-3. Dostupné z: doi:10.23919/DATE.2019.8715071

[49] WEIGL, Andrew; ANHEIER, Walter. Hardware comparison of seven random number generator tests for smart cards. *Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*, 2003, 55.

[50] SURESH, Vikram B., Daniele ANTONIOLI a Wayne P. BURLESON. On-chip lightweight implementation of reduced NIST randomness test suite. *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2013, 2013, 93-98. ISBN 978-1-4799-0601-7. Dostupné z: doi:10.1109/HST.2013.6581572

[51] JCAlgTest. *Github* [online]. [cit. 2023-03-18]. Dostupné z: https://github.com/crocs-muni/JCAlgTest

[52] Comparative table. *JCAlgTest* [online]. CRoCS MU [cit. 2023-03-18]. Dostupné z: https://www.fi.muni.cz/~xsvenda/jcalgtest/comparative-table.html

[53] GUILFOYLE, Tony, 2012. *BasicCard: BasicCard Developer Manual* [online]. 8.15. Germany: The ZeitControl BasicCard Family [cit. 2023-03-16]. Dostupné z: http://basiccard.com/index.html?download.htm

[54] Development Kit User Guide: Confirming System Variables. *Oracle* [online]. Oracle [cit. 2023-03-16]. Dostupné z: https://docs.oracle.com/en/java/javacard/3.1/guide/confirming-system-variables-tool.html
[55] GlobalPlatformPro. *Github* [online]. [cit. 2023-03-16]. Dostupné z: https://github.com/martinpaljak/GlobalPlatformPro
[56] Pyscard. *Source Forge* [online]. [cit. 2023-03-23]. Dostupné z: https://sourceforge.net/projects/pyscard/
[57] The Interpretation of Empirical Results, 2010. In: RUKHIN, Andrew, Juan SOTO, James NECHVATAL, et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [online]. Special Publication 800-22 Revision 1a. Lawrence E Bassham III: National Institute of Standards and Technology [cit. 2023-04-09]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

3DES	Triple DES
AES	Advanced Encryption Standard
AID	Application Identifier
ATR	Answer To Reset
API	Application Programming Interface
DES	Data Encryption Standard
DF	Dedicated file
EEPROM	Electrically Erasable Programmable Read-Only Memory
EF	Elementary file
IDEA	International Data Encryption Algorithm
JCAPU	Java Card Application Programming Interface
JCDK	Java Card Development Kit
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
MEL	MultOS Executable Language
MF	Mastery file
NIST	National Institute of Standards and Technology
NIST STS	NIST Statistical Test Suite
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
PRNG	PseudoRandom Number Generator
RAM	Random Access Memory
RBG	Random Bit Generator
RNG	Random Number Generator
ROM	Read Only Memory
RSA	Rivest–Shamir–Adleman
SHA1	Secure Hash Algorithm 1
SHA2	Secure Hash Algorithm 2
SIM	Subscriber Identification Module
TRNG	True Random Number Generator

Symboly:

H	entropie (Sh/symbol)
H_{max}	maximální entropie(Sh/symbol)
p_i	pravděpodobnost prvku i
q	počet prvků

SEZNAM PŘÍLOH

Příloha A - Elektronické přílohy

Elektronické přílohy byly odevzdány do systému VUT ve formátu zip. Zde jsou popsány adresáře, které tento zip soubor obsahuje.

A.1 Aplikace

Složka **SmartCardTester**

V elektronických přílohách je dostupný zdrojový kód vytvořené aplikace a soubory se zkompilovanými aplikacemi pro čipové karty a taktéž zdrojové kódy těchto aplikací.

A.2 Porovnání nástrojů

Složka **toolsComparison**

Nachází se zde výsledky testování ne zcela náhodných dat vytvořenou aplikací, nástrojem NIST STS a NistRng.

A.3 Výsledky testování karet

Složka **cardResults**

Zde jsou uvedeny výsledky každé testované karty pro různá nastavení délky posloupnosti.