VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ENTROPICKÝ GENERÁTOR NÁHODNÝCH ČÍSEL

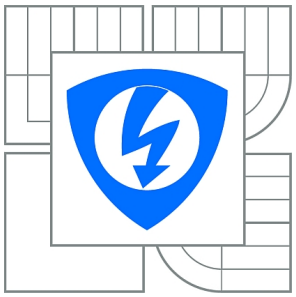
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MICHAL KOLÁŘ
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ENTROPICKÝ GENERÁTOR NÁHODNÝCH ČÍSEL

ENTROPIC GENERATOR OF RANDOM NUMBERS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

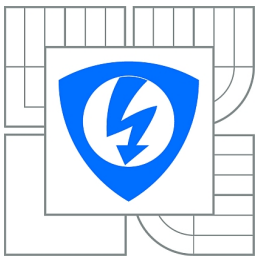
AUTOR PRÁCE — Bc. MICHAL KOLÁŘ

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL BURDA, CSc.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Michal Kolář	***ID:***	125484
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Entropický generátor náhodných čísel

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte a popište problematiku entropických generátorů náhodných čísel a způsobů jejich testování. Popište a vysvětlete zdroje entropie dostupné na běžném počítači a popište metody zjišťování jejich entropie. Vybranými metodami tyto zdroje prakticky otestujte a zhodnoťte. Dále uveďte a vysvětlete základní operace používané při konstrukci entropických generátorů. Na tomto základě navrhněte vlastní řešení entropického generátoru náhodných čísel pro běžný osobní počítač. Svůj návrh zdůvodněte, prakticky realizujte a všestranně otestujte. Dosažené výsledky zhodnoťte.

DOPORUČENÁ LITERATURA:

[1] BARKER, E., KELSEY, J.: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST SP 800-90A. National Institute of Standards and Technology, Gaithersburg 2012.
[2] BARKER, E., KELSEY, J.: Recommendation for the Entropy Sources Used for Random Bit Generation. NIST SP 800-90B. National Institute of Standards and Technology, Gaithersburg 2012.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: doc. Ing. Karel Burda, CSc.
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá problematikou generování náhodných čísel pomocí entropických generátorů. V rámci studie jsou popsány a vysvětleny zdroje entropie dostupné na počítači a popsány metody zjišťování jejich entropie. V práci jsou také rozebrány základní operace využívané při jejich konstrukci, návrh celého konceptu řešení daného generátoru pro osobní počítač podle doporučení NIST a následné způsoby jeho testování.

KLÍČOVÁ SLOVA

entropie, generátor, testování, GNP, GPNP, NIST

ABSTRACT

This paper is focused on generating random number via entropy generators. There are described sources of entropy on personal computer and described methods of detecting their entropy. There are also described basic operations used for construction and for concept solution this generator under NIST recommendation for personal computer and methods of their testing.

KEYWORDS

entropy, generator, testing, GNP, GPNP, NIST

KOLÁŘ, Michal *Entropický generátor náhodných čísel*: diplomová práce. místo: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 80 s. Vedoucí práce byl doc.Ing. Karel Burda, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Entropický generátor náhodných čísel“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

místo ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Karlu Burdovi, CSc. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Poté bych chtěl také poděkovat mojí rodině, která mě neustále podporovala a věřila mi.

místo ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

1 ÚVOD

Generátory náhodných čísel lze rozdělit na tři základní skupiny:

- generátory náhodných posloupností
- generátory pseudonáhodných posloupností
- entropické generátory

V této práci se blíže zaměřím na problematiku entropických generátorů. Jak z názvu vypovídá, jsou založeny na entropii nějakého systému. Entropie je veličina, která popisuje míru neurčitosti systému. Takový systém se může nacházet s různými pravděpodobnostmi v různých stavech. Tyto generátory budou využity pro generování náhodných čísel. V práci bude také rozebrána problematika náhodných čísel, jejich generování na osobním počítači a jejich využití v praxi. Generátory náhodných čísel jsou v současné době velmi aktuální, neboť je využívají kryptografické algoritmy pro tvorbu šifrovacích klíčů apod. Z tohoto důvodu musejí být co nejúčinnější a nejbezpečnější, aby případný útočník nemohl odhadnout generátorem generované číslo, které by pak zneužil k prolomení algoritmu. Takový generátor podle doporučení NIST naprogramuji a otestuji.

2 GENERÁTORY

Generátory náhodných čísel se nejčastěji využívají v kryptografii. Jsou široce využívané například v šifrovacích protokolech na internetu jako je Secure Socket Layer (SSL). Kryptografie klade důraz na to, aby tyto klíče nebylo možné předem odhalit ať už celé nebo jejich část, protože odvození generovaného klíče by dalo útočníkovi značnou výhodu. Aby se tomuto zabránilo, jsou parametry pro generátory voleny náhodně podle rovnoměrného rozdělení. V praxi se pro generování náhodných čísel využívají generátory binárních posloupností, které se dělí na několik základních typů.

2.1 Generátor náhodných posloupností

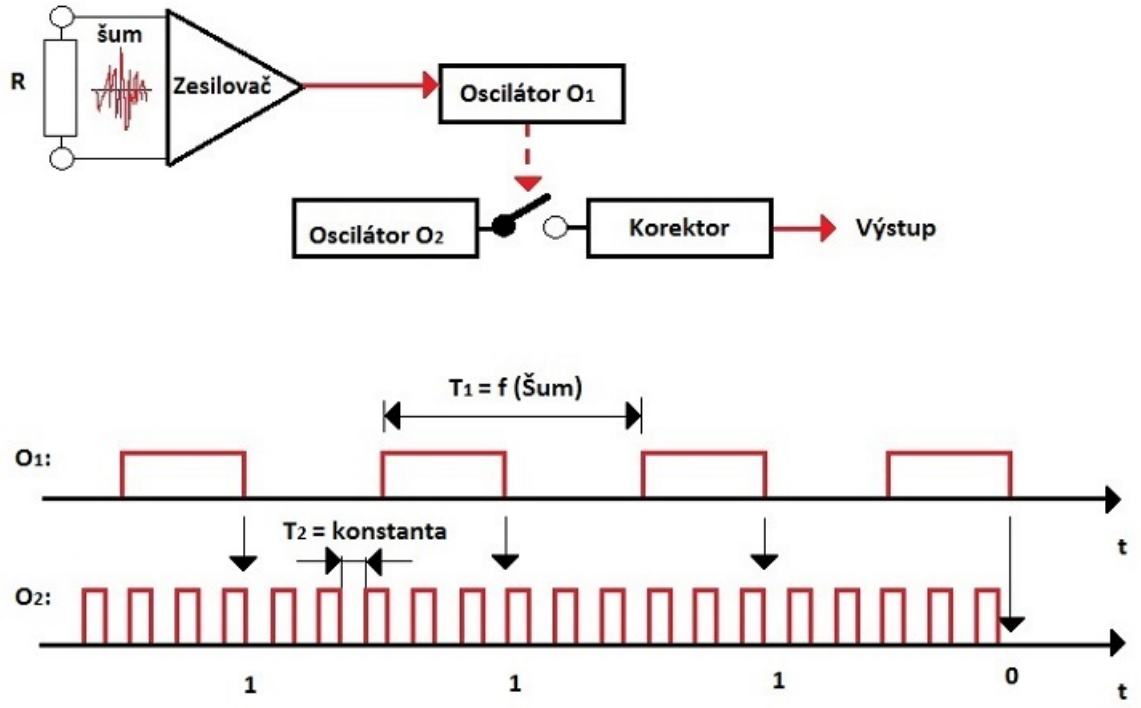
Tyto generátory využívají ke generování náhodné binární posloupnosti náhodné procesy. Mezi nejvíce využívané patří kvantové děje, jako například rozpad atomů radioaktivního prvku. Mezi další děje využitelné pro generování mohu uvést různé šumy nebo fluktace. Tyto zdroje jsou sice dostatečně náhodné, ale signály jsou velice slabé, a proto je nutné je prvně zesílit, a až poté navzorkovat. Takto odvozená posloupnost bitů samozřejmě není konečná a generátor ji dále pomocí vhodných algoritmů upravuje tak, aby byly odstraněny její statistické závistloti. Další nevýhodou pro praktické využití je rychlost, protože tyto generátory produkují jen omezené množství bitů za sekundu. Proto se nejčastěji využívají pouze pro generování náhodného inicializačního semínka pro značně rychlejší pseudonáhodné generátory.

Příklad generátoru náhodné posloupnosti v procesorech Intel Pentium III je zobrazen na obrázku 2.1.

V generátoru, uvedeném na obrázku, se jako náhodný proces využívá tzv. Johansonova šumu, což je v tomto případě teplotní šum rezistoru R. Pro úpravu výstupních posloupností z generátorů podobných tomuto se často využívá von Neumannova korektoru, jehož účelem je eliminovat nerovnoměrný výskyt jedniček a nul. Mezi další úpravy výstupních posloupností patří statické testování, které je prováděno v závislosti na míře bezpečnosti buď v pravidelných časových intervalech nebo před každou vygenerovanou sekvencí.[5]

2.2 Generátor pseudonáhodných posloupností

Tento druh generátorů, označovaných jako GPNP, využívá deterministických postupů pro odvozování binárních posloupností. Vytváří řetězce náhodně uspořádaných bitů, které jsou při aplikaci vhodného algoritmu dostatečně odolné i pro využití



Obr. 2.1: Generátor náhodné posloupnosti v procesorech Intel Pentium III. [5]

v kryptografii. Základem každého pseudonáhodného generátoru je nějaký automat, který slouží k inicializaci tohoto generátoru. Tento automat má konečný počet stavů, které postupně mění a každému změněnému stavu odpovídá jiná hodnota výstupního bitu. Nevýhodou je tedy perioda opakování náhodné posloupnosti, naopak výhodou je snadná implementace takových generátorů i jejich rychlost generování náhodných posloupností. Generátor se tak chová v podstatě jako jednosměrná funkce a útočník by neměl být schopen z výstupu odvodit, jak vypadala vstupní hodnota.[5]

Základní dělení je na lineární a nelineární generátory. Lineární mají vynikající statické vlastnosti, ale jsou málo odolné proti kryptoanalýze, kdežto nelineární jsou vůdči kryptoanalýze mnohem odolnější.

Jako příklad lineárního generátoru lze uvést Korgurentní generátor (LKG). Je to jeden z vůbec prvních a nejjednodušších generátorů, který se začal používat již v padesátých letech dvacátého století. Je definován vztahem:

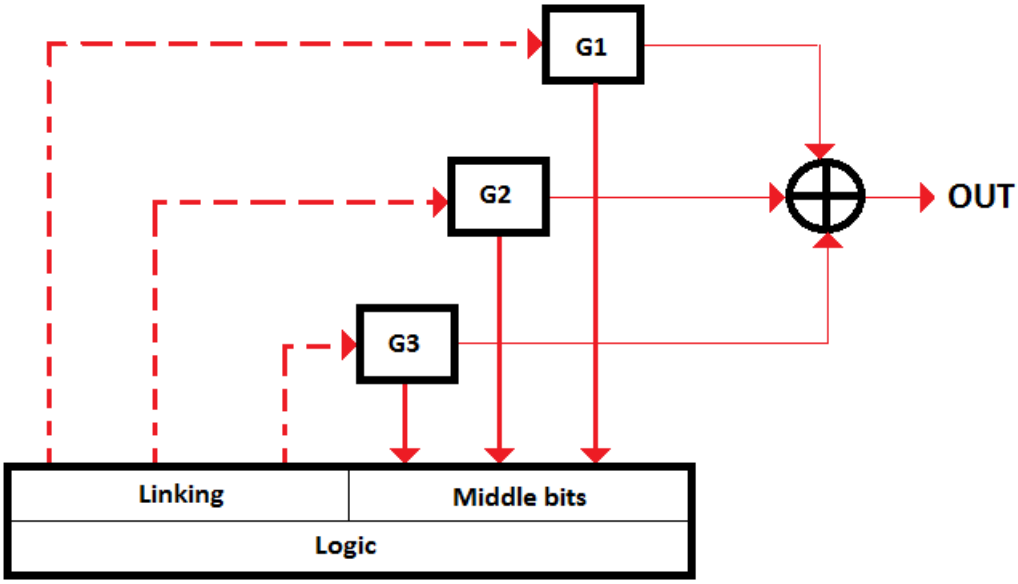
$$x_{i+1} = (ax_i + c) \bmod m \tag{2.1}$$

Konstanty **a**, **c**, **m** jsou vhodně zvolená čísla přičemž:

- **c** a **m** jsou nesoudělná čísla
- **a-1** je dělitelné všemi provočíselnými faktory **m**
- **a-1** je násobek 4, jestliže **m** je násobek 4

Výstupem generátoru jsou celá čísla, která jsou rovnoměrně rozložena v rozsahu $0 \leq x_i < m$. Maximální počet možných hodnot je v tomto rozsahu omezený, a proto se začne nejpozději po **m** vygenerovaných číslech opakovat stejná posloupnost - zde se projeví výše zmíněná nevýhoda periody opakování. Navíc při nevhodně zvolených konstantách se periodicita projeví mnohem dřív. Další nevýhoda generátoru se projevuje opět při špatně zvolených konstantách **a**, **c** a **m** pro generování náhodných bodů v prostoru. Vygenerované body leží jen v několika málo rovnoběžných rovinách (někdy dokonce méně než v 5% možných ploch) a zbytek prostoru je prázdný, přičemž by měl být rovnoměrně vyplněný v celém svém objemu.

Další významnou nevýhodou tohoto generátoru je situace kdy, uživatel zvolí **m** jako mocninu o základu 2. Pak ve výsledné vygenerované sekvenci bitů jsou bity nižších řádů nenáhodné. Například LSB má periodu maximálně 2 a pro každý další bit tato hodnota podle druhé mocniny stoupá.



Obr. 2.2: Generátor využivaný v mobilních telefonních sítích.[5]

Nelineární generátor nelze popsat soustavou lineárních rovnic. K dispozici je celá řada způsobů, jak takový generátor realizovat. Příkladem může být generátor

pro šifrování hovorů v mobilních telefonních sítích (šifra A5/1). Schéma generátoru ukazuje obrázek 2.2.

Celý generátor je tvořen pomocí třech generátorů LFSR. Ty jsou ve schématu označené jako **G1**, **G2** a **G3**. Všechny výstupní bity jednotlivých generátorů jsou mezi sebou xorovány a tím vznikne výsledný výstupní bit. Počet buněk v registrech jednotlivých generátorů je $r_1 = 19$ bitů, $r_2 = 22$ bitů a $r_3 = 23$ bitů. Nejsou ovšem sledovány hodnoty všech bitů pro každý generátor, ale jen střední bity. Jakmile má jeden z generátorů odlišnou hodnotu tohoto středního bitu od ostaních, dojde k jeho zastavení po daný takt. Tím pádem se nezmění jeho stav, což způsobuje potřebnou nelinearitu celého systému. Generátor je pak mnohem více odolnější kryptoanalýze než samostatný LFSR.[5]

2.3 Entropický generátor

Entropický generátor je kombinací obou předchozích generátorů tzn. kombinací GNP a GPNP. První zmíněný se využívá pro inicializaci a jeho hlavní součástí je nějaký zdroj entropie. Druhý zmíněný se používá především pro svoji statistickou spolehlivost a rychlost.

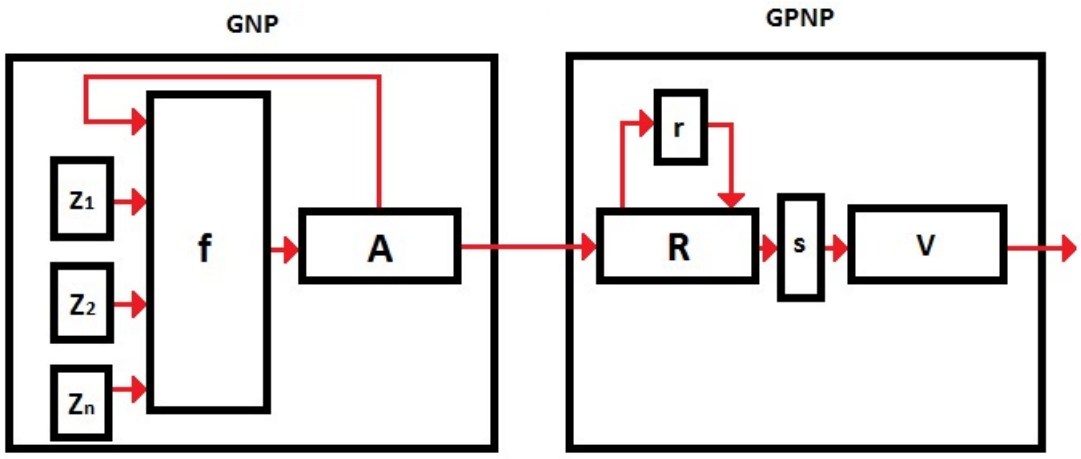
Informační entropie úzce souvisí s termodynamickou entropií ve fyzice. Často se také nazývá jako Shannonovou entropií podle Claude E. Shannona, který zformuloval mnoho zásadních poznatků teoretické informatiky. Pro systém, který má konečný počet možných stavů $S \in \{s_1, s_2, ..., s_n\}, n \leq \infty$, a který má pravděpodobnostní distribuci $P(s_i)$, je informační entropie definována jako střední hodnota:

$$H(S) = -\sum_{i=1}^{n} P(s_i) \log_2 P(s_i) \tag{2.2}$$

V tomto případě je entropie střední hodnota informace jednoho kódovaného znaku - bitu. Shannonova entropie také tvoří limit při bezeztrátové kompresi dat, která udává, že komprimovaná data nelze beze ztráty informace „zhustit“ více, než dovoluje jejich entropie. Protože uvažujeme rovnoměrné rozložení, maximální entropie systému pro $P(s_i) = \frac{1}{n}$ pro $\forall i$ bude:

$$H(S) = -\sum_{i=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = -\log_2 \frac{1}{n} = \log_2 n \tag{2.3}$$

Jako zdroje entropie pro tyto generátory se nejčastěji využívají například náhodné pohyby myší, kolísání otáček pevného disku nebo náhodné bity na výstupu zvukové karty, když je vypojený mikrofon. GPN z těchto zdrojů energie akumuluje posloupnost bitů, kterou po dosažení daného objemu přenese do GPNP. Ta slouží pro tento generátor jako inicializační semínko a samotný generátor GPNP je základ entropického generátoru. Pokud je hodnota P_0 rovna 1 nebo 0, má zdroj nulovou hodnotu entropie. Naopak pokud je hodnota $P_0 = P_1 = 0,5$ má daný zdroj maximální možnou hodnotu entropie, která je rovna hodnotě prvního bitu.



Obr. 2.3: Entropický generátor.

Zdroje entropie, které je možné využít v praxi, neposkytují takto vysokou entropii výstupních binárních symbolů, a proto využíváme těchto zdrojů více současně. Zde lze pro generování využít i GNP, jelikož není vyžadována vysoká stabilita ani spolehlivost, protože se využívá více generátorů současně. Příklad entropického generátoru popisuje schéma na obrázku 2.3.

Bloky Z_1 až Z_n jsou generátory, které využívají jako zdroje entropie například výše zmíněně náhodné pohyby myší nebo stisky kláves na klávesnici. Jakmile generátory vygenerují dostatečný objem dat (určitý počet bitů), je dále kumulován funkcí **f** do akumulátoru **A**. Ta je přenesena do bloku **R**, což je registr generátoru GPNP a slouží pro inicializaci. Celý blok GPNP slouží jako základ entropického generátoru. Funkce **s** z aktuální hodnoty stavového registru **R** odvodí obsah výstupního registru **V** a následně stavová funkce **r** na základě aktuální hodnoty stavového registru **R** odvodí novou hodnotu tohoto registru. Vygenerované pseudonáhodné bity se postupně odebírají z registru **V**. Jakmile je obsah registru **V** vyčerpán, generující funkcí **s** se odvodí nový obsah pro registr **V** a stavová funkce **r** znovu zajistí změnu obsahu stavového registru **R**.[5]

Jako další příklad entropického generátoru lze uvést BSAFE [4]. Jedná se o kryptografický generátor, který využívá hešovacích funkcí MD5 a SHA1. Lze ho charakterizovat pomocí vzorce 2.4.

$$S_j = H(S_{j-1}||X_j) \tag{2.4}$$

Operace || zde vyjadřuje zřetězení posloupností X_j) a S_{j-1}, příčemž X_j) slouží jako inicializační semínko. Operace H symbolizuje zvolenou hešovací funkci. Výsledná posloupnost na výstupu tohoto generátoru je vyjádřena funkcí 2.5.

$$Y_{j,i} = H(S_{j,i}),\ S_{j,i} = (S_j + C \cdot i) \bmod 2^L \tag{2.5}$$

Tento generátor bohužel využívá hešovací funkce MD5 a SHA1, které v dnešní době není obtížné prolomit, a proto se jejich používání nedoporučuje.

Entropické generátory jsou v současné době pravděpodobně nejvhodnějším řešením pro problematiku generování náhodných čísel pro využití v kryptografii.

Generátory náhodných čísel mohou být ale také založeny na náhodných makroskopických jevech, které využívají zařízení jako jsou ruleta nebo loterijní automaty. Tyto jevy jsou, podle teorie nestabilních dynamických systémů a podle teorie chaosu, zcela nepředvídatelné. I přesto, že podle Newtonovy mechaniky jsou makroskopické procesy deterministické, tak výsledek vhodně navrženého zařízení na pricipu například rulety nelze předvídat, protože je závislý na velmi detailních počátečních podmínkách při každém novém opakování.

2.4 Způsoby testování generátorů

Jelikož generátorů existuje celá řada, je třeba rozlišovat kvalitu náhodných čísel či posloupností, které generují. Pro tyto účely existuje velké množství testů, které hledají opakující se části sekvencí a na základě kterých lze rozhodnout o tom, zda generované sekvence jsou skutečně náhodné. Výstupy statistických testů aplikovaných na náhodné sekvence lze popsat pomocí pravděpodobnosti, protože jsou předem známé. Protože jeden test nemůže zcela rozhodnout o náhodnosti testované sekvence, aplikuje se vždy několik testů, z nichž každý sekvenci posoudí z jiného hlediska. Až výstup všech testů nám podá ucelený výsledek.

National Institute of Standards and Technology (NIST) vytvořil sadu celkem 15-ti statických testů, které rozhodují o náhodnosti binárních posloupností, které

vygenerují hardwarové i softwarové generátory. Každý z těchto testů posuzuje danou posloupnost z jiného hlediska a každý se zaměřuje na aspekty, které se mohou vyskytnout u nenahodilé sekvence.

Jednotlivé testy budou detailněji rozebrány a popsány níže. Seznam celého souboru testů: [7]

- Test četnosti
- Test četnosti v bloku
- Test shodné série
- Test nejdelší shodné série v bloku
- Test dílčích matic
- Test diskrétní Fourierovy transformace
- Test nepřekrývajících se šablon
- Test překrývajících se šablon
- Test Lineární zpětné vazby
- Test četnosti série
- Test přibližné entropie
- Test kumulativních součtů
- Test náhodných návštěv
- Test variací náhodných návštěv
- Maurerův test

2.4.1 Test četnosti

V testu se provádí kontrola celkového počtu vygenerovaných jedniček a nul v dané sekvenci. Test má určit, zda-li počet jedniček a nul je ekvivalentní s předpokládaným počtem, který se dá u náhodné sekvence očekávat. Ten by měl být přibližně shodný a pro výpočet tohoto testu se využívá Gaussovo rozdělení pravděpodobnosti. Všechny další zmíněné testy jsou závislé na výsledku tohoto testu.

2.4.2 Test četnosti v bloku

V testu se daná sekvence rozdělí na určitý počet bloků o velikosti **M** bitů a zkoumá se počet jedniček a nul v každém tomto bloku zvlášť. Četnost by v každém bloku měla být rovna přibližně **M/2**, jak je u náhodné sekvence předpokládáno. Test využívá referenčního rozdělení, které určuje míru shody daného podílu v bloku o délce **M** bitů s očekávaným podílem **M/2**. To odpovídá rozdělení pravděpodobnosti χ^2.

2.4.3 Test shodné série

V sekvenci se může shodná série definovat jako nepřerušená řada stejných bitů (jedniček nebo nul). Taková sekvence má danou délku bitů k, obsahuje přesně n shodných bitů a je ohraničena vždy bity opačné hodnoty. V testu se zkoumá, kolik takových shodných sérií se v dané sekvenci vyskytuje. V testu se rozhoduje, zda-li tento počet shodných sérií jedniček a nul o různých délkách k, je možné očekávat od náhodné sekvence. Výsledek testu tedy udává, je-li frekvence změny jedniček a nul přílíš vysoká, příliš nízká nebo v pořádku.

2.4.4 Test nejdelší série v jednom bloku

Podobně jako v testu shodné série se i zde zkoumá určitá série stejných bitů. V tomto testu je definována jako nejdelší série jedniček o velikosti M bitů z posuzovaného bloku a cílem tohoto testu je určit, zda-li je tuto velikost M shodných bitů možné očekávat u náhodné sekvence. Testování se provádí pouze pro shodné série jedniček, protože pokud se v testu projeví neočekávané hodnoty pro tyto série, tak je zřejmé, že neočekávané hodnoty by se objevily i pro shodné série nul. Pro výpočet statistiky je opět využito rozdělení pravděpodobnosti χ^2.

2.4.5 Test dílčích matic

V testu jsou zkoumány hodnoty submatic v rámci celé sekvence. Celou sekvenci o délce n bitů rozdělíme na matice $M \cdot Q$. Celkový počet těchto matic pak bude roven:

$$N = \frac{n}{M \cdot Q} \tag{2.6}$$

Pokud nějaké bity přebývají, jsou zanedbány. Výsledkem testu je zhonocení lineární závislosti mezi řadami konstantních délek v rámci celé zkoumané sekvence.

2.4.6 Test diskrétní Fourierovy transformace

Jak již z názvu vypolývá, test využívá pro výpočet spektra zkoumané sekvence diskrétní Fourierovy transformace. Ve vypočítaném spektru se prověřují nechtěné výskyty periodicity. Ty by totiž naznačovaly, že se nejedná o náhodně vygenerovanou sekvenci. Všechny složky spektra by měli pro danou sekvenci být shodně zastoupeny s maximální povolenou odchylkou 5 %.

2.4.7 Test nepřekrývajících se šablon

Protože existují generátory, jejichž výstupem je příliž mnoho ekvivalentních a periodických sekvencí, tak je zapotřebí tyto sekvence testovat. Tento test odhaluje právě tyto generátory. V testu se zkoumá počet výskytů předem nadefinovaných sekvencí bitů v celé vygenerované sekvenci. Pro testování je použito okno délky n bitů, které odpovídá délce hledané definované šablony. Pokud se bity definované šablonou v tomto okně nenechází, okno se posune o jeden bit dál. Pokud dojde k nalezení šablony, okno se pak posune na následující bit po nalezené šabloně.

2.4.8 Test překrývajících se šablon

Princip testu je téměř totožný, jako v testu nepřekrývajících se šablon. Tudíž také zkoumá počet výskytů předem nadefinovaných sekvencí bitů v celé vygenerované sekvenci. Pro testování je použito okno délky n bitů, které odpovídá délce hledané definované šablony. Rozdíl je až v tom, že pokud je nalezena hledaná sekvence, okno se posouvá pouze o jeden bit dále a ne až na následující bit po nalezené šabloně.

2.4.9 Test Lineární zpětné vazby

Náhodné sekvence jsou charakterizovány větší délkou bitů pro Linear Feedbak Shift Register (LSFR). Test se tedy zaměřuje na zkoumání délky tohoto posuvného registru a určuje její složitost. Ta by měla být dostatečně složitá, aby se dala považovat za náhodnou. Pokud je tedy výsledná bitová délka LSFR pro testovanou sekvenci přílíš krátká, lze tvrdit, že se nejedná o náhodnou sekvenci.

2.4.10 Test četnosti série

Protože náhodná čísla mají rovnoměrné rozložení, tak každý n-bitový vzor má stejnou pravděpodobnost výskytu, stejně jako všechny ostatní n-bitové vzory. Tento test má tedy určit, zda-li počet 2^m n-bitových vzorů zkoumané sekvence odpovídá náhodné sekvenci. Kdybychom pro tento test uvažovali hodnotu $m = 1$, jednalo by se o první zmiňovaný test - Test četnosti.

2.4.11 Test přibližné entropie

Princip testu je stejný jako při testu četnosti série, který je zmíněný výše. Test je také zaměřen na výstkyt n-bitových vzorů ve zkoumané sekvenci. U toto testu se ovšem sleduje četnost vzorů n a $m + 1$.[7]

2.4.12 Test kumulativních součtů

Random Walk (RW) lze definovat jako kumulativní součet sousedních bitů v sekvenci. Test je zaměřen na střední vzdálenost od počátečního bodu průběhu RW. Test rozhodne, zda-li je kumulativní součet úseků testované sekvence buď moc malý nebo moc velký vzhledem k očekávané hodnotě, kterou by měla mít náhodná sekvence. Pro náhodné sekvence se střední vzdálenost od počátečního bodu blíží nule, zatímco pro nenáhodné procesy bude dosahovat znatelně vyších hodnot.

2.4.13 Test náhodných návštěv

V průběhu testu jsou zkoumány počty cyklů, které se během kumulativního součtu RW navštíví přesně X-krát. Tento kumulativní součet je odvozen z částečných součtů poté, co je sekvence (0,1) transformována na sekvenci (-1,1). Celý cyklus RW se skládá ze sekvence kroků stejné, náhodné délky, která začíná od začátku sekvence a končí jejím posledním bitem. Testuje se, zda-li počet návštěv jednotlivých stavů v průběhu cyklu lze považovat za náhodný. Celý tento test je série 8-mi menších testů, kdy každý tento test probíhá s jinou hodnotou stavu. Stavy pro testování: $-4, -3, -2, -1$, 1, 2, 3, 4.

2.4.14 Test variací náhodných návštěv

Narozdíl od předešlého testu se tento zaměřuje na celkový počet návštěv jednotlivých stavů při kumulativním součtu RW. Výstupem testu je pak odchylka v počtu návštěv od očekávaného počtu, který by měla mít náhodná sekvence. Celý test je souborem 18ti testů pro stavy $-9, -8, -7, -6, -5, -4, -3, -2, -1$, 1, 2, 3, 4, 5, 6, 7, 8, 9.

2.4.15 Maurerův test

Jedná se o univerzální statistický test. Test zkoumá, zda-li je možné sekvenci zkomprimovat tak, aby nedošlo ke ztrátě informace. Pokud je nějakou sekvenci možné markantně zkomprimovat a nedojde k žádné ztrátě infromace, je považována za náhodnou.[7]

3 ZDROJE ENTROPIE OSOBNÍHO POČÍTAČE

Na osobním počítači se nachází velké množství zdrojů entropie, které mohou sloužit jako inicializační semínko pro generátory náhodných čísel. Většinou jsou založeny na mikroskopických dějích, které generují nízkoúrovňový, statisticky náhodný signál. Takový signál může být třeba tepelný šum fotoelektrického jevu, či jiných dalších kvantových jevů. Tyto procesy jsou teoreticky zcela nepředvídatelné. Většina hardwarových generátorů náhodných čísel se skládá právě z převodníku těchto fyzikálních jevů na elektrický signál. Ten se většinou musí dále zesilovačem zesílit a dalšími obvody upravit, aby získal měřitelnou úroveň. Nakonec pak analogový nebo digitální převodník převádí na výstup binární sekvenci jedniček a nul. Některé jsou vhodnější, některé méně, některé například vyžadují přítomnost uživatele a nejsou proto vhodné pro počítače v serverovnách.[6]

Vhodné zdroje entropie osobního počítače:

- Klávesnice
- Myš
- Obsah disků
- Otáčky disku nebo ventilátoru procesoru
- Datum a čas
- Šum na vstupu zvukové karty
- Pakety na vstupu síťové karty
- Množství volné paměti
- Podrobnosti často měněných souborů
- Teplota CPU
- Pixely na monitoru
- Tepelný šum odporu
- Šum na vstupu TV karty

3.1 Metody zjišťování entropie

Pro měření entropie se často používá odhad pomocí výkonných kompresních algoritmů. Výstupní řetězec ze zdroje entropie o velikosti **n** bitů zkomprimujeme vhodným kompresním programem a dostaneme řetězec o délce **d** bitů. Střední entropii **E** tohoto zdroje je poté možné odhadnout podle: $E = d/n = x$ [Shannon/bit]. Z tohoto vzorce vyplývá, že $1/E = x$ bitů na výstupu zdroje entropie v sobě obsahuje

neurčitost jednoho skutečně náhodného bitu.

Pro měření je třeba porovnat výsledky alespoň dvou různých kompresních programů. Porovnání by mělo být provedeno pro různě dlouhé posloupnosti bitů, pro dlouhé posloupnosti by se hodnoty E měly ustálit.

V této kapitole popíšu možnosti využití každého z výše jmenovaných zdrojů, změřím hodnoty entropie, které jsou schopny poskytnout a jejich výhody i nevýhody. Na závěr této kapitoly uvedu celkové zhodnocení všech zdrojů entropie a vyberu několik nejvhodnějších.

3.1.1 Měření entropie

Jako první kompresní program jsem zvolil **7zip**. Je freewarový software, vyvíjený Igorem Pavlovem a distribuován pod licencí GNU LGPL. Je konkurencí pro známé programy jako jsou WinZip nebo WinRAR. 7-zip používá přednostně kompresní algoritmus **LZMA**. Kromě toho nabízí i další kompresní algoritmy jako PPMD, bzip2 nebo Deflate, ale právě **LZMA** má požadovaný velmi vysoký komprimační poměr.

Jako druhý kompresní program jsem se rozhodl zvolit **gZIP**. Jedná se také o freewarový software, který je založený na algoritmu **DEFLATE**. Ten je kombinací LZ77 a Huffmanova kódování. **DEFLATE** byl určený k nahrazení LZW a dalších patentem-zatížených algoritmů pro kompresi dat, který měl v té době omezenou použitelnost komprese a dalších populárních archivátorů.

Měření jsem vždy prováděl pro bloky dat o velikosti 50 000, 100 000, 500 000 a 1 000 000 bitů, pokud zdroje takové množství dat dokázaly v rozumném čase poskytnout. Pro některé jsem musel zvolit jiný postup měření, který vždy pro konkrétní zdroj popíšu. Výstupní posloupnost ze zdroje entropie jsem vždy srovnal se stejnou posloupností zkomprimovanou pomocí obou zvolených programů a výsledná hodnota je aritmetický průměr entropií obou zkomprimovaných posloupností. Jako efektivnější se nakonec ukázal **7zip**, ale rozdíl nebyl markantní. Při porovnávání bylo nutné uvažovat i hlavičky kompresních programů, které zůstávají vždy stejné. Z tohoto důvodu jsem ze zkomprimovaných dat vždy udělal dump pomocí programu **Hiew32**, čímž jsem hlavičku odstranil.

3.1.2 Klávesnice a myš

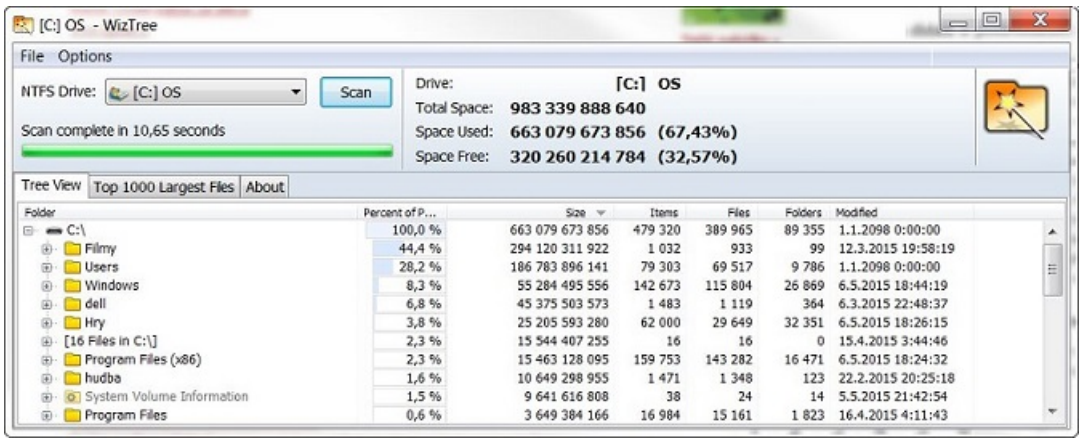
Jednou z nejčastějších možností pro generování náhodných čísel je využití klávesnice nebo myši. Náhodné stisky kláves nebo pohyby myší slouží pro generování inicializačního semínka. Toho využívá už velmi dlouho celá řada softwarů, které pro zahájení generování vyžadují od uživatele stisk nějaké klávesy nebo pohyb myši.

Program zaznamená tyto hodnoty a čas, kdy k události došlo. Tyto informace nelze předem nějak předvídat, takže inicializační semínko pro generátor lze považovat za velmi bezpečné. Výhodou této metody je jednoduchá implementace. Bohužel nevýhod u této metody je také celá řada. Nelze ji použít například pro serverové počítače, ke kterým je jen velmi omezený přístup, případně daný počítač nemusí mít implicitně myš a klávesnici připojenou, jako je tomu například u moderních serverů pro správu kamerových systémů. Z vlastní zkušenosti mohu jmenovat například servery od GEUTEBRÜCK nebo od CISCO. Další podstatnou nevýhodou je pak kolize více procesů, které vyžadují od uživatele zadání náhodného vstupu. Tyto nevýhody lze nazvat jako technické. Další nevýhodou spojenou s tímto způsobem generování náhodného semínka je samotný lidský faktor. Pokud uživatel, například z důvodu urychlení celého rutiního procesu, bude mačkat nebo držet stále stejnou klávesu, dojde ke značnému znehodnocení kvality vstupního semínka pro generátor, a celý systém, který by takový generátor využíval, by se stal mnohem zranitělnější.

Měření jsem se rozhodl provést dvě různá. První pro uživatele, který se snaží poskytnout co nejvíce náhodná data a druhé pro uživatele, který chce urychlit process generování tím, že a používá stejné pohyby myši a nízký počet kláves. Pro obě měření jsem vytvořil dostatečně dlouhou posloupnost dat, abych ji pomocí **Hiew32** mohl rozdělit na požadované bloky pro měření. Výsledek prvního měření byl průměrně 0,324 Sh/b. Výsledek druhého měření byl průměrně 0,049 Sh/b.

3.1.3 Obsah disků a podrobnosti souboru temp

Další možnosti vycházejí z dat uložených přímo na počítači, které se velmi často mění. Pro generování inicializačního semínka lze také využít například i podrobností souboru Temp. Obsah tohoto souboru je v podstatě neustále měněn a tím pádem i jeho podrobnosti. Je proto vhodným zdrojem entropie.



Obr. 3.1: Měření volného místa na disku

Měření jsem prováděl pomocí utility **WizTree**. Výhodou této metody je opět poměrně snadná implementace. Nevýhoda a jistá zranitelnost takto generovaných náhodných čísel je dána tím, že vstupní data pro tento proces se dají zjistit. Pokud by tedy daný počítač byl napaden a útočníkovi by se podařilo zpřístupnit data uložená na disku, získal by tím i vstupní hodnoty pro generátor a mohl by je zneužít. A protože se pro zpracování těchto dat často používají veřejně publikované metody, stačilo by mu do nich zjištěné hodnoty dosadit. Takto by jednoduše získal vygenerované hodnoty například kryptografických klíčů, které byly považovány za náhodné, a tudíž bezpečné, a mohl by je dále zneužít.

Zaznamenání velkého množství dat by pomocí této metody trvalo dlouho, takže jsem si vystačil s kratší posloupností. Navíc se měnilo pouze několik posledních bitů, což se nakonec promítlo i ve výsledné entropii. Měření delší posloupnosti by tedy dalo opět téměř stejnou hodnotu. Výsledkem měření entropie pro obsah disků bylo průměrně 0,081 Sh/b. Pro podrobnosti souboru Temp byl výsledek o trochu lepší a to 0,107 Sh/b.

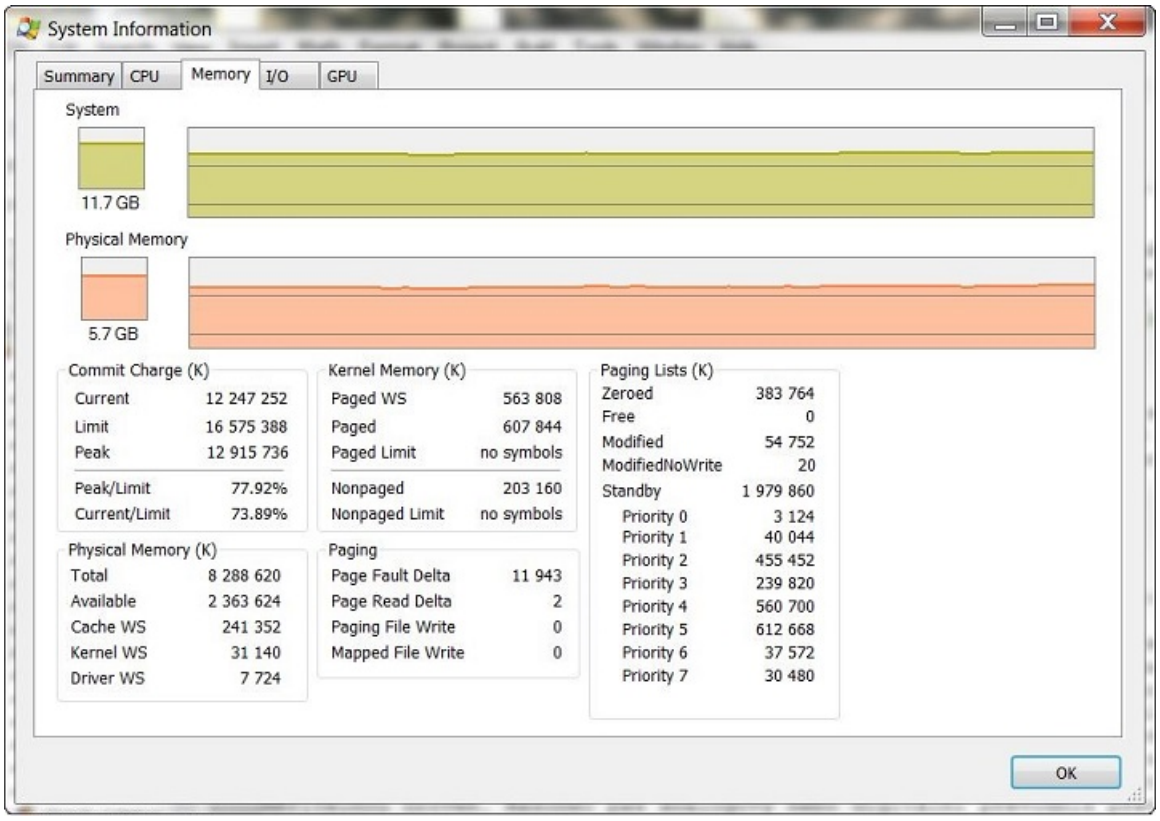
3.1.4 Datum a čas, množství volné paměti a využití CPU

Datum a čas jde využít také jako zdroj entropie, protože se neustále mění. Výhodou je také snadná implementace. Nevýhodou ovšem je, že se v podstatě mění jen posledních pár bitů, které představují setiny a vteřiny. Navíc je zde podstatná periodicita a tudíž i poměrně snadná odvoditelnost.

Volné množství paměti nebo vytížení procesoru se také mění, především pokud uživatel pracuje. Systém Windows neustále zapíná různé procesy podle toho, jaké aplikace uživatel používá. Pokud ale uživatel vykonává delší dobu stejnou činnost, množství paměti se mění jen v posledních pár bajtech a vytížení procesoru je pak také téměř konstantní. Navíc obě dvě měřené veličiny mají často dost podobné hodnoty a může tak častěji docházet k periodicitě. Měření jsem prováděl pomocí **Process Exploreru**, se kterým mám hodně zkušeností z praxe. Zde opět stačilo zaznamenat kratší posloupnost, protože ve všech případech se nejvíce měnilo několik posledních bitů. Entropie poskytovaná měřením volné paměti byla průměrně 0,103 Sh/b, využitím CPU 0,097 Sh/b a měření data a času poskytlo 0,092 Sh/b.

3.1.5 Pakety na vstupu síťové karty

Velice dobře využitelnou možností jsou pakety na vstupu síťové karty. U každého osobního počítače se automaticky počítá s připojením k internetu a použití této karty je tedy velice vhodné. Na jejím vstupu je velmi intenzivní provoz a to i v případě, že uživatel přímo nepoužívá internet.

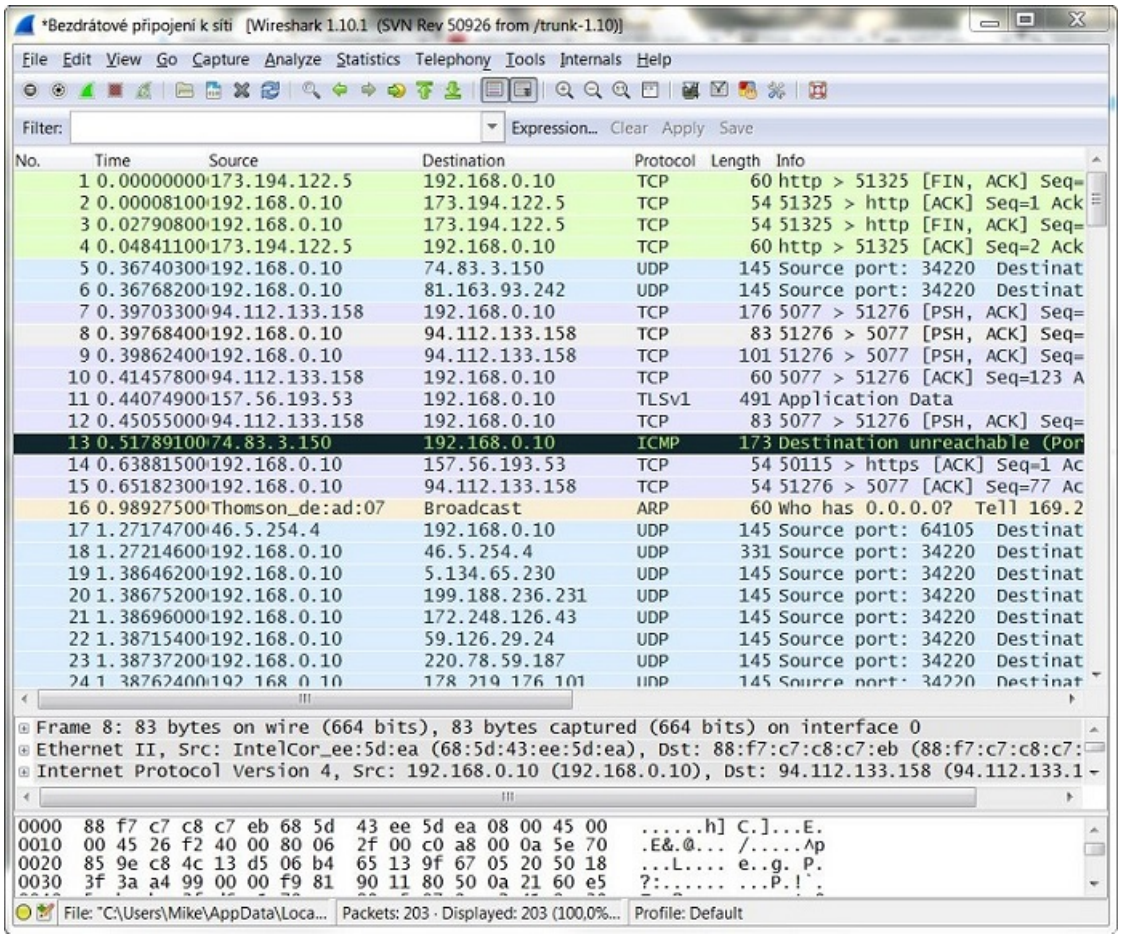


Obr. 3.2: Měření volného množství paměti.

Řada programů používá připojení k internetu pro kontrolu aktualizací, stahování nových updatů nebo filtrování provozu. Lze jmenovat například Windows Update, Bránu Firewall, libovolný antivirový program nebo program pro stahování torrentů. Možným příkladem je bohužel i nejrůznější malware, který často internet využívá pro šíření.

Na obrázku 3.3 jsou vidět zachycené pakety za necelé dvě vteřiny měření. Je tedy vidět, že i takto krátké měření obsahuje velké množství paketů, které je možné použít pro tvorbu inicializačního semínka. Z paketu je přitom možné použít například data z těla paketu, kombinaci IP adres a dalších dat. Možností a různých kombinací je tedy skutečně obrovské množství.

Výhodou tohoto zdroje entropie je nezávislost na nějakém zásahu nebo zadání vstupních informací od uživatele. Další výhodou je, že výstupní hodnoty z tohoto zdroje nemají žádnou periodu opakování a jsou zcela náhodné. Použití tohoto zdroje entropie pro tvorbu inicializačního semínka se tedy jeví jako velmi spolehlivé a bezpečné řešení. Další výhodou je i dobrá implementace, protože pro práci s internetem a analýzu paketů jsou k dispozici ověřené knihovny pro jazyky C a C++ a dobře popsané tutoriály.



Obr. 3.3: Zachycené pakety pomocí wiresharku.

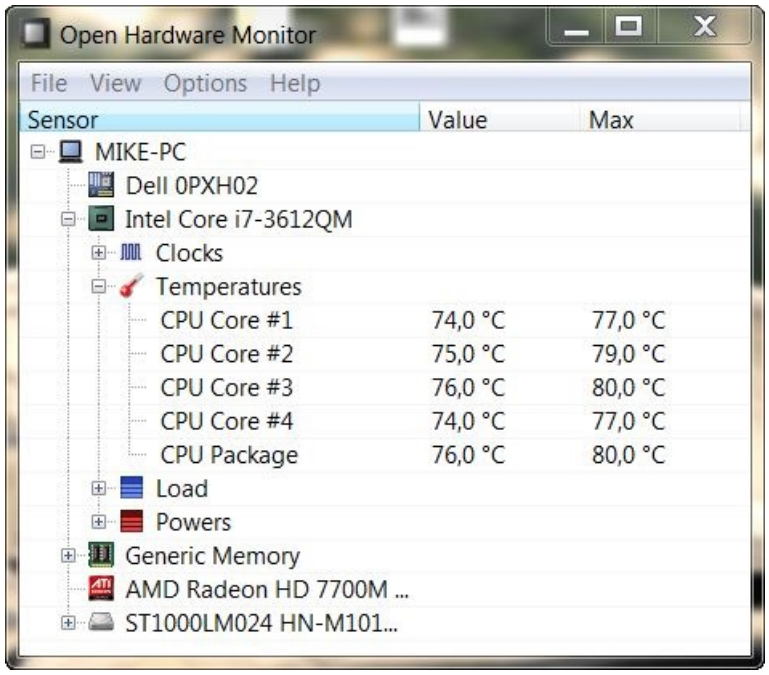
Naměřená data jsem zaznamenával do textového souboru a rozdělil je na jednotlivé bloky požadovaných délek. Poté zkomprimoval a pomocí **Hiew32** z něj dumpnul hlavičky. Tento zdroj vykázal druhý nejlepší výsledek ze všech testovaných zdrojů. Průměrně poskytovaná entropie byla 0,823 Sh/b.

3.1.6 Tepelný šum odporu a teplota CPU

Do další kategorie možností, které lze využít pro vytvoření inicializačního semínka, se řadí jevy náhodných fyzikálních procesů, jako jsou například změny teploty CPU a teplotní šum odporů.

Měření jsem prováděl pomocí utility **Open Hardware Monitor** na svém ntb. V případě stolního PC může sloužit i k měření otáček chladičů jednotlivých procesorů, které se dají také využít jako zdroj entropie.

Nevýhodou je, že teplotní šum odporů se těžce měří a teplota CPU má často velice podobnou hodnotu, což jsem ověřil i měřením. Při více jádrech se situace poněkud zlepší, ale stále neposkytuje uspokojivé výsledky. Chladič každého CPU



Obr. 3.4: Měření teploty procesorů

ji stále udržuje na podobné hodnotě. Rozdíl nastává, když počítač vykonává náročnější činnost, ovšem i když se teplota zvýší, tak ji chladič bude držet opět na podobné hodnotě. Stejně jako v případě data a času nebo volné paměti, i zde je velká náchylnost pro periodicitu.

Měření probíhalo stejně jako v kapitole 3.1.4 a výsledky tohoto zdroje byly 0,058 Sh/b.

3.1.7 Pixely na monitoru

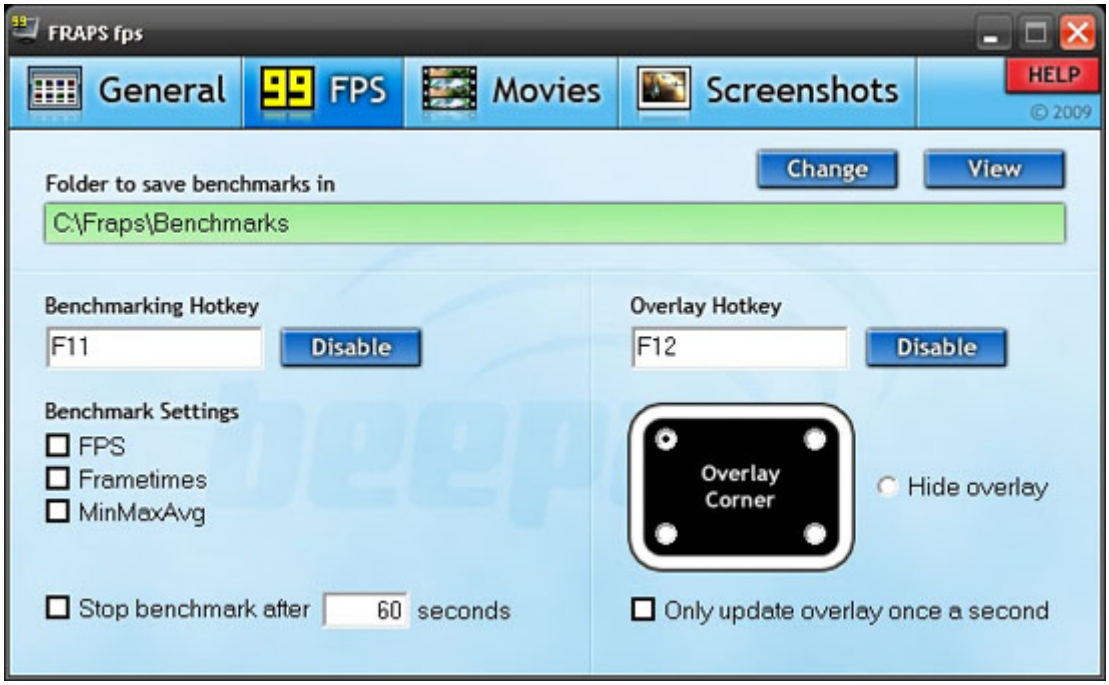
Pohyby myši, otevírání nových oken a další činnosti, které při běžné práci na počítači uživatel dělá, způsobují změny obrazu. Tyto změny je možné také využít jako zdroj entropie. Nabízejí se dvě možnosti, jak efektivně využít těchto změn.

První možností je, že generátor při inicializaci vytvoří defaultní screenshot obrazovky, proti kterému bude porovnávat další screenshot, vytvořený při požadavku na generování pseudonáhodných posloupností. Generátor spočítá rozdíl mezi pixely v pořízeném screenshotu oproti defaultnímu a tato data použije pro tvorbu inicializačního semínka.

Druhou možností je při zavolání požadavku na generování náhodných dat nahrát několik vteřin záznamu. Z tohoto záznamu jsou poté vypočítány změny, které se udály a tato data jsou následně využita pro tvorbu inicializačního semínka.

Obě možnosti mají své výhody i nevýhody. V první možnosti je vypočítávání roz-

dílu velice rychlé, a pokud generátor využívá stabilizační komponent, stačí i změna jednoho pixelu, aby výsledná entropie pro tvorbu semínka byla zcela odlišná od té defaultního screenshotu. Nevýhodou ovšem je velice náročná implementace, pro jazyky C a C++ není žádná ani freewarová knihovna, která by implementaci usnadnila pomocí předem připravených API funkcí. V možnosti druhé je výhodou kvalitnější entropie, protože se vypočítává z několikavteřinového záznamnu. Je proto dobře použitelná i bez stabilizačního komponentu. Nevýhody jsou zde ovšem značné. První nevýhodou je vcelku pomalé a náročné zpracování, protože zpracování zaznamenaného obrazu je náročné na výpočet pro CPU a nahrání zabere také několik vteřin, takže generátor by to zpomalovalo. Druhou nevýhodou je jako v případě první možnosti velmí náročná implementace. Opět není k dispozici knihovna nebo hlavičkové soubory pro usnadnění implementace. Je tu možnost využití freewarového softwaru pro záznam obrazu, jako je například **Fraps**, který je možné stáhnout přímo z oficiálních stránek `http://www.fraps.com/download.php`.



Obr. 3.5: Aplikace Fraps pro záznam obrazu.

Generátor by pak volal jen spuštění Fraps, který by vytvořil záznam a daný soubor by už zpracoval dále algoritmus generátoru. Ovšem tím by se generátor stal špatně přenositelný a bylo by nutné k jeho používání mít nainstalovanou i tuto aplikaci.

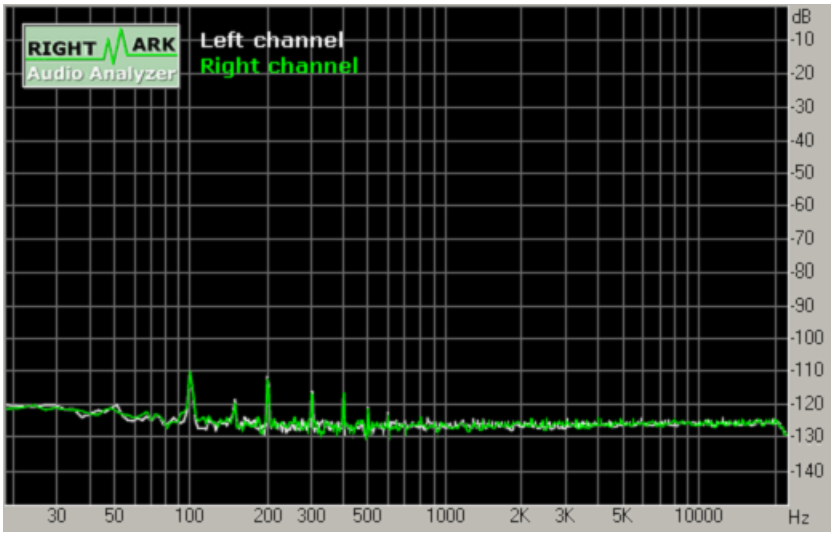
V tomto měření jsem využil první popsanou možnost a pomocí **Hiew32** jsem odstraňoval hlavičky i z vytvořených screenshotů, protože formáty JPEG, PNG apod.

ji mají také vždy stejnou. Takto upravené soubory jsem poté porovnával s dalšími screenshoty a poté s komprimovanými. Měření prokázalo nejlepší výsledek v testu, průměrná etropie byla 0,971 Sh/b, a to i při velmi malé změně obrazu.

3.1.8 Šum na vstupu zvukové karty

Další možností je využití dat ze zvukové karty. Pracovní počítač se často nachází ve velmi rušném prostředí a na vstupu zvukové karty jsou tak k dispozici náhodná data, která můžeme velice dobře využít jako zdroj entropie. Uživatel také často sleduje filmy nebo poslouchá muziku, což jsou další elementy, které způsobují provoz na zvukové kartě.

Pro měření šumu a dalších vlastností zvukové karty za různých podmínek lze použít například **RightMark Audio Analyzer 5.5**. Jeho ovládání je velice jednoduché a intuitivní. Na obrázku 3.6 je měření šumu zvukové karty pro stálé pracovní prostředí.



Obr. 3.6: Měření velikosti šumu.

Stejně jako v případě síťové karty, i zde je možná kombinace více možností, jak pracovat s daty ze zvukové karty. Kromě samotného šumu je možné do materiálu pro tvorbu semínka využít i například maximální či minimální amplitudu nebo další data, která se dají libovolně nakombinovat.

Dále je nutné zvolit, jak bude generátor data zaznamenávat. První možností je neustálé čtení dat ze zvukové karty a jejich ukládání do paměti. Tato implementace je poměrně složitá pro práci s pamětí, aby nedošlo k jejímu přeplnění nebo zahlcení systému. Generátor by tedy musel udržovat pouze určitě množství dat v paměti, které by se neustále měnilo a přepisovalo, a až v případě potřeby by bylo využito pro

tvorbu inicializačního semínka nebo při jeho obnově. Toto řešení také více zatěžuje běh celého počítače.

Druhou možností je zaznamenání určitého množství dat a ta pak využít ke tvorbě semínka. Tato metoda sice trochu zpomalí inicializaci generátoru a obnovu semínka, ale je mnohem méně náročná na implementaci a nezatěžuje počítač během práce generátoru, ale jen v případech, kdy je nutné obdržení čerstvé entropie.

Výhodou tohoto zdroje entropie, stejně jako v případě zvukové karty, je nezávislost na zásahu nebo zadání vstupních informací od uživatele. Další výhodou také je, že výstupní hodnoty z tohoto zdroje nemají žádnou periodu opakování a jsou zcela náhodné. V naprosto tichém prostředí mohou být hodnoty šumu podobné, ale pokud pro tvorbu semínka použijeme kombinaci více dat, bude zaručena vždy kvalitní etropie. Pokud nebude použitá kombinace více dat, je možné použít stabilizační komponent, který zajistí kvalitní entropii. Tento zdroj entropie je tedy velice vhodný pro tvorbu inicializačního semínka i z hlediska bezpečnosti.

Menší nevýhodou je implementace, která je poněkud složitější. K dispozici je ale několik vhodných knihoven pro jazyky C a C++, které práci se zvukovou kartou usnadňují pomocí připravených API funkcí.

Pro záznam dat jsem naprogramoval funci, kterou pak používám i ve svém generátoru. Ta uloží 5 vteřin záznamu do textového souboru pro další práci. Provedl jsem dvě různá měření pro dva typy prostředí. Měření pro klidné prostředí vykázalo průměrnou hodnotu 0,112 Sh/b, pro proměnlivé prostředí pak hodnotu 0,346 Sh/b. Pokud využijeme kombinace více hodnot, jak bylo popsáno výše, získáme entropii až 0,544 Sh/b.

3.1.9 Výsledné zhodnocení

V předešlé kapitole jsem rozebral jednotlivé zdroje entropie a jejich výhody a nevýhody. Jako nejméně vhodné se jeví použití těchto zdrojů:

- Otáčky disku nebo ventilátoru procesoru
- Datum a čas
- Teplota a vytížení CPU
- Tepelný šum odporu

Důvodem je jejich slabá entropie, protože se velice málo mění a je zde velká pravděpodobnost na periodicitu. Z tohoto důvodu nejsou přílíš vhodné. Zdroje které poskytují kvalitnější entropii jsou:

- Klávesnice
- Myš
- Množství volné paměti a obsah disků
- Podrobnosti často měněných souborů

Tyto zdroje poskytují kvalitnější entropii, ale mají několik zásadních nevýhod. U klávesnice a myši je třeba, aby uživatel zadával vstupní data. Množství volné paměti se sice mění častěji, ale stále je náchylné na periodicitu. Obsahy disků a často měněných souborů jsou zase snadno dostupné pro případného účtočníka. Jako nejvhodnější zdroje entropie lze využít:

- Šum na vstupu zvukové karty
- Pakety na vstupu síťové karty
- Pixely na monitoru

Tyto zdroje poskytují nejkvalitnější entropii, protože se jedná o velice náhodné jevy. Všechna měření poskytovala velice rozdílná data pro všechny zdroje. Výhodou také je, že poskyují i více měřitelných veličin, které lze kombinovat a tím entropii ještě zvýšit.

Tabulka shrnující všechna měření a dosažené hodnoty entropie pro jednotlivé zdroje:

Zdroj entropie	**Dosáhnutá entropie [Sh/b]**
Otáčky disku/ventilátoru	0,062
Datum a čas	0,092
Teplota CPU a šum odporu	0,058
Vytížení CPU	0,097
Klávesnice a myš - náhodné pohyby	0,324
Klávesnice a myš - stejné pohyby	0,049
Obsah disků	0,081
Množství volné paměti	0,103
Podrobnosti souboru Temp	0,107
Šum na vstupu zvukové karty - klid	0,112
Šum na vstupu zvukové karty - ruch	0,346
Šum na vstupu zvukové karty - kombinace	0,544
Pakety na vstupu síťové karty	0,823
Pixely na monitoru	0,971

Tab. 3.1: Shrnutí jednotlivých měřeních.

4 ZÁKLADNÍ OPERACE PŘI KONSTRUKCI ENTROPICKÝCH GENERÁTORŮ

Pro konstrukci entropických generátorů je jedním z rozhodujících faktorů výběr vhodného zdroje šumu nebo náhodných dat, které bude generátor dále zpracovávat. Dalším neméně důležitým faktorem je i volba vhodného kryptografického algoritmu, který zajistí, že vygenerovaná sekvence bude skutečně náhodná a bude bezpečné ji použít například pro tvorbu kryptografických klíčů. Další kryptografický algoritmus by měl také sloužit k ochraně samotného generátoru proti možným pokusům o jeho napadení, či ochraně jeho interního stavu nebo semínka.

4.1 Výběr vhodného zdroje šumu nebo náhodných dat

Kvalitní a spolehlivý generátor pseudonáhodné posloupnosti, jenž využívá nějakého zdroje šumu nebo náhodných dat, je základem každého entropického generátoru. Jeho volba je tedy kriticky důležitá, protože musí splňovat celou řadu žádaných vlastností. Naopak slabin by měl mít pokud možno co nejméně. GNPN by měl splňovat především:

- Vysoká rychlost generovaných bitů
- Vysoká entropie
- Minimální periodicita[1]

Dále je při výběru naopak nutné zohlednit i několik faktorů, které by daný generátor rozhodně splňovat neměl:

- Perioda menší než 2^{64}
- Být obsahem knihoven pro programovací jazyky
- Základem je lineární korgurentní generátor
- Nenáhodná posloupnost bitů nejnižších řádů

Jako příklad generátoru náhodných čísel z knihoven pro programovací jazyky může být široce rozšířená funkce **rand()**. Generátory z těchto knihoven nemají standardizovanou implementaci a o jejich nevhodnosti jsem se přesvědčil i při práci na bakalářské práci, kdy jsem potřeboval generovat čísla o velikosti až 1024 bitů, a používání této funkce, jako jedné z pomocných v celém algoritmu, nepřinášelo vhodné

výsledky. O nevhodnosti použití lineárního korgurentního generátoru jsem se již zmiňoval výše, tento generátor má navíc i jednu další z nevhodných vlastností a tou je nenáhodná posloupnost bitů nižšího řádu. Použití takového generátoru jako GPNP pro kvalitní entropický generátor je tedy zcela nevhodné.[8]

Jako vhodný zdroj se naopak jeví využití například zvukové karty. Ta je v drtivé většině integrovaná na všech základních deskách do osobních počítačů a je nedílnou součástí každého notebooku. Další výhodou generátoru, který bude využívat jako zdroj entropie zvukou kartu, bude samostatnost. Nebude třeba, aby uživatel vkládal nějaký vstup, ze kterého by se vytvořilo inicializační semínko.

Jako další vhodný zdroj šumu může být využito náhodných bitů na vstupu síťové karty. Stejně jako zvuková karta je i síťová karta integrována na všech základních deskách, kterými se osazují osobní počítače a notebooky. Potenciál pro vhodný zdroj etropie je zde poměrně velký, protože v současnosti je automatický předpoklad, že PC bude připojen k internetu. I když uživatel nebude přímo sledovat internetové stránky, několik aplikací bude přes různé porty komunikovat. Například Microsoft Windows Upadete/Firewall, Antivirový program nebo další komunikační programy, které může využívat přímo uživatel jako je Skype, ICQ, Hamachi apod. Na vstupu síťové karty se tím pádem objevuje velké množství bitů, které lze využít jako zdroj entropie. Stejně jako řešení se zvukovou kartou má i toto řešení výhodu naprosté nezávislosti na vstupu od uživatele.

4.1.1 Volba vhodné konstrukce zdroje

Protože zdroje šumu produkují pouze surová data, které by generátor jen obtížně zpracovával, je nutné do zdoje zahrnout ještě další prvky, které zajistí požadovanou kvalitu a entropii bitového toku, který bude ze zdroje předáván na vstup generátoru. Mezi tyto prvky patří:

- Digitalizace
- Stabilizační komponent[1]

Funkce prvního jmenovaného je jasná. Pokud zdroj šumu neposkytuje přímo binární data, je třeba je digitalizovat a převést do vhodné podoby, kterou budeme dále využívat. Pokud již zdroj poskytuje digitální vzorky, může tento komponent sloužit pro další úpravu vzorku například pomocí von Neumannova korektoru, který pomáhá odstranit statistické závislosti. Jeho principem je, že zpracovává celý proud bitů jako proud bez překrývajících se dvojic po sobě jdoucích bitů a generuje z něj výstupy následovně:[11]

(1) Je-li na vstupu "**00**"nebo "**11**", celý vstup je zahozen (žádný výstup),
(2) V případě, že na vstupu je "**01**"nebo "**10**", tak výstupem je vždy první bit.

Druhým důležitým komponentem, který doporučení podle NIST zahrnuje je tzv. stabilizační komponent. Ten slouží ke zvýšení kvality a množství entropie, kterou zdroj poskytuje, pomocí vhodného kryptografického algoritmu. Ty jsou rozděleny na dvě skupiny, a to na schválené algoritmy a neschválené.

Doporučení NIST umožňuje implementaci i neschválených kryptografických algoritmů, ale ty nezaručují, že zdroj bude garantovat úplnou entropii. Generátor s takovou implementací navíc musí projít intenzivním testováním, aby byla zaručena jeho správná funkcionalita a dostatečná kvalita poskytované entropie.[1]

Využití schválených algoritmů má ovšem nesporné výhody. Schválené funkce rozdělí rovnoměrně entropii po celém výstupu stabilizačního komponentu a poskytnou tak úplnou entropii. Navíc schválené algoritmy mají rozšířenou dokumenci pro implementaci a jejich využití usnadní práci vývojáře. Mezi schválené kryptografické algoritmy patří:

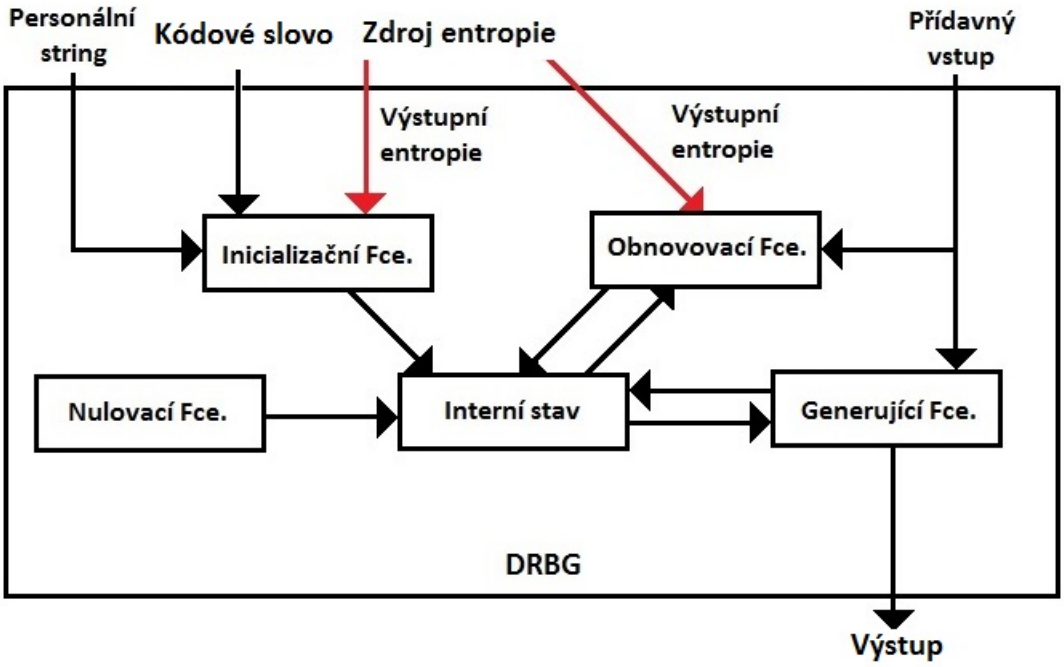
- **HMAC** se schválenou hashovací funkcí
- **CMAC** se schválenou blokovou šifrou
- **CBC-MAC** se schválenou blokovou šifrou
- Jakákoliv dle NIST schválená hashovací funkce
- **hash_df** specifikovaná v SP 800-90A
- **bc_df** specifikovaná v SP 800-90A se schválenou blokovou šifrou.[1]

Všechny tyto funkce jsou zdokumentovány v publikaci SP 800-90B.[2]

4.2 Výběr vhodného algoritmu pro DRBG mechanismus

DRBG mechanismus na obrázku 4.1 obsahuje několik funkcí, které zajišťují správný chod celého generátoru. Jedná se o tyto fuknce:

- Inicializační funkce
- Generující funkce
- Obnovovací funkce
- Nulovací funkce
- Testovací funkce stavu[2]



Obr. 4.1: Obecné schéma DRBG mechamismu

Inicializační funkce obdrží entropii ze vstupu (Entropy Input) a zajistí její skombinování s kódovým slovem (Nonce) a personálním stringem (Personalization String), jak je zobrazeno na obrázku 4.2. Takto vytvoří inicializační semínko, které bude sloužit pro inicializaci interního stavu generátoru.

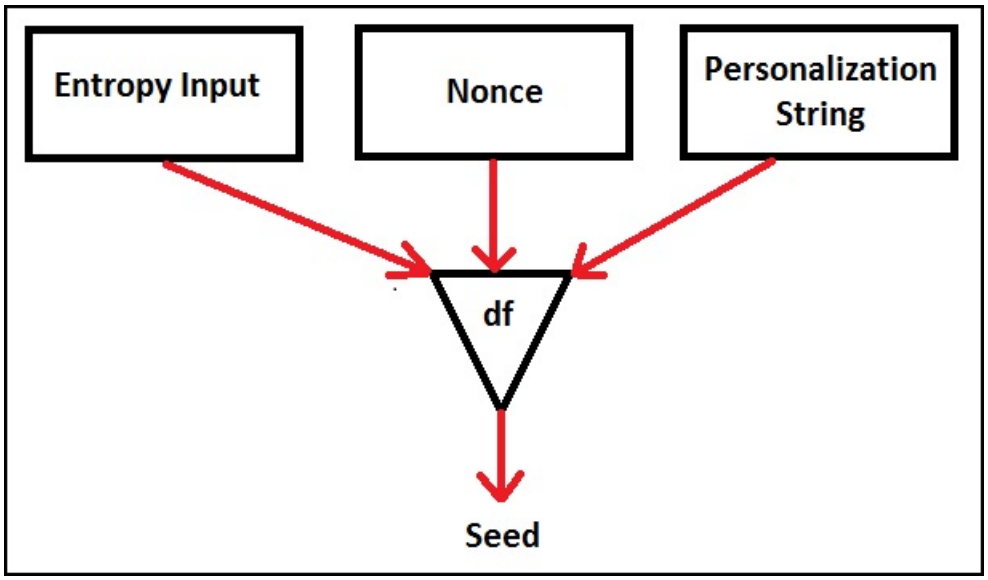
Entropy Input obsahuje výstup z nějakého vhodného zdroje entropie, jak bylo popsáno výše.

Nonce slouží pro zvýšení entropie a tím i kvality výsledného inicializačního semínka. Je několik možností implementace. První možnost spočívá ve využití stejného zdroje entropie jako pro *Entropy Input*, přičemž velikost kódového slova bude alespoň 1/2 bitů stringu, který bude na vstupu *Entropy Input*. Druhou možností jsou další náhodné hodnoty, u kterých bude zajištěno, že se nebudou opakovat s periodou menší než je velikost 1/2 bitů na vstupu *Entropy Input*.[2]

Personalization String je doporučen implementovat pro další zvýšení kvality entropie. Měl by být co nejunikátnější a obsahovat nějakou specifickou informaci. Jako vhodné řetězce pro použití jsou například:

- Sériová čísla zařízení
- Veřejné šifrovací klíče
- Identifikace uživatele počítače
- IP adresa v síti

- Datum a čas
- Náhodná čísla
- Identifikace konkrétní aplikace
- Náhodné, neodvoditelné stringy



Obr. 4.2: Konstrukce inicializačního semínka.

Generující funkce ověřuje vstupní parametry a na základě požadavku generuje pseudonáhodné bity s využítím aktuálního vnitřního stavu generátoru a po dokončení požadavku vygeneruje nový interní stav pro další požadavek. Právě pro Generující funkci je třeba zvolit vhodný a schválený kryptografický primitiv, který bude používat. NIST doporučuje několik možností:

- Hash_DRBG
- HMAC_DRBG
- CTR_DRBG
- Dual_EC_DRBG[2]

Hash_DRBG a *HMAC_DRBG* jsou založeny na použití jednosměrných hashovacích funkcí. NIST umožňuje použít všechny schválené hashovací funkce uvedené v tabulce 4.3.

V současnosti se už doporučuje spíše používaní hashů z rodiny SHA-2, nejběžněji používaný je právě SHA-256 pro svůj 256-bitový výstup a vyšší bezpečnost. Prakticky jeho jediná větší nevýhoda spočívá v tom, že není podporován na systémech s Microsoft Windows XP SP2 a nižších.

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Délka výstupního bloku v bitech	160	224	256	384	512
Maximální délka vstupní entropie	$\leq 2^{35}$ bitů				
Velikost Semínka pro Hash_DRBG	440	440	440	888	888
Maximální délka personálního stringu	$\leq 2^{35}$ bitů				
Maximální délka přídavného vstupu	$\leq 2^{35}$ bitů				
Maximální počet bitů pro jeden požadavek	$\leq 2^{19}$ bitů				
Maximální počet požadavků mezi obnovou semínek	$\leq 2^{48}$				

Obr. 4.3: Definice hashovacích funkcí pro DRBG mechanismus.

CRT_DRBG je založen na použití blokových šifer. Jako schválené blokové šifry podle NIST je možné použít TDEA a AES. Jejich bližší specifikace jsou níže v tabulce 4.4 :

	3 keys TDEA	AES-128	AES-192	AES-256
Délka výstupního bloku v bitech	64	128	128	128
Délka klíče	128	168	192	256
Délka semínka	232	256	320	384
Maximální velikost vstupní entropie	$\leq 2^{35}$ bitů			
Maximální délka personálního stringu	$\leq 2^{35}$ bitů			
Maximální délka přídavného vstupu	$\leq 2^{35}$ bitů			
Maximální počet bitů pro jeden požadavek	$\leq 2^{19}$ bitů			
Maximální počet požadavků mezi obnovou semínek	$\leq 2^{48}$			

Obr. 4.4: Definice blokových šifer pro DRBG mechanismus.

Stejně jako v případě hashovacích funkcí se již doporučuje používat alespoň AES-128, nejlépe však AES-256, který v současnosti patří k nejrozšířenějším.

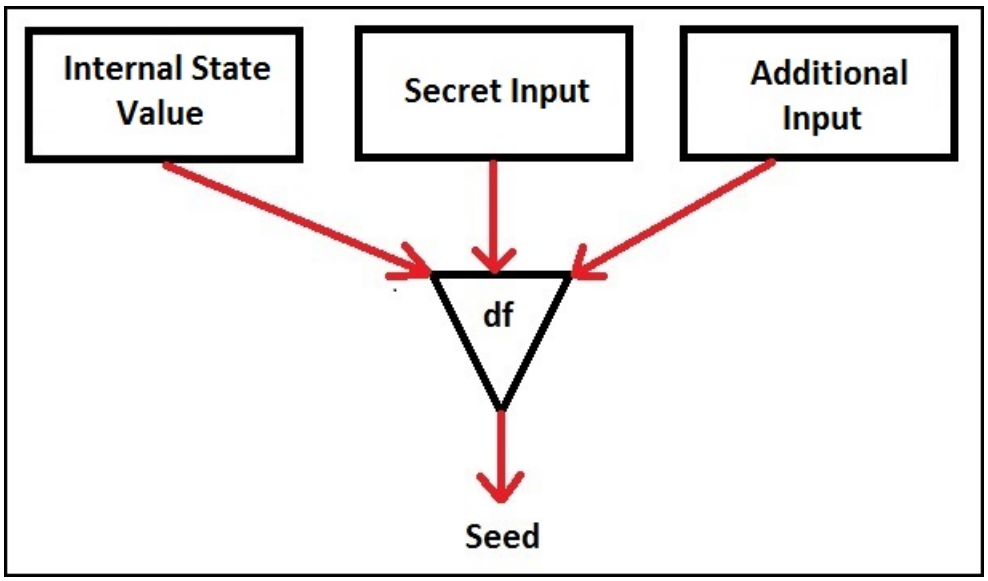
Dual_EC_DRBG jsou založeny na početních teoretických problémech jako je například problém diskrétního logaritmu. NIST doporučuje použití právě problému eliptické křivky diskrétního algoritmu. V tabulce 4.5 jsou uvedeny algoritmy, které by měly být použity při implementaci tohoto řešení.

Při použití alogritmu P-256 bych doporučil použití hashovacích fukncí z rodiny SHA-2, stejně jako v případě řešení *Hash_DRBG*. Implementace řešení *Dual_EC_DRBG* zaručí velkou bezpečnost celého systému, ale pro vývojaře je velmi náročné.

	P-256	P-384	P-521
Velikost základního pole v bitech	256	384	521
Veikost výstupního bloku v bitech	240	368	504
Vhodné hash funkce	SHA-1, SHA-2	SHA-1, SHA-2	SHA-1, SHA-2
Maximální velikost vstupní entropie	$\leq 2^{13}$ bitů		
Maximální délka personálního stringu	$\leq 2^{13}$ bitů		
Maximální délka přídavného vstupu	$\leq 2^{13}$ bitů		
Maximální počet bitů pro jeden požadavek	*max. velikost výstupu × ressed_interval*		
Maximální počet požadavků mezi obnovou semínek (ressed_interval)	$\leq 2^{32}$ bloků		

Obr. 4.5: Definice pro Dual_EC_DRBG mechanismus.

Obnovovací fuknce obrží novou entropii ze vstupu (Secret Input) a skombinuje ji s aktuálním vnitřním stavem (Internal State Value) a možným přídavným vstupem (Additional Input), pokud je implementován. Tímto vytvoří nové semínko pro generátor a vytvoří i nový interní stav, který bude použit pro generování.



Obr. 4.6: Obnovovací proces semínka.

Obnovovací funce je zavolána v případě, že byl vyčerpán maximální interval pro generování z aktuálního semínka nebo v případě, že bude zjištěna nedostatečná entropie aktuálního semínka.[2]

Secret Input je stejný zdroj entropie, který poskytuje entropii i pro inicializační funkci a je povinnou součástí obnovovacího procesu, stejně jako Internal State Value.

Additional Input je nepovinnou součástí a jeho implementace zcela záleží na vývojáři. Jeho použití se ale doporučuje, protože jednak zkvalitní nově vytvořené semínko i vnitřní stav a umožňuje jistou customizaci. Tím pádem pokud někdo bude chtít implementovat daný generátor do svojí aplikaci, který ho bude využívat, může navíc zvolit svůj vlastní přídavný vstup, který bude entropii a kvalitu semene ovlivňovat.[2]

Nulovací funkce slouží k vynulování interního stavu. Takový požadavek může vyvstat například z nějaké chyby, kterou návrátí testovací funkce, nebo nedostatečné kvality semínka, ze kterého byl interní stav vytvořen. Nulovací funkce je také vždy zavolána při zapnutí generátoru, aby zajistila, že interní stav bude vynulovaný a bude připravený pro nové semínko.

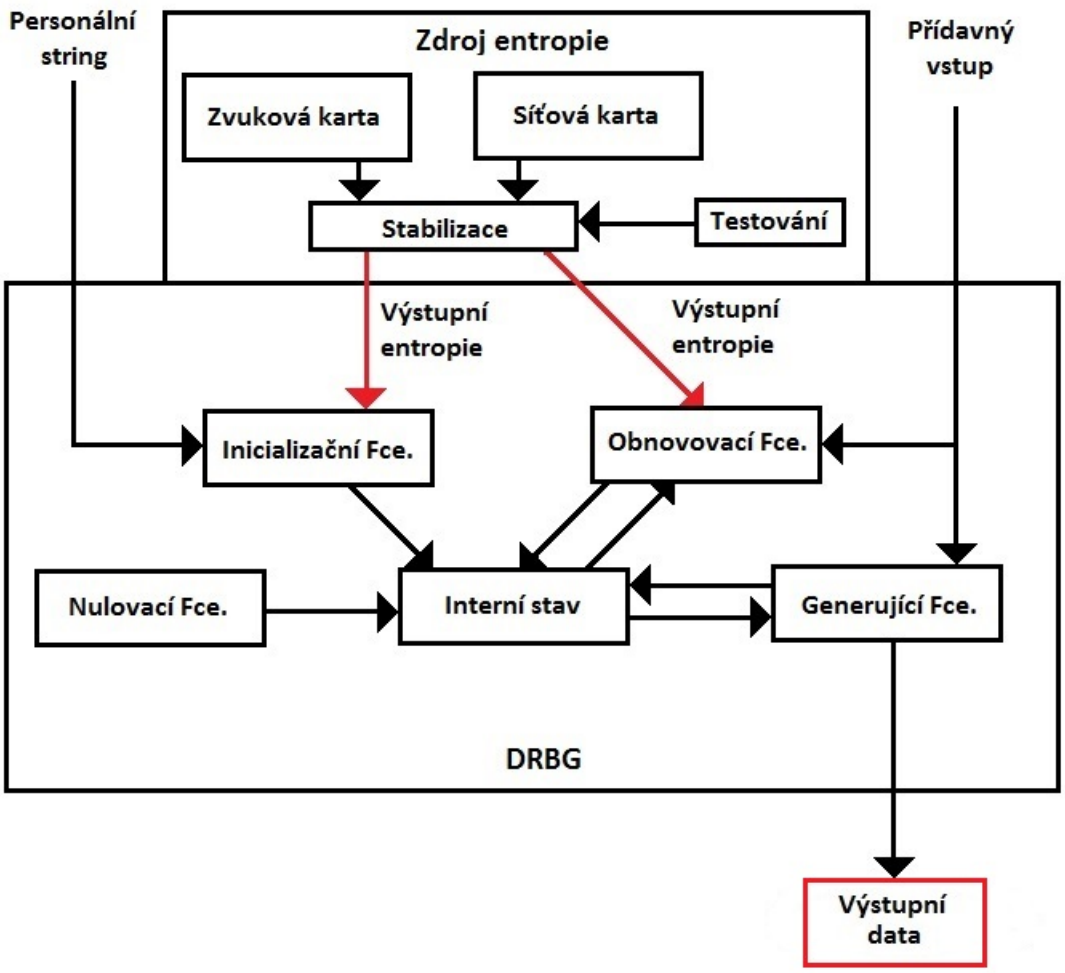
Testovací funkce stavu generátoru slouží, jak už z názvu vypovídá, k ověřování správného chodu generátoru a všech jeho částí a mechanismů. Každá část generátoru by měla mít funkci pro testování správné funkčnosti. Dle doporučení NIST by minimálně tyto funkce měly být testovány:

- Inicializační funkce
- Generující funkce
- Obnovovací funkce
- Nulovací funkce

Testovací funkce by měly být použity i při programování, aby se ověřila správná funkčnost každého bloku. K tomu se využívá technika tzv. předem známých výstupů, kdy pro dané vstupní parametry předem víme, jak bude vypadat výstup a můžeme ho porovnávat. Pokud bude platit nerovnost, víme, že v implementaci je někde chyba. Tyto funkce kontrolují především platnost vstupních parametrů a tzv. predikci, kdy vyhodnocují, jestli není možné danou hodnotu nějakým způsobem odhadnout. Pokud odhalí jakoukoliv odchylku, navracejí chybové statusy, které mohou navodit generování nového semínka nebo vnitřního stavu, případně zcela pozastavit generování náhodné posloupnosti.[2]

5 NÁVRH ŘEŠENÍ ENTROPICKÉHO GENERÁTORU

Návrh celého generátoru vychází z doporučeních NIST SP 800-90A[1] a NIST SP 800-90B[2]. Navržený entropický generátor je na obrázku 5.1. Blokové schéma se skládá ze dvou stěžejních částí, které spolu jako celek tvoří generátor. Nyní popíšu jeho jednotlivé části a vysvětlím, proč jsem se rozhodl je takto zvolit, a dále jejich výhody a nevýhody.



Obr. 5.1: Blokové schéma navrženého generátoru

5.1 Zdroj entropie

Spolehlivý zdroj entropie je základem každého kvalitního generátoru. Návržený generátor má spolehlivě pracovat na běžném osobním počítači. Ten nabízí několik možností, které lze využít jako zdroje entropie. Pro zdroj entropie doporučuje NIST použití alespoň dvou různých zdrojů šumu nebo náhodných dat. Já jsem se rozhodl využít zvukové a síťové karty.

5.1.1 Zvukové karta

Zvukovou kartu jsem zvolil z několika klíčových důvodů. Celý rozbor zvukové karty, jako možného zdroje etropie, jsem popsal v kapitole 3, teď pouze uvedu hlavní důvody, proč jsem se takto rozhodl:

- Velmi dobře zdokumentovaná implementace
- Spolehlivý zdroj náhodných dat
- Nevyžaduje vstup nebo aktivitu uživatele
- V současnosti téměř bez vyjímky integrována na všech základních deskách

Jistou nevýhodou může být fakt, že pokud bude počítač pracovat ve zvukově stabilním prostředí, tak může dojít k jistému stereotypu výstupním dat. Jelikož ale bude tento zdroj pracovat společně s druhým zdrojem náhodných dat, není nutné se touto nevýhodou znepokojovat. Pokud by přece jen tento jev měl nadmíru znehodnocovat výsledný tok bitů, implementoval bych Von Neumanův korektor, který by tuto nevýhodu značně potlačil.[11]

5.1.2 Síťová karta

Jako druhý zdroj náhodných dat jsem zvolil síťovou kartu. Podrobnější rozbor síťové karty je uveden v kapitole 3, tudíž opět uvedu pouze stěžejní důvody, které ovlivnily moji volbu právě pro tento zdroj náhodných dat:

- Velmi kvalitní zdroj náhodných dat
- Nevyžaduje vstup nebo aktivitu uživatele
- Standardní součást každého osobního počítače
- Vhodné pro implementaci právě do kombinace se zvukovou kartou
- Jednodušší implementace

Kombinace těchto dvou zdrojů náhodných dat se přímo nabízí. Obě dvě karty jsou téměř bez vyjímky intergrovány na všech základních deskách. Počítač připojený k internetu je naprostá nutnost už od Windows 8, které vyžadují připojení do sítě k dokončení instalace. Navíc generátor má být navržen pro běžný osobní počítač, nikoliv serverový. Lze tedy předpokládat, že běžný uživatel bude sledovat filmy, poslouchat muziku, prohlížet internetové stránky nebo stahovat data. Všechny tyto činnosti ovlivňují výstupní data, která budou z výstupu těchto karet generována a budou tak utvářet kvalitní materiál pro vytvoření inicializačního semínka. Další výhodou této kombinace je i fakt, že pokud bude uživatel například pouze zpracovávat dokumenty a bude ve zvukově statisticky stabilním prostředí, počítač připojený k internetu bude stále generovat dostatečně kvalitní náhodná data, ať už prostřednictvím Windows Update, antivirovým programem, nebo stahováním na pozadí. Elementů, které generují provoz na síťové kartě je celá řada, aniž by uživatel přímo využíval internetového prohlížeče.

5.1.3 Stabilizace

Komponent stabilizace, který slouží ke zkvalitnění entropie a potlačení nežádoucího biasu, je možné podle doporučení NIST SP 800-90B[2] realizovat pomocí schválených i neschválených algoritmů.

Ve svém návrhu jsem se rozhodl využít jedné z doporučených funkch uvedených v kapitole 4.1.1:

- **HMAC** se schválenou hashovací funkcí
- **CMAC** se schválenou blokovou šifrou
- **CBC-MAC** se schválenou blokovou šifrou
- Jakákoliv dle NIST schválená hashovací funkce
- **hash_df** specifikovaná v SP 800-90A[1]
- **bc_df** specifikovaná v SP 800-90A[1] se schválenou blokovou šifrou.

Rozhodl jsem se využít funkce **hash_df** z několika důvodů. Její implementace je velmi dobře popsaná a budu ji využívat i později v návrhu DRBG mechanismu generátoru. Hashovací funkce jsou navíc až dvojnásobně rychlé oproti blokovým šifrám, a tak je jejich využití ideální. Jako hashovací funkci jsem zvolil SHA-256 pro zajištění požadované bezpečnosti a především pro výstupní délku řetězce 256bitů, kterou bude požadovat DRBG mechanismus generátoru.

Využítí jedné ze schválených funkcí navíc zajistí, že bude zdroj garantovat plnou entropii, aniž by musel procházet dodatečným testování, jak je tomu nutné v případech, kdy se vývojář rozhodne využít neschválený kryptografický primitiv.

Stabilizační komponent díky hashovací funkci zajistí, že i při naprosto minimální změně náhodných dat, které obrží ze zvukové a síťové karty, bude výstup zcela odlišný od předchozí hodnoty.[1]

5.1.4 Testování

Testovací funkce lze rozdělit na dvě stěžejní skupiny.

- Testování stabilizačního komponentu
- Testování výstupních dat během generování

Při spuštění generátoru, který si začne žádat data ze zdroje entropie je nutné prvně otestovat stabilizační komponent, jestli pracuje správně. Testování stabilizačního komponentu bude probíhat na základě předem známé hodnoty výstupu. Tzn. že testovací funkce mu předá takové vstupní parametry, pro které bude znát správnou výstupní hodnotu. Pokud jí komponent tuto hodnotu navrátí, pracuje správně, v opačném případě funkce vrátí chybu.

Dále se musí testovat výstupní data, která bude zdroj předávat do DRBG mechanismu. K tomu budou sloužit dva různé testy:

- Test počtu opakování
- Test detekce ztráty entropie

Test počtu opakování zaručuje, že na výstupu zdroje nebude stále stejná veličina. V případě, že by se zdroje náhodných dat buď zasekly nebo vlivem těžko předvídatelných okolností po nějakou dobu dodávaly pouze jedna a ta stejná data, tento test to odhalí. Pokud k této situaci dojde, test přeruší dodávání dat do DRBG mechanismu do té doby, než budou zase data v pořádku.[1]

Tento test může být náročný na využití paměti, proto je jeho vhodná implementace velmi důležitá. Nesmí celý generátor příliš zpomalovat a nesmí využívat příliš mnoho paměti. Během generování se všechna data ukládají do datasheetu a vyhodnocují. Posuzují se dvě základní proměnné a jedna konstanta **C**:

- **A** = nejčastěji opakovaná hodnota
- **B** = počet, kolikrát se hodnota A vyskytla
- **C** = hodnota, při jejímž překročení test navrátí chybu

Během testu jsou hodnoty A, B a C ukládány do paměti, proto je potřeba její dostatek a je vhodné omezit maximální možnou velikost datasheetu, aby nedocha-

zelo k jejímu nadměrnému využívání. Test lze popsat následujícím vzorcem:

$$C = 1 + \left[\frac{(-log(W))}{H} \right] \geq 2 \tag{5.1}$$

Tento vzorec platí pro hodnty $W > 0$. Hodnota konstanty C je nejmenší celé číslo, které splňuje tuto nerovnost. Hodnoty W a C se musí vhodně zvolit, aby nebylo testování příliš přísné ani příliž benevolentní. Přesné hodnoty budou stanoveny během implementace a testování.[1]

Test detekce ztráty entropie, je navržen pro detekci velkých ztrát entropie, protože první zmiňovaný test zkoumá pouze, zda-li se neopakují příliš často stejné hodnoty na výstupu, ale samotnou kvalitu entropie neposuzuje. Tento test také běží neustále za běhu generátoru a udržuje si tři proměnné veličiny a tři konstaty:[1]

- **A** = hodnota aktuálního zkoumaného vzorku
- **S** = celkový počet vzorků zkoumaný od začátku testu
- **B** = počet opakování vzorku A v celkovém počtu vzorků S
- **N** = celkový počet vzorků, které je nutné obdržet pro spuštění testu
- **W** = pravděpodobnost pro false posivite, podle NIST $W = 2^{-30}$
- **C** = Mezní hodnota po jejímž překročení dojde k selhání testu

Určení parametru N

Známého také jako *velikost okna* se podle doporučení NIST omezuje na použití těchto čtyř hodnot: 64, 256, 4096 a 65536. Měly by běžet paralelně alespoň dva tyto testy, každý pro jinou velikost okna. Vhodné hodnoty je pro každý zdroj náhodných dat nutné správně určit, protože například test pro velikost okna 65536 vzorků může detekovat i velmi malé ztráty entropie, zatímco okno o velikosti 16 vzorků bude detekovat pouze obrovské ztráty.

NIST doporučuje vždy použít okno o velikosti 4096 vzorků, protože splňuje minimální požadovanou hranici ztráty entropie, kterou by měla testovací funkce odhalit. Ta byla stanovena na 30%. Jako druhou velikost okna jsem se rozhodl zvolit 256 vzorků. Tabulka 5.1 zobrazuje odhadovanou schopnost detekce ztráty entropie pro jednotlivé velikosti oken, kdy H udává počet bitů entropie.[1]

Protože mnou navržený zdroj zahrnuje i stabilizační komponent, postačuje jako druhá velikost testovacího okna 256 vzorků. Funkce **hash_df** bude zajišťovat velmi

H	Velikost okna			
	64	256	4096	65536
1	67%	39%	13%	3%
2	56%	32%	11%	3%
4	100%	31%	12%	3%
8	54%	40%	19%	6%
16	69%	56%	40%	22%

Tab. 5.1: Detekovatelná ztráta entropie pro jednotlivé velikosti oken

kvalitní změnu výstupní entropie, takže není nutné testovat i tak malou změnu entropie, jako jsou 3% pro velikost okna 65536.

Určení parametru C

Při překročení této hodnoty dojde k selhání testu a pozastavení předávání dat na výstup zdroje. Hodnota C se určuje podle minimální entropie pro jeden vzorek H, velikosti okna N a přijatelnou false positive hranice W. Vzorec pro výpočet této hodnoty se používá funkce *CRITBINOM()*, která je dostupná například v Microsoft Excel. Příklad vzorce pro Microsoft Excel:

$$= CRITBINOM(N, 2\hat{}(-H), 1, -T) \tag{5.2}$$

Tabulka 5.2 znázorňuje hraniční hodnoty pro jednotlivé velikosti oken podle počtu bitů entropie. Přesnou hodnotu C určím během implementace zdroje, při testování nejvhodnějšího nastavení a správné funkčnosti těchto testovacích funkcí.

Správnou funčknost všech komponentů zdroje budu ověřovat v testovacím rozhraní. Samotné testovací rozhraní budu využívat pouze v rámci implementace a ladění, ve finální fázi generátoru nebude dostupné, testovat se budou až výstupní data.

Pouze data, která úspěšně absolvují všechny tyto testy, budou předána na výstup zdroje, ze kterého, na základě požadavku, budou dále předána do DRBG mechanismu. Tento návrh zaručuje, že všechna data, která generátor využije k vytvoření inicializačního semínka, budou mít vysokou hodnotu entropie.

N	64	256	4096	65536
H	Cutoff	Cutoff	Cutoff	Cutoff
1	51	168	2240	33537
2	35	100	1193	17053
3	24	61	643	8705
4	16	38	354	4473
5	12	25	200	2321
6	9	17	117	1220
7	7	15	71	653
8	5	9	45	358
9	4	7	30	202
10	4	5	21	118
11	3	4	15	71
12	3	4	11	45
13	2	3	9	30
14	2	3	7	21
15	2	2	6	15
16	2	2	5	11
17	1	2	4	9
18	1	2	4	7
19	1	1	3	6
20	1	1	3	5

Tab. 5.2: Tabulka hraničních hodnot C pro jednotlivé velikosti oken

5.2 Personální string

Doporučení NIST silně doporučuje použití personálního stringu, proto jsem se ho rozhodl do implementace zahrnout také. Bude spolu se zdrojem entropie a kódovým slovem tvořit základ pro kvalitní inicializační semínko. Aby byl co nejvíce unikátní měl by obsahovat nějakou specifickou informaci. Vhodné unikátní řetězce, které jsem uvažoval:

- Sériové číslo konkrétního zařízení
- Veřejné šifrovací klíče uživatele
- IP adresa v síti
- Identifikace uživatele počítače
- Identifikace konkrétní aplikace
- Náhodné, neodvoditelné stringy
- Datum a čas

Nakonec jsem se rozhodl využít kombinaci stringů **IP adresy počítače** v síti a **Identifikaci uživatele**. Doporučení NIST uvádějí, že by personální string měl být tajná informace, aby ji útočník nemohl zneužít. **Datum a čas** jsem využít nechtěl, protože tím by se personální string na různých počítačích podobal - záleželo by jen na konkrétním čase.[1]

Implementace **Náhodných/neodvoditelných stringů** by vyžadovala použití buď konkrétního souboru, který by tyto stringy obsahoval, nebo přímo v programu velkou databázi různých stringů. Obojí je značně nevýhodné - soubor nemusí být přenositelný mezi systémy Windows / Unix a zabírá další místo na disku. Databáze stringů přímo v programu by zase zbytečně zabrala velký kus paměti. Navíc pro obě možnosti platí stejná nevýhoda, jako u času, tzn. že by na všech počítačích personální stringy byly vybírány ze stejného seznamu a nebyly by tím pádem tolik unikátní.

Identifikace konkrétní aplikace zní jako dobrý nápad. Ale ta v mém návrhu bude už realizovaná pomocí přídavného vstupu, takže je zbytečné, aby v tomto případě byla ještě i na místě personálního stringu. Pokud by se ale vývojář rozhodl přídavný vstup nezařazovat, byla by tato možnost určitě dobrá volba.

Sériové číslo konkrétního zařízení je poměrně složité na implementaci, protože každý počítač se může skládat z kompomentů rozdílných výrobců apod., odladit tedy funkci, která by vždy vrátila správnou, použitelnou hodnotu, by bylo pro tento návrh zbytečně složité.

Veřejný šifrovací klíč uživatele splňuje požadavek na unikátnost a poskytuje vysokou entropii. Nevýhoda je ovšem v tom, že je veřejný. Tím pádem by mohl do generátoru vnést jistou slabinu, protože pokud by útočník znal veřejný klíč uživatele, který generátor používá, mohl by ho zneužít k pokusům o odvození výstupních náhodných dat z generátoru.

Použití **Identifikace uživatele** na konkrétním počítači se tedy jeví jako vhodná volba. Pro každý počítač bude tento řetězec jiný, samozřejmě najdou se i shodné, ale bude jich markantně menší množství, než v případě využití například času. Navíc pokud by útočník dokázal prolomit ochrané mechanismy DRBG generátoru, název počítače mu bez hesla bude téměř k ničemu.

IP adresa počítače je další vhodná volba. IPv4 adres je obrovské množství a po přechodu na protokol IPv6, který je nevyhnutelností, jich bude ještě mnohem více. V dnešní době je navíc hojně využívané tzv. natování adres, takže možností je opravdu velké množství, a pokud bude tato informace skombinována s identifikací počítače daného uživatele, dostaneme velmi unikátní string pro každou počítačovou stanici.

5.3 Kódové slovo

Kódové slovo je stejně jako personální string nutné implementovat především kvůli bezpečnosti, ale i zkvalitnění entropie pro inicializační semínko. Podle doporučení NIST SP 800-90A[1] jsou dvě možnosti, jak jej realizovat:

- Jako hodnotu, která musí mít velikost alespoň 1/2bitů, jakou má požadovaná celková velikost bitů entropie ze zdroje.
- Jako hodnotu, u které bude zajištěno, že se nebude opakovat častěji, než je 1/2 bitů z celkové velikosti bitů vyžadované ze zdroje entropie.

Rozhodl jsem se kódové slovo realizovat, jak je popsáno v prvním případě. Je to především z důvodu jednodušší implementace, protože bude obdrženo ze stejného zdroje entropie, jako primární vstup entropie. To znamená, že nebude třeba použít další funkci, pro obdržení kódového slova, ale využije se jedné společné funkce **GetEntropyInput(min_entropy, min_length, max_length)**, která bude mít parametry pouze navýšené o požadavou minimální velikost kódového slova. Tzn, že všechny parametry budou navýšeny alespoň o 1/2 polovinu celkové velikosti. Samotnou fukci rozberu podrobněji níže v popisu implementace DRBG mechanismu.

5.4 Přídavný vstup

Konkrétní přídavný vstup nebude v tomto návrhu přímo implementován. Uživatelé, kteří by chtěli využívat tohoto generátoru, budou mít možnost přidat ještě vlastní vstup, pomocí nachystané funkce. Funkce bude vypadat následovně:

additional_input = getAdditionalInput(security_strength, min_length, max_length)

Parametry má stejné jako funkce **GetEntropyInput**, která bude rozebraná později. Parametry jsou nastavené tak, aby se v základním nastavení nijak nepodílely na funkčnosti generátoru. Pokud bude daný uživatel chtít přidat do návrhu vlastní vstup, jednoduše modifikuje tuto funkci, aby její parametry ovlivnily výsledný **additional_input**, a tím i obnovení semínka.

5.5 DRBG mechanismus

Spolehlivý DRBG mechanismus je základem kvalitního generátoru náhodných veličin. Obsahuje celou řadu funkcí, které zajišťují jeho bezpečnost a spolehlivost. NIST

doporučuje použití několika algoritmů, které splňují tyto požadavky a na jejich základě pak pracují všechny další potřebné funkce. Tyto algoritmy lze rozdělit na tři základní skupiny:[1]

- DRBG mechanismy založené na haskovacích funkcích - **hash_df** a **HMAC**
- DRBG mechanismy založené na blokových šifrách - **CTR_DRBG**
- DRBG mechanismy založené na početních problémech - **Dual_EC_DRBG**

Rozhodl jsem se využít **hash_df** algoritmu a to z několika důvodů:

- Nejméně problematická implementace ze všech mechanismů.
- Hashovací funkce jsou ověřené, bezpečné a při použití funkce SHA-256 bude zajištěna vhodná délka výstupu o síle 256 bitů.
- Používání SHA-256 je velmi rozšířené v praxi a dobře zdokumentované.
- **hash_df** bude využito už ve stabilizačním komponentu zdroje náhodných dat, takže bude zaručena dobrá kompatibilita. NIST navíc doporučuje použití stejného kryptografického primitivu pro všechny části generátoru.

Další výhodou tohoto řešení je fakt, že bude plně potlačena **Backtrack resistance** (odolnost proti zpětnému odvození), protože **hash_df** využívá hashovací funkce a ty jsou jednosměrné. V implementaci tudíž nebude zahrnut parametr, který kontroluje tento stav. Navíc pro prolomení SHA-2 v současné době není známý žádný útok nebo slabina, takže bude zajištěná bezpečnost celého systému. Nyní popíšu všechny části a funkce DRBG mechanismu, které budou založené na zvoleném algoritmu.

Bezpečnostní sílu generátoru **(Security strenght)** jsem se rozhodl zvolit pevně 256. Tento počet bitů je v současné době téměř nemožné prolomit a používá ho drtivá většina doporučených bezpečnostních standardů/funkcí. Díky tomu, že bude tento parametr nastavený napevno, dojde ke zjednodušení implementace, protože z inicializační funce dojde k vyjmutí tohoto parametru a jeho ověřování.

Dále je třeba zvolit minimální akceptovatelnou délku bitů entropie **min_entropy**, se kterou bude generátor pracovat. Moderní generátory jsou často navržené tak, že **min_entropy = min_length** parametr. Rozhodl jsem se zvolit tuhle možnost, protože tím opět dojde ke zjednodušení implementace a nevnese to do celého mechanismu žádnou slabinu.

5.5.1 Inicializace generátoru

Samotná inicializace generátoru obsahuje několik funkcí, které zajištují potřebné hodnoty pro vytvoření inicializačního semínka a vnitřního stavu generátoru. Během inicializace se testují všechny vstupní parametry jednotlivých funkcí.

Inicializační funkce se stará o obdržení personálního stringu a požadovaného počtu bitů entropie ze zdroje. V případě implementace, kdy velikost požadované síly bezpečnosti entropie **(security_strenght)** ze zdroje je volitelná nebo není zajištěna **prediction_pesistance** obnovováním semínka a vnitřního stavu pomocí obnovovací funkce, uvažují se ještě tyto dva další parametry.[1]

InstantiateFunction(personalization_string)

Další funkcí, která se stará o obdržení entropie ze zdroje a kódového slova je **GetEntropyInput**. Protože jsem se rozhodl implementovat kódové slovo jako náhodnou veličinu, která bude pocházet ze stejného zdroje jako hlavní vstup, lze použít pouze jedno volání této funkce, pro obržení obou veličin. Pouze dojde k navýšení hodnot parametrů o 1/2 velikosti. Status pouze navrací buď **SUCCESS**, při úspěšném vytvoření entropy_input nebo **ERROR**, pokud při průběhu funkce nastala nějaká chyba.

(status, entropy_input) = GetEntropyInput(min_length, max_length)

Jamile proběhne úspěšně vykonání funkcí InstantiateFunction a GetEntropyInput, budou k dispozici všechny potřebné hodnoty pro vytvoření semínka a interního stavu. K tomuto slouží **InstantiateAlgorithm**.

initial_working_state = InstantiateAlgorithm (entropy_input, personalization_string

Výstupní proměnná **initial_working_state** bude obsahovat všechny potřebné hodnoty interního stavu tzv. **seed_material**, se kterým bude dále pracovat zvolený **hash_df** mechanismus. Jedná se o tyto hodnoty:[1]

- hodnota **seed_material**, která vznikne jako *seed_material = entropy_input || nonce || personalization_string*.
- hodnota **reseed_counter**, která představuje celé číslo, po jehož vyčerpání bude zavolána funkce pro obnovení semínka.

5.5.2 Interní stav

Interní stav generátoru **working_state** je možné implementovat dvěma způsoby. Buď je umožněno používání více interních stavů zároveň nebo se zachovává pouze jeden stav, který se vždy změní po použití generujícího algoritmu nebo obnovovací funkce. V mém řešení jsem se rozhodl využít pouze jeden interní stav. Ten uchovává tyto tři hodnoty, které později využívají **Generate_algorithm** a **Reseed_algorithm**:[1]

- hodnotu **V** - inicializační semínko (seed)
- hodnotu **C** - hodnota po aplikaci **hash_df** funkce
- hodnota **reseed_counter** - počítadlo

Proces vytváření hodnot **V**, **C** a **reseed_counter** je následující:

1. *seed* = **Hash_df**(*seed_material, seedlen*)
2. **V** = *seed*.
3. **C** = **Hash_df**((0x00 || *V*), *seedlen*)
4. *reseed_counter = 1*
5. Return **V**, **C** a **reseed_counter** jako **working_state**.

5.5.3 Obnova semínka

Tato funkce slouží k obnově semínka a interního stavu, které již vyčerpalo svůj interval pro generování náhodných čísel. Tento interval jsem podle doporučení NIST stanovil na **100 000** cyklů. Tato funkce není podle doporučení NIST povinná pro implementaci, ale já se ji rozhodl využít, protože zajišťuje potlačení **prediction_resistance**. Tento fakt vede k zjednodušení implementace ostatních funkcí, protože je z nich tento parametr vynechán a samotný DRBG mechanismus zajišťuje větší bezpečnost. Pokud nastane během obnovovacího procesu chyba, program navrátí textovou zprávu o této chybě. Pro obdržení dat z přídavného vstupu (additional_input) bude použita funkce **ReseedFunction**:

ReseedFunction(additional_input)

Protože pro obnovu semínka se používá entropie ze stejného zdroje jako pro inicializační semínko, můžeme využít již popsanou funkci GetEntropyInput pro získání této entropie:

(status, entropy_input) = GetEntropyInput(min_length, max_length)

Tímto budou k dispozici všechny potřebné hodnoty pro vygenerování nového semínka a nového vnitřního stavu generátoru. Pro tento proces slouží **ReseedAlgorithm**:[1]

new_working_state = ReseedAlgorithm (working_state, entropy_input, additional_input)

Výstupní proměnná **new_working_state** obsahuje nové hodnoty **V**, **C** a **ressed_counter**, které jsou uloženy jako nový vnitřní stav generátoru pro generování dalších pseudonáhodných posloupností. Proces pro výpočet nových hodnot je následující:

1. seed_material = 0x01 || **V** || entropy_input || additional_input.
2. seed = **Hash_df** (*seed_material*, *seedlen*).
3. **V** = seed.
4. **C** = **Hash_df** ((0x00 || V), *seedlen*).
5. reseed_counter = 1.
6. Return **V**, **C**, and **reseed_counter** jako **new_working_state**.

5.5.4 Nulovací funkce

Tato funkce slouží pouze k vynulování vnitřního stavu, buď na základě požadavku od generátoru, například při zapnutí, nebo při detekci nějaké chyby pomocí testovacích funkcí. Konstrukce funce:

status = UninstantiateFunction(state_handle)

Funkce obsahuje pouze jediný parametr a to state_handle, který je ukazatelem na vnitřní stav, který má být vynulovaný. Funkce pouze navrátí status, který bude obsahovat SUCCES při úspěšném vynulování vnitřního stavu nebo ERROR v případě neúspěšné operace.[1]

5.5.5 Generování pseudonáhodné posloupnosti

Pokud všechny předešlé funkce a algoritmy proběhnou v pořádku, je k dispozici kvalitní semínko a interní stav generátoru. Tímto je možné přejít už k samotnému

generování konečné, výstupní, pseudonáhodné posloupnosti. K tomu slouží **GenerateFunction** a **GenerateAlgorithm**. První zmíněná fuknce slouží k poslednímu ověřování všech důležitých parametrů, ale protože můj návrh zaručuje potlačení resistancí, security_strength má pevnou hodnotu a generátor využívá jednoho interního stavu, téměř všechny parametry jsou vynechány. Původní funkce:[1]

GenerateFunction(state_handle, requested_number_of_bits, requested_security_strength, prediction_resistance_request, additional_input)

Parametry, které zůstávají, jsou pouze **requested_number_of_bits** a **additional_input**. První zmíněný představuje celé číslo, které značí požadovanou délku řetězce výstupních bitů. Druhý představuje data z přídavného vstupu, který v mé implementaci bude představovat pouze prázdnou fuknci, kterou bude možné customizovat, podle požadavků každého vývojáře zvlášť. Funkce po vynechání nepotřebných parametrů:

GenerateFunction(requested_number_of_bits, additional_input)

Tímto získáme hodnotu posledního parametru, pro **GenerateAlgorithm**. Výstupem této funkce budou pseudonáhodné bity, které budou ukládány do textového souboru. Po vygenerování požadovaného počtu náhodných sekvencí bude výstupní soubor podroben testům, popisovaným v kapitole 2.4.

(status, pseudorandom_bits, new_working_state) = GenerateAlgorithm (working_state, requested_number_of_bits)

Pokud funkce proběhne úspěšně, bude navrácen status **SUCCES**. V opačném případě bude navrácena hodnota **ERROR**. Při úspěšném proběhnutí funkce bude navíc navrácen nový vnitřní stav a generátor bude připravený na další požadavek pro generování. Proces pro generování výstupních bitů a nového vnitřního stavu:[1]

1. If $reseed\_counter > reseed\_interval$ - návrat požadavku pro obnovení semínka
2. If ($additional\_input \neq Null$), then do
2.1 w = **Hash** (0x02 $||$ V $||$ $additional\_input$).
2.2 V = $(V + w)$ mod $2^{seedlen}$.
3. ($returned\_bits$) = **Hashgen** ($requested\_number\_of\_bits$, V).
4. H = **Hash** (0x03 $||$ V).
5. V = $(V + H + C + reseed\_counter)$ mod $2^{seedlen}$.
6. $reseed\_counter = reseed\_counter + 1$.

7. Return **SUCCESS**, **returned_bits**, and the new values of **V**, **C**, and **reseed_counter** for the new_working_state.

Hashgen (*requested_number_of_bits, V*):

1. $m = \left\lceil \frac{requested\_no\_of\_bits}{outlen} \right\rceil$
2. $data = V$.
3. $W =$ the *Null string*.
4. For $i = 1$ to m
4.1 $w_i =$ **Hash** (data).
4.2 $W = W \mid\mid w_i$.
4.3 $data = (data + 1) \bmod 2^{seedlen}$.
5. *returned_bits* = Leftmost (*requested_no_of_bits*) bits of W.
6. Return *returned_bits*.

Jak již bylo řečeno výše, **returned_bits** budou uloženy do textového souboru s příponou .txt pro další pohodlné zpracování a testování.

5.5.6 Testovací funkce

Testovací funkce jsou nutné pro ověření správně funkčnosti jednotlivých částí generátoru. Veškeré testy jsou založeny na předem známém výsledku, takže se funkcím předají vstupní parametry, pro které je znám výstup. Pokud některá z testovacích funkcí odhalí selhání dané části generátoru, bude navrácena chybová hláška. Všechny funkce proběhnou při zapnutí generátoru a ověří všechny jeho části. Až pokud všechny funkce navráhí status **SUCCES**, může generátor začít pracovat. Níže jsou uvedeny testované funkce a parametry, které budou staticky zadány a musí navrátit požadovaný, předem známý výsledek.

Testování inicializačních funkcí

Pevně dané parametry pro testování:

- entropy_input
- nonce
- personalization_string

Pokud výstupní hodnota a staticky zadaná honota budou rovné, bude navrácen status **SUCCES** a generátor může dál pracovat. V opačném případě bude navrácen stav **ERROR** a uživatel bude informován o selhání.[1]

Testování obnovovacích funkcí

Pevně dané parametry pro testování:

- entropy_input
- additional_input
- interna_state

Pokud výstupní hodnota a staticky zadaná honota budou rovné, bude navrácen status **SUCCES** a generátor může dál pracovat. V opačném případě bude navrácen stav **ERROR** a uživatel bude informován o selhání. [1]

Testování generujících funkcí

Pevně dané parametry pro testování:

- requested_no_of_bits
- additional_input
- internal_state
- ressed_counter

Testovací funkce také zahrunuje testování hodnoty ressed_counter, u které ověřuje, zdali generátor správně reaguje na překročený nebo rovný počet maximální hodnoty intervalu pro obnovení semínka. Pokud výstupní hodnota a staticky zadaná hodnota budou rovné a reakce na překročení ressed_counter bude správná, bude navrácen status **SUCCES** a generátor může dál pracovat. V opačném případě bude navrácen stav **ERROR** a uživatel bude informován o typu selhání.[1]

Testování nulovací funkce

Pevně dané parametry pro testování:

- internal_state

Pokud výstupní hodnota a staticky zadaná honota - v tomto případě nulová, budou rovné, bude navrácen status **SUCCES** a generátor může dál pracovat. V opačném případě bude navrácen stav **ERROR** a uživatel bude informován o selhání, ale generátor může dále pokračovat v generování, protože použití této funkce není zcela limitující pro generování kvalitní pseudonáhodné posloupnosti. Pouze v případě

chyby některé z dalších funkcí by bylo nutné celý generátor vypnout a znovu zapnout, protože nulovací funkce by nebyla schopná smazat již nevyhovující internal_state.[1]

5.6 Rozhraní generátoru

Rozhodl jsem se neimplementovat žádné konkrétní grafické rozhraní. Hlavním cílem této implementace je zajištění výstupních, kvalitních, pesudonáhodných bitů. Celý generátor bude tudíž navržen jako konzolová aplikace, na kterou budou přes standardní vstup zadávány příkazy a přes standardní výstup bude uživatel informován o chybách a případných dalších hlášeních.

Přiklonil jsem se k tomuto řešení i z toho důvodu, že additional_input bude otevřený pro custom implementaci. Každý, kdo se tedy rozhodne používat tento generátor, bude mít kvalitní a spolehlivé funkční jádro a grafické rozhraní, případně přídavný vstup si bude moci implementovat podle svých požadavků.[3]

5.7 Testování

Výsledný soubor jsem podrobil sérii 15-ti testů, probíraných v kapitole 2.4. Veškeré potřebné soubory a dokumentaci je možné stáhnout z adresy `http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html`.

5.7.1 Nastavení pro testování

Testování je vhodné provádět na uxinovém systému. Stáhnout je možné velké množství distribucí, ovšem doporučená je distribuce Ubuntu s compilerem GCC. Stáhnul jsem tedy nejnovější verzi a nainstaloval ji do virtuálního prostředí VMWARE, které pro účely testování bude naprosto dostatečné. Abych mohl začít s testováním, je nutné provést několik kroků:[7]

- Do pracovního adresáře s projektem je třeba nakopírovat stažený **sts-2.1.2.tar** soubor, což je nejnovější verze. Poté provést istrukci *tar -xvf sts-2.1.2.tar*
- Tím by se mělo rozbalit šest různých podsložek: *data, experiments, include, obj, src, templates a soubor makefile.*
- Podadresář *data* je určený pro RNG soubory, které chceme testovat. Jsou podporované dva formáty. První je formát ASCII, obsahující jedničky a nuly a druhý jsou binární datové soubory.
- Podadresář *experiments* je úložištěm emprických výsledků pro statické testy, to obsahuje další podsložky. Podsložka *AlgorithmTesting* je výchozí pro emprické výsledky pro RNG soubor v adresáři *data.*

- Podadresář *include* obsahuje hlavičkové soubory pro statické testy, pseudonáhodné generátory a související procedury.
- Podadresář *obj* obsahuje soubory pro jednotlivé statické testy, pseudonáhodné generátory a související procedury.
- Podadresář *src* obsahuje zdrojové kódy pro každý ze statických testů.
- Podadresář *templates* obsahuje sérii neperiodických šablon pro různé velikosti bloků, které jsou využívány ke statickému testu nepřekrývajících se šablon.¨
- Nastavení globálních parametrů testu
- Editování souboru *makefile*.
- Po spuštění souboru *makefile* se ve složce projektu vytvoří exe soubor *assess*.
- Testovaná data budou vyhodnocena po zadání následujícího příkazu: *assess 1000000*.[7]

Před testováním je možné ještě nastavit globální parametry pro test. Pokud je k dispozici dostatek paměti, neměl by software selhlat, ale protože testuji ve virtuálním prostředí, je vhodné tyto parametry zahrnout, abych tak předešel chybám. Globální parametry použité při testování:

Source Code Parameter	Default Parameter	Description/Definition
ALPHA	0.01	Significance Level
MAXNUMOFTEMPLATES	40	Non-overlapping Templates Test
NUMOFTESTS	16	Max number of tests
NUMOFGENERATORS	12	Max number of PRNGs

Obr. 5.2: Obnovovací funkce.

Při editaci souboru *makefile* je nutné upravit tyto řádky:

- **CC** (používaný ANSI C kompiler)
- **ROOTDIR** (kořenový adresář projektu)

Délka 1 000 000 bitů je doporučená minimální délka pro testování, aby nedošlo ke zkreslení testu. Můj generátor produkuje hexa čísla, takže po vygenerování dostatečného množství dat je už jednoduché je převést do binární podoby a vložit soubor do adresáře **data**.[7]

Jakmile je vše správně nastaveno, může se začít s testováním.

5.7.2 Vyhodnocení výsledků testu

Test skončil s následujícím výsledkem:

0 failed individual tests with THRESHOLD 0.001000 on 834 individual tests

Pokud by tedy některá z hodnot **p_value** měla hodnotu pod 0.001, daný test by skončil neúspěšně. Testovaný soubor ovšem prošel úspěšně všemi testy a generátorem vyprodukovaná data je tedy možné považovat za zcela náhodná.

Abych měl jistotu, test jsem několikrát zopakoval pro několik dalších vygenerovaných souborů a všechny testy skončily úspěšně. Všechny výsledné i průběžné hodnoty jednoho z testů jsou v příloze.

6 NAPROGRAMOVANÝ GENERÁTOR

Pro vývoj generátoru jsem se rozhodl použít jazyk C a jako vývojové prostředí Visual Studio 2010. Jazyk C je vhodný pro práci s bity na nejnižší úrovni a s prostředím Visual Studia 2010 jsem se seznámil v předmětech BPC1 a BPC2 na bakalářském studiu.

Pro čtení potřebných dat ze zvukové a síťové karty bylo třeba zvolit vhodné knihovny. Tyto externí knihovny je třeba správně nainstalovat a přidat do projektu.

6.1 Knihovna pro zvukovou kartu

Rozhodl jsem se použít knihovnu **PortAudio**. Má několik předností, díky kterým je vhodná pro implementaci. Je to freewarová, multiplatformní audio I/O knihovna vhodná pro tvorbu programů v jazycích C a C++. Použitelné platformy:

- Windows
- Macintosh
- OS X
- Unix (OSS/ALSA)

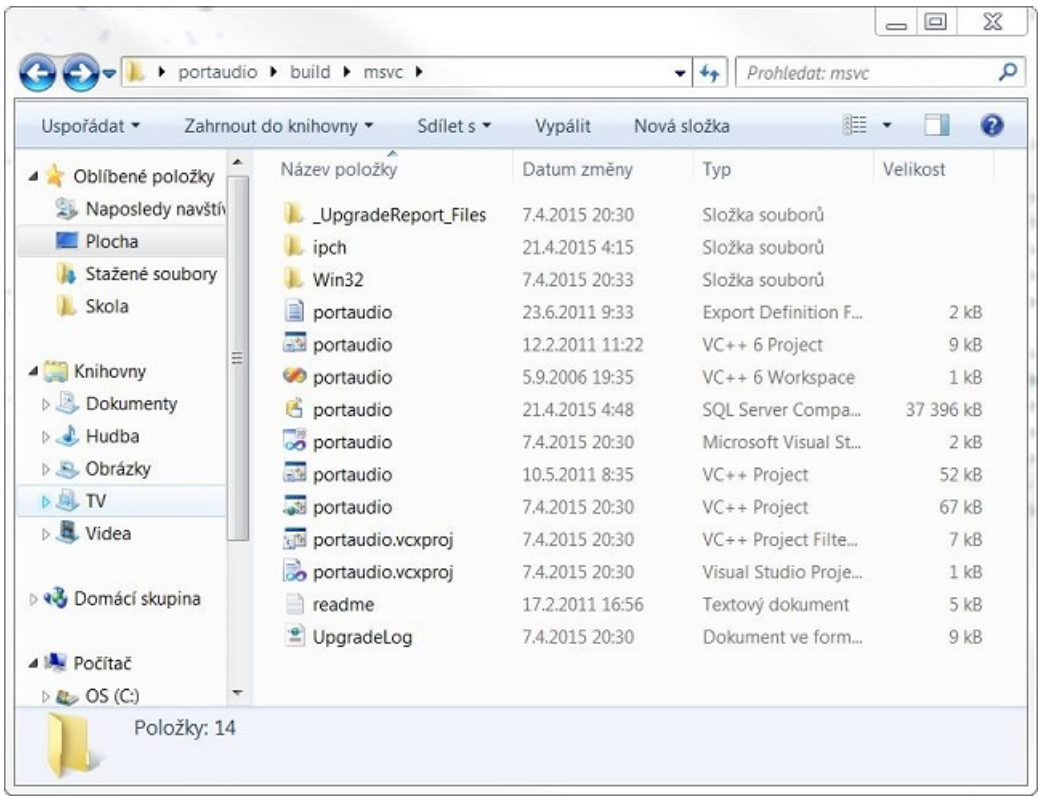
Generátor, který bude využívat její funkce, bude tedy funkční na většině současných platforem. Přímo od autorů této knihovny jsou k dispozici tutoriály a základní programy, které demostrují možnosti využití této knihovny. Poskytuje množství API funkcí pro nahrávání, přehrávání zvuku a další úpravy dat ze zvukové karty. Aby bylo možné její funkce v projektu využívat, je nutné ji správně sestavit a do projektu nalinkovat.

Knihovnu doporučuji stáhnout z oficiálních stránek: `http://www.portaudio.com/download.html`. Jsou zde vždy uvedeny nejnovější stabilní a otestované verze, v případě problémů je možné stáhnout i starší verze. Na těchto stránkách je i veškerá potřebná dokumentace. Poslední stabilní verze: *pa_stable_v19_20140130.tgz* obsahuje všechny potřebné soubory.

Sestavení knihovny se liší podle používané platformy a vývojového prostředí. Pro Visual Studio 2010 je možné otevřít předpřipravený projekt, který se nachází ve složce `portaudio\build\msvc` ve staženém archivu, jak je zobrazeno na obrázku 6.1. Pro správné zbuildování projektu je nutné mít nainstalovány potřebné aktualizace a další knihovny od Microsoftu:[10]

- Service pack 5 pro Visual Studio 6+

- Processor Pack
- Platform SDK
- DirectX 9.0 SDK Update



Obr. 6.1: Umístění předpřipraveného projektu.

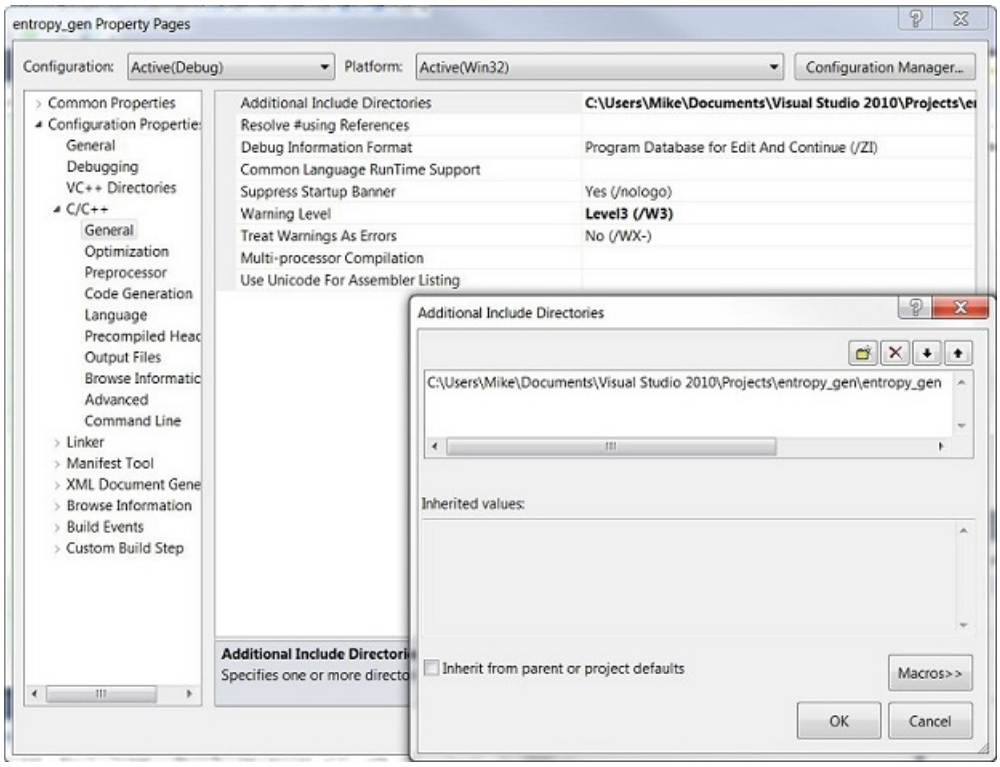
Po úspěšném načtení celého projektu je nutné do jeho umístění nakopírovat všechny potřebné soubory ze složky `src\hostapi\asio\ASIOSDK` ze staženého archivu. Jednotlivé složky, jejichž obsah je potřeba vykopírovat:[10]

- `portaudio\src\hostapi\asio\ASIOSDK\common`
- `portaudio\src\hostapi\asio\ASIOSDK\host`
- `portaudio\src\hostapi\asio\ASIOSDK\host\sample`
- `portaudio\src\hostapi\asio\ASIOSDK\host\pc`

Před samotným sestavením je nutné ještě zvolit požadovanou konfiguraci buď Win32 nebo x64. Pro obě je možné zvolit debug, release a releasemindependency verzi. Poslední zmíněná verze odstraňuje závislost na ostatních základních knihovnách operačního systému. Pokud je vše správně nastaveno, projekt se sestaví a vytvoří pro zvolený systém potřebné knihovny, aby bylo možné v projektu používat hla-

vičkový soubor **portaudio.h**. Ve složce `portaudio\build\msvc\Win32\Debug` nalezneme sestavené soubory, konkrétně **portaudio_x86.lib** a **portaudio_x86.dll**.

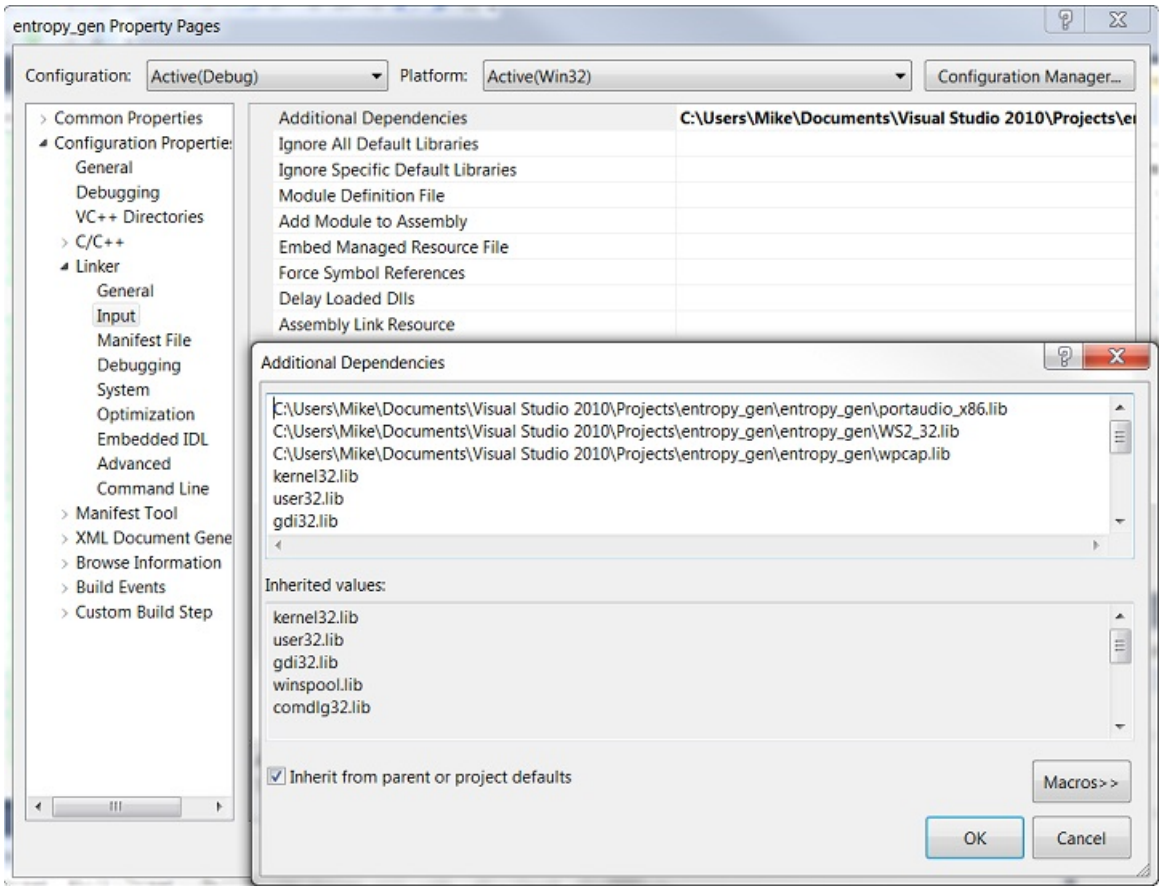
Tyto knihovny je nutné nakopírovat do projektu, kde mají být použity spolu se všemi hlavičkovými soubory, které jsou ve staženém archivu `portaudio\include`. Dále, aby bylo možné využívat všechny API funkce knihovny, přidáme do projektu **#include "portaudio.h"** a musíme v nastavení projektu nastavit i cestu k samotnému souboru, protože se jedná o externí knihovnu a kompilátor by ji implicitně nenašel. Ta se přidává do položky **C/C++ -> General -> Additional Include Directories**, jak je zobrazeno na obrázku 6.2.



Obr. 6.2: Nastavení cesty k hlavičkovému souboru portaudio.h.

Dále se do nastavení musí přidat i cesta ke knihovně **portaudio_x86.lib**. Ta se nastavuje v položce **Linker -> Input -> Additional Dependencies**, jak je zobrazeno na obrázku 6.3.

Jakmile jsou všechna nastavení hotová a projekt zbuildujeme, je možné začít využívat všechny funkce, které knihovna poskytuje.



Obr. 6.3: Nastavení cesty ke knihovně portaudio_x86.lib.

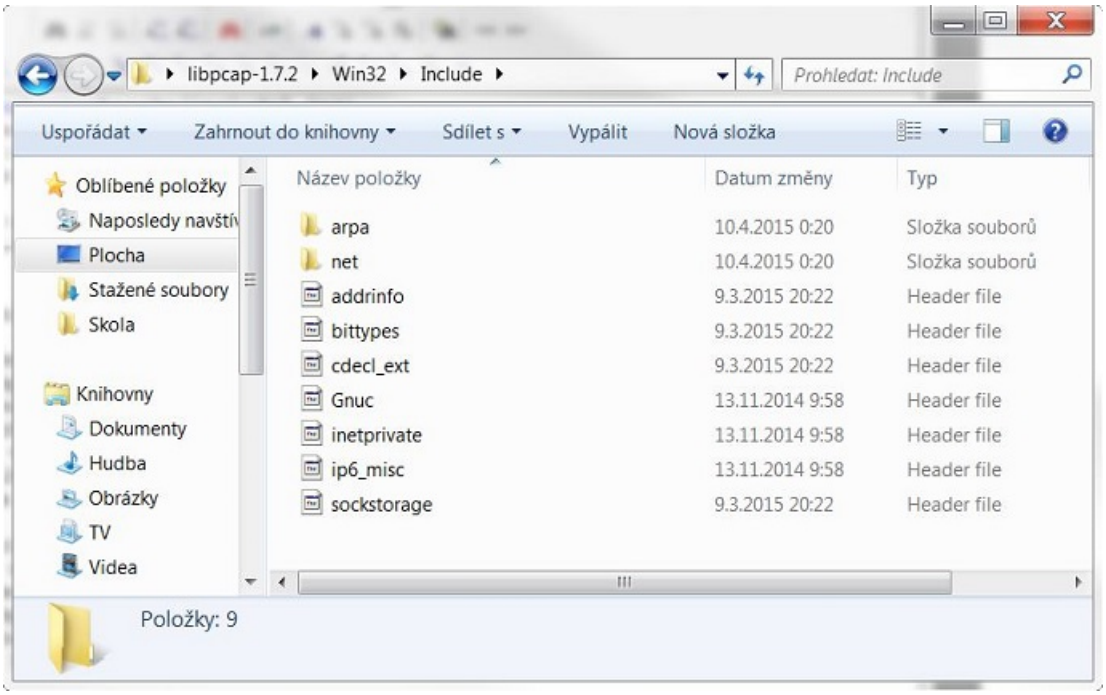
6.2 Knihovna pro síťovou kartu

Jako knihovnu pro čtení dat ze síťové karty jsem se rozhodl využít freewarovou knihovnu **libpcap**. Tuto knihovnu využivají i známé programy jako Wireshark nebo Tcpdump a je velmi dobře zdokumentovaná. Nejnovější verzi lze stáhnout na oficiálních stránkách, kde vycházejí pravidelné updaty otestovaných, stabilních verzí: `http://www.tcpdump.org/#latest-release`.[9]

Nejnovější verze *libpcap-1.7.2.tar.gz* obsahuje všechny potřebné hlavičkové soubory, ale pro využití knihovny na OS Windows je nutné stáhnout a nainstalovat ještě developerský balíček, který je dostupný ke stažení na `http://www.winpcap.org/devel.htm`

Po instalaci jsou k dispozici knihovny **wpcap.lib** a **packet.lib**, které je třeba nalinkovat do projektu. Ty je nutné opět přidat i do položky **Linker -> Input -> Additional Dependencies**, jak už bylo ukázáno na obrázku 6.3 výše. Ze staženého archivu *libpcap-1.7.2.tar.gz* poté musíme do projektu překopírovat ještě soubory umístěné ve složce `libpcap-1.7.2\Win32\Include`, jak je vidět na obrázku 6.4 a opět je přidat také do **C/C++ -> General -> Additional Include Di-**

rectories v nastavení, jak bylo popsáno výše na obrázku 6.2.

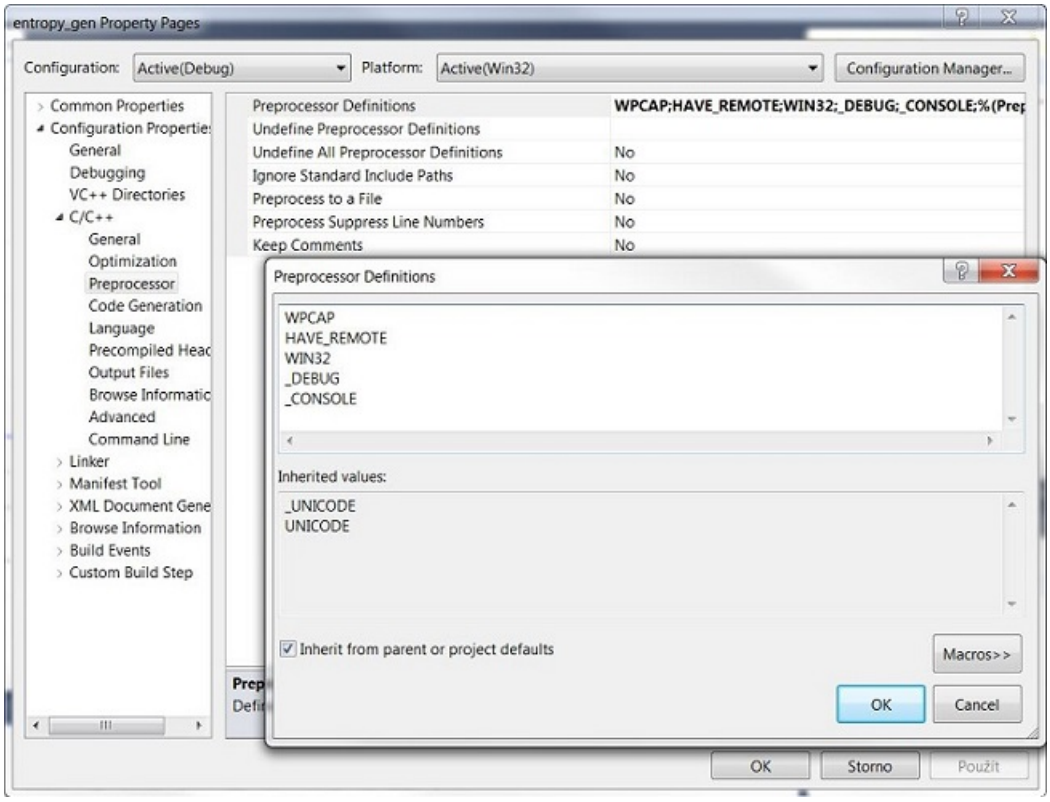


Obr. 6.4: Složka s potřebnými hlavičkovými soubory.

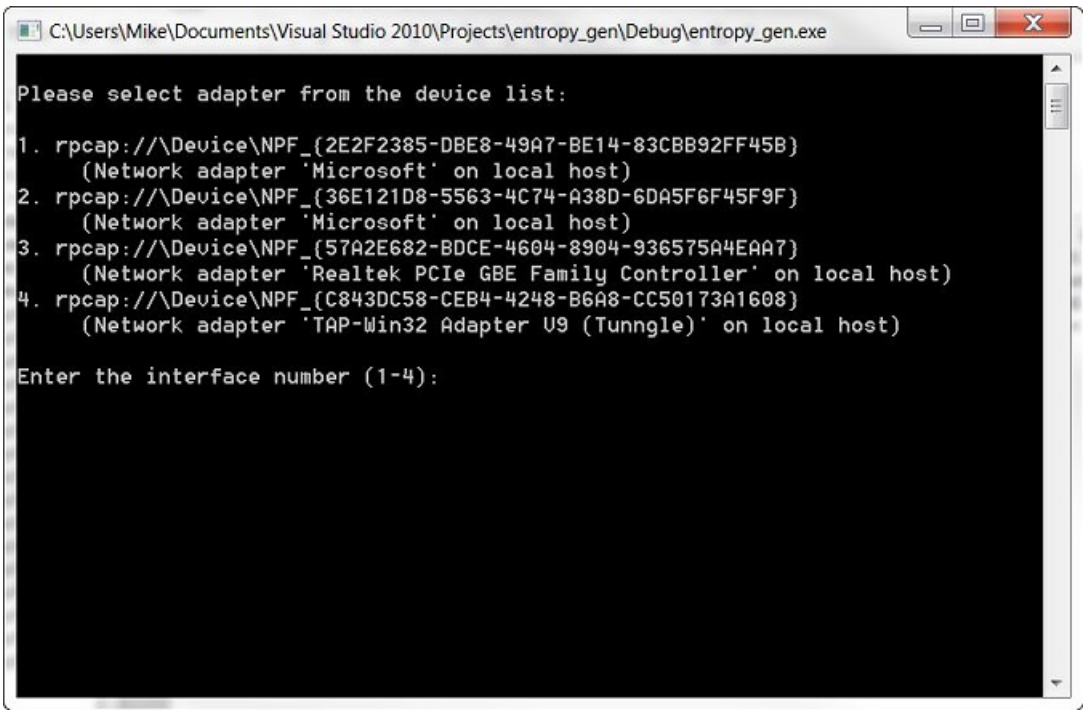
Kromě těchto knihoven je nutné nalinkovat ještě i **ws2_32.lib** a dodatečně stáhnout hlavičkové soubory **inttypes.h** a **time.h**, které stačí překopírovat do složky s projektem. Jakmile jsou všechny potřebné soubory překopírovány, lze do programu přidat hlavičku **#include "pcap.h“**. Před samotným buildnutím programu je třeba přidat ještě dvě preprocesorové definice sice: **WPCAP** a **HAVE_REMOTE**. To se nastavuje v **C/C++ -> Preprocessor -> Preprocessor Definitions** viz obrázek 6.5.

6.3 Vytvořený program

Vytvořený program je konzolová WIN32 aplikace. Při zapnutí program vyhledá všechna dostupná síťová zařízení a uživatel vybere, která chce využít pro generování náhodných čísel. Poté je třeba vložit požadovaný počet pseudonáhodných čísel. Po vložení příslušné volby se spustí detekce vhodného audio I/O rozhraní, která je automatická. Poté začne generátor pracovat a generovat pseudonáhodné hexa sekvence do výstupního souboru **random.txt**. Vstupní rozhraní programu je na obrázku 6.6.

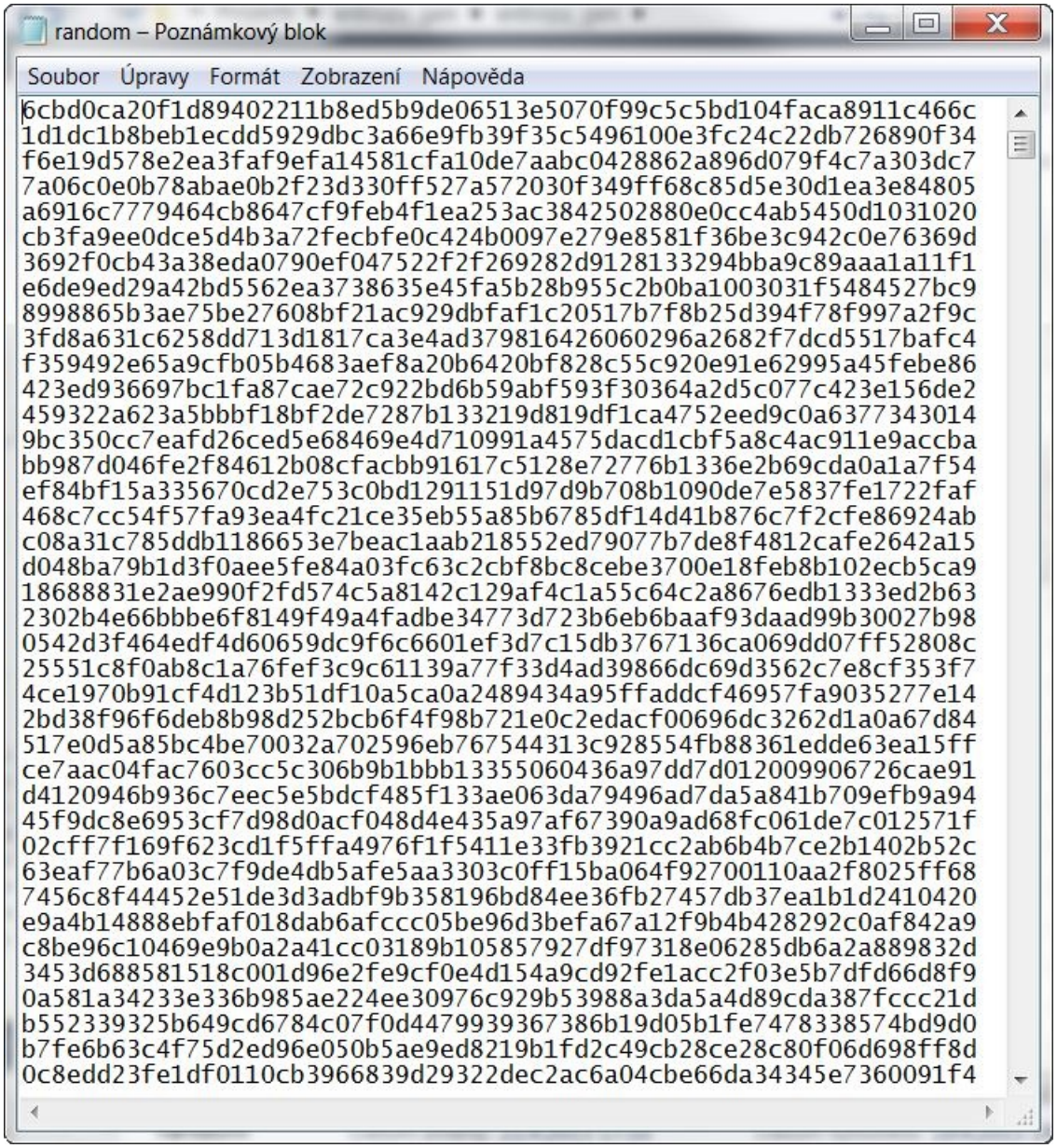


Obr. 6.5: Nastavení preprocessorových definic.



Obr. 6.6: Inicializační obrazovka generátoru.

Program v konzoli vypisuje oznámení o postupech programu. Například, že probíhá obnovovací proces semínka nebo že nastala nějaká chyba. Pokud se jedná o chybu, kterou program umí sám vyřešit, bude pracovat dál, v opačném případě bude nutný vstup uživatele. Část výstupního souboru ukazuje obrázek 6.7.



Obr. 6.7: Inicializační obrazovka generátoru.

Obsah tohoto souboru bude podroben souboru testů NIST, který jsem již probíral v kapitole 2.4. O generování náhodného výstupu se stará generující funkce, jejíž zdrojový kód je vidět na obrázku 6.8. Výstupní data jsou hexa stringy o velikosti 32 bitů. S každou vygenerovanou sekvencí dochází ke změně hodnot vnitřního stavu

a snížení počítadla *ressed_counter*. Jakmile je počítadlo rovno nule, dojde k zavolání fuknce pro obnovení semínka a vnitřního stavu generátoru. Zdrojový kód obnovovací funce je na obrázku 6.9.

```
1053 int GenerateAlgorithm (unsigned char *working_state_V, int requested_number_of_bits){
1054
1055     int m=0;
1056     unsigned char H[32];
1057     m = requested_number_of_bits/32;
1058     double two = 2;
1059     double power = pow(two,256);
1060
1061         if (state.ressed_counter == 0)
1062           RessedAlgorithm(state.V, NULL);
1063
1064
1065          sha256_init(&ctx);
1066          sha256_update(&ctx,(working_state_V+m),440);
1067          sha256_final(&ctx,returned_bits);
1068          print_hash(returned_bits);
1069
1070     for (i=0 ; i < 32 ; i++){
1071         fprintf(file, "%02x", returned_bits[i]);
1072     }
1073     fprintf(file, "\n");
1074
1075     //printf("\nNew H: ");
1076      sha256_init(&ctx);
1077      sha256_update(&ctx,(working_state_V),440);
1078      sha256_final(&ctx,H);
1079
1080      for(i=0; i <32 ; i++){
1081      state.V[i] =  state.V[i] + state.C[i] + H[i] % (int) power;
1082      }
1083      print_hash(H);
1084      //printf("\nNew V: ");
1085      print_hash(state.V);
1086
1087      state.ressed_counter--;
1088      printf("\nCounter %d: ", state.ressed_counter);
1089
1090     return 0;
1091 }
```

Obr. 6.8: Generující funkce.

Obnovovací funkce znovu zavolá funkci **GetEntropyInput**, aby obdržela nová data ze zdrojů entropie a mohla z nich vytvořit nové hodnoty vnitřního stavu a semínko. Lze využít stejnou funkci pro získání entropie, jako pro inicializaci generátoru, protože podle doporučení NIST by obnovovací funkce měla využívat stejné zdroje entropie jako inicializační funkce. Dojde také k resetování počítadla *ressed_counter*. Jak jsem zmínil již dříve **Additional_input** nebude přímo implementován.

```
964 int RessedAlgorithm(unsigned char *old_seed, char *additional_input){
965
966     GetEntropyInput(32);
967     seed_material[50] = NULL;
968     char us_hash[32];
969     char partial[50];
970     if(additional_input != NULL)
971       strcpy(partial, additional_input);
972
973     //strcpy(partial," ");
974     for (i=0; (i < 32 ) ; i++){
975         us_hash[i]=(char) hash[i];
976     }
977     strcpy(partial, us_hash);
978     printf("\nReSeed: ");
979     for (i=0; (i < 50 ) ; i++){
980     printf("%c", partial[i]);
981     }
982
983     for (i=0; (i < 50 ) ; i++){
984         seed_material[i]=(unsigned char) partial[i];
985     }
986     printf("\nReUnSeed: ");
987     for (i=0; (i < 50 ) ; i++){
988     printf("%c", seed_material[i]);
989     }
990     printf("\nOld_Seed: ");
991     print_hash(old_seed);
992
993     unsigned char seed[100];
994     unsigned char new_seed[32];
995     for (i=0; (i < 50 ) ; i++){
996     seed[i] = seed_material[i];
997     }
998     for (i=50; (i < 100 ) ; i++){
999     seed[i] = old_seed[i];
1000    }
1001    //NEW WORKING_STATE state;
1002   sha256_init(&ctx);
1003   sha256_update(&ctx,seed,440);
1004   sha256_final(&ctx,new_seed);
1005   printf("\n\nNew working state V: \n");
1006   for (i=0; (i < 32 ) ; i++){
1007    state.V[i]=new_seed[i];
1008   }
1009   print_hash(state.V);
1010
1011   sha256_init(&ctx);
1012   sha256_update(&ctx,state.V,440);
1013   sha256_final(&ctx,state.C);
1014   printf("\nNew working state C: \n");
1015   print_hash(state.C);
1016
1017   state.ressed_counter = 1000;
1018
1019    return 0;
1020 }
```

Obr. 6.9: Obnovovací funkce.

Obnovovací funkce přesto testuje, jestli parametr *additional_input* není roven NULL. Pokud by obsahoval nějaká data, přidal by je do materiálu, ze kterého se bude tvořit semínko a nový vnitřní stav.

7 ZÁVĚR

Náhodná čísla byla v počátcích předmětem výzkumu kvůli hazardním hrám. Ruleta, přeskupování hracích karet a mnoho dalších byly první stroje, které využívaly náhodná čísla. Náhodná čísla jsou také využívána pro neobchodní účely. Jedno z nejširších využití mají v kryptografii. Šifrování dat je v současnosti nedílnou součástí téměř všech systémů, protože kybernetická kriminalita každým rokem roste. Moderní technologie jsou stále dostupnější a nástroje pro nabourávání počítačů nebo lámání hesel jsou čím dál více rozšířenější. Z tohoto důvodu jsou nutné silné šifrovací klíče, které zajistí bezpečný přenos dat nebo jejich uložení na disku.

Právě použití hardwarových generátorů náhodných čísel se využívá pro tvorbu šifrovacích klíčů. V práci jsem pojednal o základních typech generátorů a popsal jejich výhody a nevýhody. Z nich se jako nejvhodnější typ jeví Entropické generátory. Ovšem základem těchto generátorů jsou generátory pseudonáhodných posloupností, a tak výběr vhodného GNPN je pro celý návrh zcela zásadní. Znovu shrnu, co by měl splňovat:

- Vysoká rychlost generovaných bitů,
- Vysoká entropie,
- Minimální periodicita,
- Nemá jako základ LKG,
- Není obsahem některé z knihoven pro programovací jazyky.

Nejvhodnějšími zdroji entropie se po testování ukázaly **zvuková karta**, **síťová karta** a využití **pixelů na monitoru**. Podle doporučení NIST by měl mít generátor alespoň dva zdroje entropie. Jako vhodné zdroje entropie jsem tedy zvolil zvukovou kartu a síťovou kartu, protože při měření podaly kvalitní výsledky a jejich implementace je dobře zdokumentovaná. V současnosti jsou zvuková i síťová karta integrované na všech základních deskách, a tudíž jsou součastí každého osobního počítače. Využívá se šumu na vstupu zvukové karty a náhodných paketů na vstupu síťové karty. Velkou výhodou je, že se jedná o náhodné jevy a není zde v podstatě žádná perioda opakování, především když oba zdroje pro tvorbu inicializačního semínka skombinujeme. Jako DRBG mechanismus jsem zvolil **hash_df** pro jeho rychlost díky, použitým hashovacím funkcím i dobře zdokumentovanou implementaci pro jazyky C a C++.

Vytvořený generátor je konzolová aplikace, což pro účely testování zcela dostačuje. Generátor prošel souborem testů vytvořeným NIST a celý test prošel bez jediné chyby. Tímto jsem prokázal, že generátory navržené podle NIST jsou opravdu spolehlivé pro tvorbu náhodných čísel.

LITERATURA

[1] BARKER, E., KELSEY, J. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. NIST SP 800-90A. National Institute of Standards and Technology, Gaithersburg 2012.

[2] BARKER, E., KELSEY, J. *Recommendation for the Entropy Sources Used for Random Bit Generation*. NIST SP 800-90B. National Institute of Standards and Technology, Gaithersburg 2012.

[3] BARKER, E., KELSEY, J. *Recommendation for Random Bit Generator (RBG) Constructions*. NIST SP 800-90C. National Institute of Standards and Technology, Gaithersburg 2012.

[4] BALDWIN, W. *Preliminary Analysis of the BSAFE 3.x Pseudoradom Number Generators*. California: RSA Laboratirie's Bulletin, 1998. Dostupné z: <`ftp://ftp.rsasecurity.com/pub/pdfs/bulletn8.pdf` >

[5] BURDA, K. *Aplikovaná kryptografie*. Brno: VUTIUM, 2013. ISBN 978-80-214-4612-0.

[6] HERBERT, H. *Digital random number generator using partially entropic data*. Phoenix: United States Patent US8489660B2, 2009. Dostupné z URL: <`http://www.freepatentsonline.com/8489660.pdf` >

[7] RUKHIN, A. *A statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Generators for Cryptographic Applications*. NIST SP 800-22. National Institute of Standards and Technology, Gaithersburg 2010.

[8] CHAICHANAVONG, P. *Entropy source for random number generation*. Mountain View: United States Patent US8015224B1, 2007. Dostupné z URL: <`http://www.freepatentsonline.com/8015224.pdf` >

[9] LUIS, M. *TCPDUMP/LIBPCAP public repository*. Tcpdump/Libpcap. [online]. 2010. Dostupné z: <`http://www.tcpdump.org` >

[10] BENCINA, R., BURK, P. *PortAudio - an Open-Source Cross-Platform Audio API*. PortAudio. [online]. 2011. Dostupné z: <`http://www.portaudio.com/` >

[11] MORRISON, R. *Design of a True Random Number Generator Using Audio Input*. Oregon: Department of Electrical and Computer Engineering, 2001. Dostupné z URL: <`http://www.cs.ucsb.edu/~koc/cren/project/pp/morrison.pdf` >

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

NIST Národní Institut Standardů a Technologie

SLL Secure Socket Layer

RNG Random Number Generator

SLL Generátor Náhodných Posloupností

SLL Generátor Pseudonáhodných Posloupností

SLL Lineární Korgurentní Generátor

SLL Least Significant Bit

LFSR Linear Feedback Shift Register

CPU Central Processing Unit

LZMA Lempel-Ziv-Markov-Chain Algorithm

LZ77 Lempel-Ziv 77

LZW Lempel-Ziv-Welch

DRBG Deterministic Random Bit Generator

HMAC keyed-Hash Message Authentication Code

CMAC Cipher-based Message Authentication Code

CBC-MAC Cipher Block Chaining Message Authentication Code

Hash_df Hash-based Derivation Function

bc_df Block Ciper-based Derivation Function

AES Advanced Encryption Standard

TDEA Triple Data Encryption Algorithm

GCC GNU Compiler Collection

GCC GNU Compiler Collection

SEZNAM PŘÍLOH

A ZÁZNAM TESTU

Jakmile proběhne celý test, jsou k dipozici výstupní informace o každém z testů a na samotném konci je zhodnocení celého testu.

A.1 Test četnosti

128 bits sequence, p_value= 0.859684
256 bits sequence, p_value= 0.381574
512 bits sequence, p_value= 0.723674
1024 bits sequence, p_value= 0.080118
2048 bits sequence, p_value= 0.121912
4096 bits sequence, p_value= 0.453255
8192 bits sequence, p_value= 0.111612
16384 bits sequence, p_value= 0.295157
32768 bits sequence, p_value= 0.731971
65536 bits sequence, p_value= 0.317311
131072 bits sequence, p_value= 0.237129
262144 bits sequence, p_value= 0.383703
524288 bits sequence, p_value= 0.375270
1048576 bits sequence, p_value= 0.923756
2097152 bits sequence, p_value= 0.431968
4194304 bits sequence, p_value= 0.333646
8388608 bits sequence, p_value= 0.679651
16777216 bits sequence, p_value= 0.232156
33554432 bits sequence, p_value= 0.396067
67108864 bits sequence, p_value= 0.711841

A.2 Test četnosti v bloku

128 bits sequence, p_value= 0.483376
256 bits sequence, p_value= 0.533440
512 bits sequence, p_value= 0.437197
1024 bits sequence, p_value= 0.626861
2048 bits sequence, p_value= 0.685880
4096 bits sequence, p_value= 0.051445
8192 bits sequence, p_value= 0.166463

16384 bits sequence, p_value= 0.505201
32768 bits sequence, p_value= 0.778934
65536 bits sequence, p_value= 0.909864
131072 bits sequence, p_value= 0.303527
262144 bits sequence, p_value= 0.307461
524288 bits sequence, p_value= 0.712218
1048576 bits sequence, p_value= 0.816263
2097152 bits sequence, p_value= 0.766500
4194304 bits sequence, p_value= 0.382282
8388608 bits sequence, p_value= 0.145743
16777216 bits sequence, p_value= 0.863676
33554432 bits sequence, p_value= 0.559078
67108864 bits sequence, p_value= 0.988112

A.3 Test shodné série

p_value=0.360215 p_value=0.028350
p_value=0.757194 p_value=0.863131
p_value=0.545488 p_value=0.510722
p_value=0.609504 p_value=0.672405
p_value=0.636116 p_value=0.739055
p_value=0.576107 p_value=0.679519
p_value=0.097669 p_value=0.103672
p_value=0.372989 p_value=0.305359
p_value=0.812543 p_value=0.987290
p_value=0.501651 p_value=0.782159

A.4 Test nejdelší shodné série

p_value= 0.908779 p_value= 0.606352
p_value= 0.751282 p_value= 0.109820
p_value= 0.525007 p_value= 0.126761
p_value= 0.814290 p_value= 0.876130

A.5 Test dílčích matic na 8-mi náhodně zvolených blocích o délce 38 912 bitů

p_value= 0.682148 p_value= 0.567723
p_value= 0.680524 p_value= 0.458288
p_value= 0.458288 p_value= 0.325990
p_value= 0.968938 p_value= 0.402553

A.6 Test diskrétní Fourierovy transformace

p_value= 0.220375 p_value= 0.876124
p_value= 0.101542 p_value= 0.623804
p_value= 0.056967 p_value= 0.788177
p_value= 0.030029 p_value= 0.532931

A.7 Test nepřekrývajících se šablon

p_value= 0.635352 p_value= 0.152669 p_value= 0.647556 p_value= 0.887117
p_value= 0.890712 p_value= 0.411159 p_value= 0.410752 p_value= 0.827054
p_value= 0.617698 p_value= 0.529524 p_value= 0.635352 p_value= 0.766618
p_value= 0.856085 p_value= 0.641333 p_value= 0.855204 p_value= 0.700716
p_value= 0.236981 p_value= 0.233850 p_value= 0.882412 p_value= 0.014483
p_value= 0.319194 p_value= 0.737566 p_value= 0.816177 p_value= 0.876466
p_value= 0.695839 p_value= 0.059760 p_value= 0.364866 p_value= 0.808193
p_value= 0.966127 p_value= 0.116543 p_value= 0.196419 p_value= 0.180518
p_value= 0.370954 p_value= 0.257582 p_value= 0.464473 p_value= 0.632765
p_value= 0.230624 p_value= 0.538486 p_value= 0.901392 p_value= 0.092640
p_value= 0.682048 p_value= 0.892465 p_value= 0.741648 p_value= 0.343928
p_value= 0.734833 p_value= 0.595415 p_value= 0.849480 p_value= 0.750929
p_value= 0.240211 p_value= 0.723449 p_value= 0.147116 p_value= 0.109636
p_value= 0.814939 p_value= 0.743685 p_value= 0.604923 p_value= 0.754494
p_value= 0.927759 p_value= 0.341720 p_value= 0.693397 p_value= 0.121467
p_value= 0.833663 p_value= 0.410752 p_value= 0.127430 p_value= 0.762503
p_value= 0.922675 p_value= 0.162113 p_value= 0.512776 p_value= 0.110265
p_value= 0.479924 p_value= 0.980720 p_value= 0.172407 p_value= 0.083282
p_value= 0.068039 p_value= 0.787045 p_value= 0.523194 p_value= 0.599456

p_value= 0.806634 p_value= 0.694394 p_value= 0.369294 p_value= 0.902765
p_value= 0.552307 p_value= 0.011805 p_value= 0.835156 p_value= 0.252137
p_value= 0.973153 p_value= 0.775086 p_value= 0.446952 p_value= 0.286791
p_value= 0.329333 p_value= 0.855513 p_value= 0.885669 p_value= 0.161326
p_value= 0.480640 p_value= 0.697143 p_value= 0.867647 p_value= 0.625531
p_value= 0.057230 p_value= 0.845813 p_value= 0.781528 p_value= 0.034486
p_value= 0.959826 p_value= 0.859054 p_value= 0.319661 p_value= 0.952080
p_value= 0.508887 p_value= 0.977774 p_value= 0.986709 p_value= 0.399095
p_value= 0.908951 p_value= 0.400272 p_value= 0.081381 p_value= 0.318119
p_value= 0.147607 p_value= 0.572485 p_value= 0.492441 p_value= 0.646566
p_value= 0.323846 p_value= 0.448597 p_value= 0.730799 p_value= 0.309445
p_value= 0.201692 p_value= 0.493863 p_value= 0.980810 p_value= 0.096350
p_value= 0.192206 p_value= 0.348004 p_value= 0.732274 p_value= 0.486654
p_value= 0.400460 p_value= 0.160216 p_value= 0.605992 p_value= 0.178372
p_value= 0.150389 p_value= 0.922166 p_value= 0.961636 p_value= 0.130047
p_value= 0.594882 p_value= 0.203955 p_value= 0.240088 p_value= 0.454536
p_value= 0.130275 p_value= 0.027999 p_value= 0.693650 p_value= 0.805224
p_value= 0.771500 p_value= 0.507140 p_value= 0.792561 p_value= 0.028750

A.8 Test překrývajících se šablon

m= 9, p_value= 0.879858 m= 9, p_value= 0.435802
m= 9, p_value= 0.010758 m= 9, p_value= 0.841457
m= 9, p_value= 0.699102 m= 9, p_value= 0.964408
m= 9, p_value= 0.376847 m= 9, p_value= 0.031707

A.9 Maurerův test

Pro každý parametr **L** je testován začátek sekvence.

L= 6 p_value 0.800156 exp_value= 5.217705 phi = 5.217067
L= 7 p_value 0.207551 exp_value= 6.196251 phi = 6.193834
L= 8 p_value 0.103060 exp_value= 7.183666 phi = 7.181339
L= 9 p_value 0.308037 exp_value= 8.176425 phi = 8.175357
L= 10 p_value 0.562245 exp_value= 9.172324 phi = 9.171883
L= 11 p_value 0.039215 exp_value= 10.170032 phi = 10.168897

L= 12 p_value 0.429576 exp_value= 11.168765 phi = 11.168368

A.10 Test lineární zpětné vazby pro blok 4731 * 371 bitů

p_value= 0.637938

A.11 Test četnosti série

p_value1= 0.716996 p_value1= 0.911637
p_value2= 0.750267 p_value2= 0.546073
p_value1= 0.345596 p_value1= 0.451399
p_value2= 0.779818 p_value2= 0.215418
p_value1= 0.508493 p_value1= 0.989100
p_value2= 0.557338 p_value2= 0.798621
p_value1= 0.578131 p_value1= 0.350308
p_value2= 0.645704 p_value2= 0.975333

A.12 Test přibližné etropie

m= 3, p_value= 0.998663, Entropy per 3 bits 3.000000
m= 4, p_value= 0.983141, Entropy per 4 bits 4.000000
m= 5, p_value= 0.965163, Entropy per 5 bits 5.000000
m= 6, p_value= 0.282692, Entropy per 6 bits 6.000000
m= 7, p_value= 0.565257, Entropy per 7 bits 6.999999
m= 8, p_value= 0.449001, Entropy per 8 bits 7.999998
m= 9, p_value= 0.739376, Entropy per 9 bits 8.999995
m= 10, p_value= 0.560367, Entropy per 10 bits 9.999990
m= 11, p_value= 0.571811, Entropy per 11 bits 10.999979
m= 12, p_value= 0.911747, Entropy per 12 bits 11.999957
m= 13, p_value= 0.611212, Entropy per 13 bits 12.999914
m= 14, p_value= 0.156374, Entropy per 14 bits 13.999826
m= 15, p_value= 0.192189, Entropy per 15 bits 14.999648
m= 16, p_value= 0.649962, Entropy per 16 bits 15.999294

A.13 Test kumulativních součtů

128 bits sequence, forward p_value= 0.984155
128 bits sequence, reverse p_value= 0.369656
256 bits sequence, forward p_value= 0.422245
256 bits sequence, reverse p_value= 0.338189
512 bits sequence, forward p_value= 0.696011
512 bits sequence, reverse p_value= 0.574764
1024 bits sequence, forward p_value= 0.105368
1024 bits sequence, reverse p_value= 0.183014
2048 bits sequence, forward p_value= 0.146951
2048 bits sequence, reverse p_value= 0.032029
4096 bits sequence, forward p_value= 0.319251
4096 bits sequence, reverse p_value= 0.469326
8192 bits sequence, forward p_value= 0.213448
8192 bits sequence, reverse p_value= 0.013138
16384 bits sequence, forward p_value= 0.288042
16384 bits sequence, reverse p_value= 0.469326
32768 bits sequence, forward p_value= 0.879260
32768 bits sequence, reverse p_value= 0.993117
65536 bits sequence, forward p_value= 0.175632
65536 bits sequence, reverse p_value= 0.556137
131072 bits sequence, forward p_value= 0.120693
131072 bits sequence, reverse p_value= 0.344739
262144 bits sequence, forward p_value= 0.614992
262144 bits sequence, reverse p_value= 0.117349
524288 bits sequence, forward p_value= 0.349971
524288 bits sequence, reverse p_value= 0.652202
1048576 bits sequence, forward p_value= 0.970943
1048576 bits sequence, reverse p_value= 0.919168
2097152 bits sequence, forward p_value= 0.522677
2097152 bits sequence, reverse p_value= 0.153278
4194304 bits sequence, forward p_value= 0.580025
4194304 bits sequence, reverse p_value= 0.385477
8388608 bits sequence, forward p_value= 0.543360
8388608 bits sequence, reverse p_value= 0.875988
16777216 bits sequence, forward p_value= 0.193568
16777216 bits sequence, reverse p_value= 0.429380
33554432 bits sequence, forward p_value= 0.379446

33554432 bits sequence, reverse p_value= 0.451606
67108864 bits sequence, forward p_value= 0.929037
67108864 bits sequence, reverse p_value= 0.665856

A.14 Test náhodných návštěv

p_value= 0.277462 p_value= 0.456106 p_value= 0.182839
p_value= 0.040879 p_value= 0.997692 p_value= 0.228732
p_value= 0.098523 p_value= 0.089963 p_value= 0.715813
p_value= 0.120410 p_value= 0.136599 p_value= 0.892116
p_value= 0.173576 p_value= 0.071681 p_value= 0.659822
p_value= 0.094980 p_value= 0.294479 p_value= 0.949403
p_value= 0.293765 p_value= 0.495538 p_value= 0.870196
p_value= 0.753354 p_value= 0.290159 p_value= 0.845178
p_value= 0.994206 p_value= 0.386728 p_value= 0.313315
p_value= 0.524437 p_value= 0.532899 p_value= 0.337662
p_value= 0.201947 p_value= 0.678789 p_value= 0.137064
p_value= 0.234305 p_value= 0.442346 p_value= 0.924307
p_value= 0.697972 p_value= 0.696393
p_value= 0.924993 p_value= 0.294374

A.15 Test variací náhodných návštěv

p_value= 0.600720 p_value= 0.369235 p_value= 0.457580
p_value= 0.489379 p_value= 0.702801 p_value= 0.319971
p_value= 0.313182 p_value= 0.917021 p_value= 0.250816
p_value= 0.323157 p_value= 0.741822 p_value= 0.356113
p_value= 0.551861 p_value= 0.793257 p_value= 0.420770
p_value= 0.781854 p_value= 0.761278 p_value= 0.393721
p_value= 0.933535 p_value= 0.966603 p_value= 0.733078
p_value= 0.841518 p_value= 0.835223 p_value= 0.763595
p_value= 0.820855 p_value= 0.654968 p_value= 0.647283
p_value= 0.431927 p_value= 0.515288 p_value= 0.491334
p_value= 0.558899 p_value= 0.420468 p_value= 0.365249
p_value= 0.844155 p_value= 0.586624 p_value= 0.289430
p_value= 0.951825 p_value= 0.958952 p_value= 0.910433

p_value= 0.618983 p_value= 0.973497 p_value= 0.863319
p_value= 0.370457 p_value= 0.885612 p_value= 0.705521
p_value= 0.393435 p_value= 0.862942 p_value= 0.618342
p_value= 0.360270 p_value= 0.727210 p_value= 0.727943
p_value= 0.350480 p_value= 0.597810 p_value= 0.591354
p_value= 0.307063 p_value= 0.493277 p_value= 0.227949
p_value= 0.345995 p_value= 0.992348 p_value= 0.209391
p_value= 0.670633 p_value= 0.445224 p_value= 0.642779
p_value= 0.639185 p_value= 0.438904 p_value= 0.400534
p_value= 0.569551 p_value= 0.552307 p_value= 0.291699
p_value= 0.744037 p_value= 0.489666 p_value= 0.492022
p_value= 0.786827 p_value= 0.615434 p_value= 0.784149
p_value= 0.671609 p_value= 0.714857 p_value= 1.000000
p_value= 0.931159 p_value= 0.977313 p_value= 0.881880
p_value= 0.399636 p_value= 0.544560 p_value= 0.942316
p_value= 0.500744 p_value= 0.328330 p_value= 0.741905
p_value= 0.468835 p_value= 0.429190 p_value= 0.612706