Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Generovanie náhodných čísel na platforme Windows

Bakalárska práca

2021 **Marek Rohač**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Generovanie náhodných čísel na platforme Windows

Bakalárska práca

Študijný program:	Počítačové siete
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra elektroniky a multimediálnych telekomunikácií (KEMT)
Školiteľ:	prof. Ing. Miloš Drutarovský, CSc.
Konzultant:	

Košice 2021 **Marek Rohač**

Abstrakt v SJ

Cieľom teoretickej časti tejto práce je oboznámiť čitateľa s problematikou generovania náhodných čísel v oblasti informačnej bezpečnosti. Následne pomocou opisu štruktúr a mechanizmov operačného systému Microsoft Windows vysvetľuje proces generovania náhodných čísel na tejto platforme. Praktická časť používa programovací jazyka C, knižničné funkcie a predprogramované systémové rozhrania na tvorbu náhodných výstupov. Kvalita takto vzniknutých dát je následne overená pomocou sád štatistických testov. Pomocou experimentu v prostredí virtuálneho stroja s rovnakým operačným systémom je overená bezpečnostná chyba súvisiaca s problematikou tejto práce.

Kľúčové slová v SJ

jazyk C, OpenSSL, pseudonáhodné RNG, skutočne náhodné RNG, štatistické testy, winapi

Abstrakt v AJ

The aim of the theoretical part of this work is to acquaint the reader with the issue of generating random numbers in information security. Subsequently, by describing the structures and mechanisms of the Microsoft Windows operating system, we will explain the generation process on this platform. In the practical part are implemented library function and programmed system interfaces for the production of random outputs by C programing language. The quality of the data generated in this way is then verified by using statistical test suits. By experiment in enviroment of virtual machines with OS Windows is checked security problem related to the topic of this work.

Kľúčové slová v AJ

language C, OpenSSL, pseudo-random RNG, statistical tests, truly random RNG, winapi

Bibliografická citácia

ROHAČ, Marek. *Generovanie náhodných čísel na platforme Windows*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2021. 73s. Vedúci práce: prof. Ing. Miloš Drutarovský, CSc.

61344

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra elektroniky a multimediálnych telekomunikácií

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Počítačové siete**

Názov práce:

Generovanie náhodných čísel na platforme Windows

Random Numbers Generation on Windows Platform

Študent: **Marek Rohač**

Školiteľ: **prof. Ing. Miloš Drutarovský, CSc.**

Školiace pracovisko: **Katedra elektroniky a multimediálnych telekomunikácií**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

Na základe dostupných informácií naštudujte a opíšte mechanizmy generovania náhodných čísel, ktoré sú dostupné pre aplikačné programy na platforme Windows. S využitím jazyka C a dostupných API rozhraní otestujte základné parametre generovania náhodných čísel so zameraním na kryptografické aplikácie. Opíšte a experimentálne overte možnosť útoku na generátor náhodných čísel vo vybraných virtualizačných nástrojoch, ktoré je možné využiť na hosťovskom počítači s operačným systémom Windows.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 28.05.2021

Dátum zadania bakalárskej práce: 30.10.2020

.....

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 28. 5. 2021 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce *prof. Ing. Milošovi Drutarovskému, CSc.* za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Zoznam skratiek

ACPI Advanced Configuration and Power Interface.

AES Advanced Encrytion Standard.

AMD Advanced Micro Devices.

ANC Average Number of Cycles.

API Application Programming Interface.

ASCII American Standard Code for Information Interchange.

BBS Blum Blum Shub generator.

BS Buffer Size.

CAPI Cryptographic Application Programming Interface.

CBC Cipher Block Chaining.

CCM Cipher block chaining Counter Mode.

CFB Cipher FeedBack.

CNG Cryptography API: Next Generation.

CPU Central Processing Unit.

CryptoAPI Cryptographic Application Programming Interface.

CSP Cryptographic Service Provider.

CSPRNG Crypographically Secure Random Number Generator.

CTR Counter mode.

CV Critical Value.

DRNG Deterministic Random Number Generator.

ECB Electronic CodeBook.

FEI Faculty of Electrical Engineering and Informatics.

FIFO First In First Out.

GCC GNU Compiler Collection.

GCM Galois/Counter Mode.

HRNG Hardware Random Number Generator.

KEMT Department of Electronies and Multimedia Telecommunications.

KPI Department of Computers and Informatics.

LFSR Linear-Feadback Shift Register.

MAC Message Authentization Code.

n-DRNG non-Deterministic Random Number Generator.

NI Number of Iterations.

NIST National Institute of Standards and Technology.

OEM Original Equipment Manufacturer.

OS Operating System.

PRNG Pseudo-Random Number Generator.

QPC Query Performance Counter.

QPF Query Performance Frequency.

RBG Random Bit Generator.

RC Rivest Cipher.

RDTSC Read Time-Stamp Counter.

RNG Random Number Generator.

SHA Secure Hash Algorithm.

TPM Trusted Platform Module.

TRNG True Random Number Generator.

TSC Time Stamp Count.

TUKE Technical University of Košice.

UEFI Unified Extensible Firmware Interface.

VM Virtual Machine.

Úvod

Náhodné čísla sú nevyhnutnou súčasťou každodenného použitia počítača. Ich využitie pri práci operačného systému je rozsiahle a prebieha takmer neustále. Podieľajú sa pri zabezpečení sieťovej, a aj počítačovej ochrany pri používaní zariadení. Poskytujú teda ochranu pred možnými útokmi, prostredníctvom ich využitia v rôznych kryptografických algoritmoch. Kvalita náhodných dát, vygenerovaných počítačom, teda odzrkadľuje bezpečnosť zvolenej platformy, ktorá je poskytnutá používateľovi.

Cieľom tejto práce je definovať a klasifikovať prostriedky na tvorbu náhodne vygenerovaných čísel. Následne opisom špecifikujeme vývoj kryptografických rozhraní od svojho vzniku až po súčasnosť operačného systému Windows. Vysvetlíme princípy testovania kvality náhodných čísel. Špecifikujeme možnosti vytvárania týchto dát v uvedenom prostredí. Pomocou rozhraní a knižničných funkcií vykonáme experimenty. Charakterizujeme zvolené metódy merania a testovania. V opise uvedieme aj možné chyby, ktoré ovplyvnili pokusy. Experimenty spojené s časovým testovaním sú vykonané na troch rôznych počítačových konfiguráciách, s rovnakým operačným systémom (Windows 10). Dôvodom použitia viacerých zariadení je kvalitnejšia interpretácia výsledkov. Ďalším krokom je generovanie náhodných dát pomocou vybraných rozhraní. Následné takto vzniknuté údaje otestujeme štatistickými testami. Na základe výstupov štatistického testovanie zistíme kvalitu našich dát. Súčasťou práce je aj overenie bezpečnostného problému v prostredí virtuálneho stroja s operačným systémom Windows. Špecifikáciu rizika vykonáme prostredníctvom opisu a experimentu. Cieľom je praktické overenie aktuálnosti bezpečnostnej chyby v operačnom systéme Windows 10, ktorá vzniká pri generovaní náhodných čísel tesne po obnovení snímky obrazu systému.

Vyhodnotenie výsledkov, ktoré je získané vyššie uvedenými metódami, je obsahom samostatnej kapitoly tejto práce. Opísaná metodika je uskutočnená na celosvetovo najpoužívanejšom operačnom systéme spoločnosti Microsoft – Windows 10, v 64-bitovej verzii. Pri tvorbe programov je použitý programovací jazyk

C so 64-bitovým GCC prekladačom.

1 Typy generátorov náhodných čísel

Pod pojmom generovanie náhodných čísel vzniká predstava jednoduchého procesu ako napríklad hod kockou. V sfére počítačov sa pri tomto deji používajú špeciálne nástroje – *generátory* [1] (ďalej RNG[1]). Výstupnými hodnotami sú postupnosti bitov. Ich neskoršia grafická reprezentácia v počítači môže byť číselná alebo vo forme znakov ASCII[2] tabuľky, prislúchajúcich vygenerovaným postupnostiam. Dôsledkom toho sa čitateľ môže v problematike RNG často stretnúť s pojmom – *generátor náhodných bitov* (ďalej RBG[3]). V oblasti informačnej bezpečnosti je ich použitie skutočne rozsiahle. Radíme ich preto do množiny tzv. *kryptografických primitív*. V kryptografii tento pojem zahŕňa algoritmy, bloky a nástroje, ktoré tvoria základ pre správne fungovanie kryptografických funkcií, respektíve systémov. Podľa dokumentu Národného Inštitútu pre Štandardy a Technológie (ďalej NIST), ich delíme podľa spôsobu tvorby výstupných hodnôt na [2]:

- deterministické – *ang. deterministic RNG* (DRNG),
- ne-deterministické – *ang. non-deterministic RNG* (n-DRNG).

V tejto práci použijeme spomenuté delenie za hierarchicky najvyššie a ďalej opísané v nasledujúcich podkapitolách.

1.1 Ne-Deterministické RNG

Proces generovania náhodných čísel prebieha pri tomto type generátora na najnižších úrovniach počítača – **hardvéri**. Výstup týchto zariadení je závislý od jedného alebo viacerých fyzikálnych dejov, ktoré musia byť štatisticky nepredvídateľné. Celkovo môžeme tieto javy rozdeliť na fyzikálne procesy:

- s kvantovou náhodnosťou,

[1] Z ang. *Random Number Generator*.
[2] Z ang. *American Standard Code for Information Interchange*.
[3] Z ang. *Random Bits Generator*.

- bez kvantovej náhodnosti.

Dôvodom použitia oblasti kvantovej fyziky je ideálna vlastnosť pre generovanie – aktuálna nemožnosť, resp. neschopnosť predikcie týchto procesov. Za spomenutie tiež stojí, že rýchlosť generovania čísel je v tomto prípade pomalšia ako pri deterministických generátoroch. Aj dôsledkom toho sa stretávame s menším zastúpením týchto generátorov v bežnej prevádzke. Veľmi dobrým príkladom použitia je vojenské odvetvie. Najmä pri šifrovaní a následnej preprave veľmi dôležitých, resp. citlivých správ prostredníctvom počítačových sieti. Na označenie takýchto generátorov sa používajú označenia – TRNG[4] a HRNG[5].

1.1.1 Fyzikálne procesy s kvantovou náhodnosťou

Základným zdrojom náhodnosti sú v tomto prípade mechanizmy kvantovej fyziky na atómovej a subatómovej úrovni. Príkladom takýchto dejov môže byť:

- rozpad jadra rádioaktívnych prvkov,
- výstrelový šum – z ang. *Shot/Poisson noise* [3],
- prechod fotónov cez polo-priesvitné zrkadlo [3], [4],
- a iné.

Na overenie skutočnej náhodnosti dát sa v tomto prípade používajú tzv. **Bellove testy** [5]. Za spomenutie určite stojí aj tzv. **Geigerovo počítadlo** (z ang. *Geiger counter*). Tento prístroj sa požíva na zisťovanie ionizujúceho žiarenia atómov. Pri pripojení k počítaču môže slúžiť tiež ako zdroj náhodných dát.

1.1.2 Fyzikálne procesy bez kvantovej náhodnosti

Základ týchto dejov tvoria tepelné procesy. Detekcia je v tomto prípade jednoduchšia ako pri kvantových javoch. Uvedené deje sú náchylnejšie na útok, pri ktorom sa napríklad zníži prevádzková teplota daného systému na nižšiu, ako pri ktorej je zariadenie na tvorbu náhodných dát, schopné pracovať správne. Príkladom tepelných dejov je **tepelný šum**, ktorý vytvára rezistor. Ten je zosilnený tak, aby poskytoval náhodný zdroj napätia. Okrem spomenutého sa používa aj **atmosferický** [6] a **lavínový šum** (z ang. *Avalanche noise*) [7]. Pomerne jednoduché a nepredvídateľné deje sú tiež:

[4] Z ang. *True Random Number Generator*.
[5] Z ang. *Hardware Random Number Generator*.

- **posun hodín**[6] [8],
- **nestabilita hodín**[7] [9].

1.2 Deterministické RNG

DRNG[8], resp. DRBG[9], je taký generátor náhodných čísel/bitov, ktorý potrebuje na vygenerovanie výstupu prístup k zdroju náhodnosti, minimálne pri spustení – *inicializácií*. Následne generátor aplikuje pripravený algoritmus. Výstupom je postupnosť bitov, vytvorená na základe tajnej počiatočnej hodnoty (z ang. *seed*). Tento pojem charakterizujeme ako reťazec bitov, ktorý sa používa pri inicializácií kryptografických nástrojov.

Z vyššie uvedeného vyplýva, že tento typ generátorov vytvára **pseudonáhodné**, teda nie celkom náhodne dáta. Kvôli tomu sa takéto generátory často označujú ako PRNG (z ang. *Pseudo-Random Number Generator*) [10]. V uvedenom dokumente sa obdobne nachádzajú odporúčania a štandardizačný opis použitých mechanizmov, stavov a funkcií pre deterministické generátory. Ďalším, ale nie menej podstatným faktom ostáva, že celý proces generovania dát je podmienený počiatočnou hodnotou **seed**. Tento údaj predstavuje najzraniteľnejšie miesto týchto generátorov. Ak útočník získa túto bitovú postupnosť, tak na základe predpísaného postupu dokáže presne zreprodukovať rovnakú výslednú postupnosť bitov.

1.2.1 Pseudo-náhodné a kryptografický bezpečné RNG

V súčasnosti existuje veľa voľne dostupných internetových zdrojov kde sú zverejnené zoznamy pseudo-náhodných generátorov spoločne s kladmi i zápormi konkrétneho typu. Preto v tejto práci nebudeme pokračovať v tomto smere. Vývoj v oblasti výpočtovej techniky a technológií samotných, však v priebehu poslednej dekády rapídne pokročil. Aj dôsledkom toho je nutnosť mať k dispozícií generátory, ktorých kvalita výstupu nie je ľahko napadnuteľná, respektíve prelomiteľná. Na základe tejto myšlienky a dokumentu [11], následne delíme PRNG na:

- bežné PRNG,
- kryptograficky bezpečné PRNG – CSPRNG[10].

[6] Z ang. *Clock drift*.
[7] Z ang. *Clock jitter*.
[8] Z ang. *Deterministic Random Number Generator*.
[9] Z ang. *Deterministic Random Bit Generator*.
[10] Z ang. *Cryptographically Secure PRNG*.

Hlavným cieľom CSPRNG je vytvoriť výstupné dáta, ktoré sú kvalitou takmer na nerozoznanie od TRNG. Kryptograficky bezpečné generátory musia spĺňať 3 základne podmienky:

- úspešné absolvovanie štatistických testov,
- úspešný The next bit test [12],
- odolnosť voči narušeniu stavu – z ang. *State compromise extensions*.

Pod posledným pojmom sa rozumie prípad, v ktorom útočník zistí ľubovoľný zo stavov daného generátora. Napriek tejto znalosti nebude možné zistiť predchádzajúci ani budúci stav a teda zostaviť pôvodné dáta. V tejto súvislosti sa zvyknú používať aj termíny tzv. „***Spätná/Do-predná bezpečnosť***“, ktoré lepšie popisujú danú bezpečnostnú podmienku.

Spätná bezpečnosť[11], opisuje zabezpečenie pre prípad zistenia ľubovoľného stavu v určitom čase. Následne však nie je možné určiť akýkoľvek budúci výstup. Tento problému rieši dostatočne silný zdroj náhodnosti. Údaje v jednotlivých stavoch sa následne periodicky menia, resp. aktualizujú pomocou tejto náhodnosti.

Druhý prípad, teda **do-predná bezpečnosť**[12], na druhej strane hovorí o bezpečnosti pri zisťovaní stavu generátora smerom k zdrojovým dátam. Zjednodušene povedané, ak útočník zistí niektorý zo stavov generátora, tak aj napriek tejto znalosti nebude možné zistiť predchádzajúci stav a zároveň aj dáta. Tento prvok bezpečnosti sa dá ľahko zaručiť tým, že funkcia, ktorá posúva stav dopredu, bude jednosmerná. Príkladom sú hašovacie funkcie.

Označenie „kryptograficky bezpečný RNG“ aj napriek splneniu uvedených podmienok nie je veľmi jednoduché získať. Dôsledkom toho v praxi často trvá roky testovania, kým certifikačná organizácia označí algoritmus ako CSPRNG.

Uplatnenie kryptograficky bezpečných RNG je konkrétne v oblasti kryptografickej bezpečnosti. Takéto generátory sa uplatňujú napríklad pri generovaní:

- kľúčov pre kryptografické algoritmy – šifrovanie/dešifrovanie,
- inicializačných vektorov [13],
- maskovacích hodnôt [14],
- jednorázových čísel – z ang. *nonce* [15],
- dodatočných vstupov – z ang. *salt* [16].

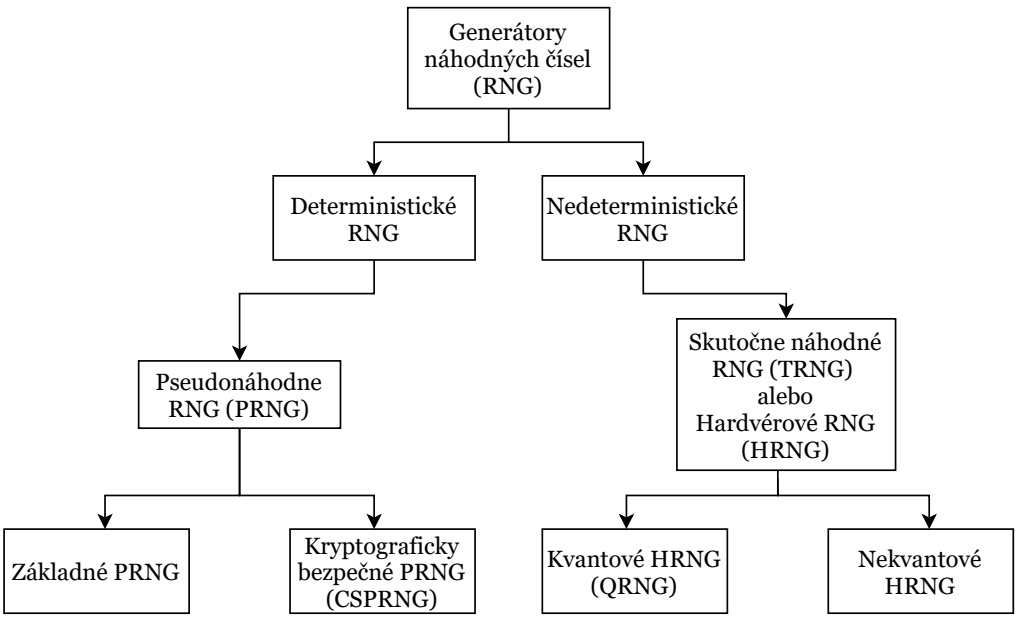
[11] Z ang. *Backward security*, tiež známa ako *break-in recovery*.
[12] Z ang. *Forward security*.

Bezpečnosť kryptografického systému je hodnotená práve na základe veľkosti náhodne vygenerovaných kľúčov. V súčasnosti dizajn väčšiny CSPRNG vieme rozdeliť do troch kategórií. Jednou z nich je skupina generátorov založená na kryptografických primitívoch. Konkrétne tieto pracujú na princípe hašovacích funkcií, blokových a prúdových šifier, ktoré sú vysvetlené v dokumente [17, kap. 5]. Ďalší typ návrhu je založený na základe zložitých matematických problémov. Hádam najznámejším je tzv. Blum Blum Shub generátor – BBS, ktorý je predmetom opisu práce [18]. Posledný dizajn charakterizujeme slovami – **špeciálny návrh**. Do tejto kategórie zahŕňame algoritmy, ktorých dizajn bol vyvinutý, resp. navrhnutý špeciálne pre splnenie vyššie spomenutých podmienok CSPRNG. Príkladom takýchto generátorov sú:

- Fortuna – používa ho macOS aj OS Linux – [19],
- Rivest šifra – ozn. RC – [20, kap. 3.1.1],
- Pokročilý šifrovací štandard – ozn. AES[13] – [21].

1.3 Súhrnná klasifikácia generátorov

Na základe vyššie uvedených skutočností je evidentné, že v tejto problematike dochádza v počítačovej sfére k častej zámene pomenovaní. Schéma číslo 1.1 znázorňuje rozdelenia opísané v tejto kapitole.

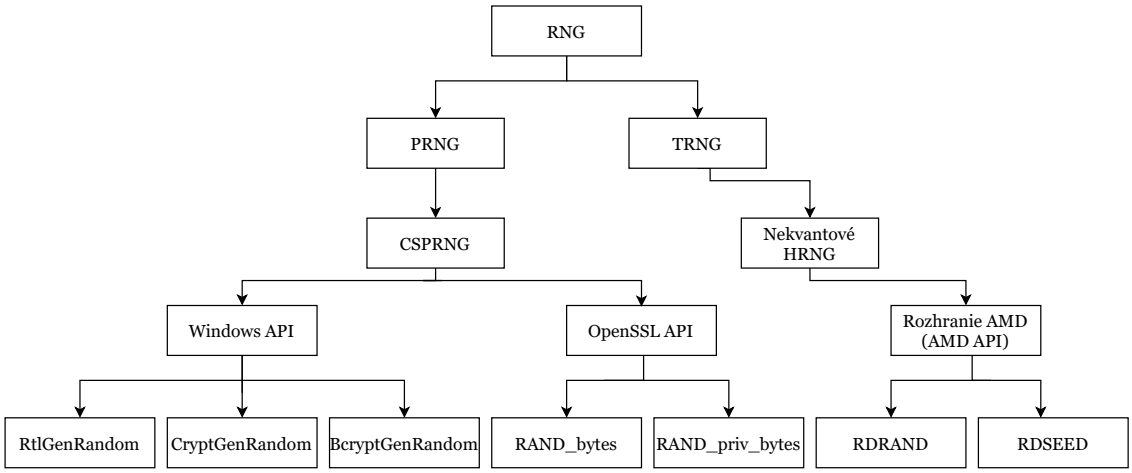


Obr. 1.1: Schéma rozdelenia RNG

[13] Z ang. *Advanced Encryption Standard*.

V tejto práci sa zameriame na generátory, ktoré sú vhodné pre kryptografické účely. Presnejšie aplikujeme naše meracie a testovacie metódy na rozhrania operačného systému Windows, ktoré sprostredkujú tieto služby na počítači. Okrem nich opíšeme a otestujeme aj iné možnosti generovania náhodných dát, ktoré má používateľ na tejto platforme. Vyššie uvedené podmienky spĺňajú len CSPRNG a HRNG.

Pri hardvérových generátoroch sa pri experimentoch zameriame iba na nekvantové. Dôvodom je, že dáta chceme vygenerovať sami, bez nutnosti použitia prostriedkov tretích strán. Schéma číslo 1.2 znázorňuje RNG a rozhrania, ktoré sú predmetom skúmania tejto práce. Posledné bloky znázorňujú konkrétne funkcie jednotlivých rozhraní, pričom ich opis je obsahom kapitoly 5.

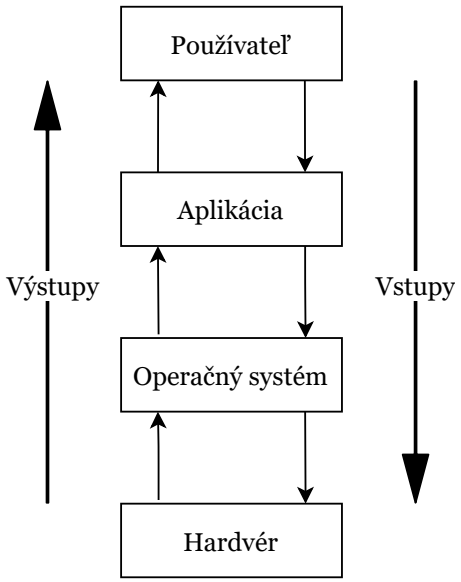


Obr. 1.2: Schéma rozhraní použitých v práci

2 Operačný systém Windows

Pred analýzou konkrétnej implementácie je nutná definícia niektorých pojmov. Prvým bude samotný OS.

Operačný systém je komplexný program – softvér, ktorý spravuje hardvér počítača. Tvorí základ pre aplikačné programy. Pracuje ako sprostredkovateľ medzi používateľom počítača a jeho hardvérom. Inými slovami, prácu tohto systému by sme v bežnej pracovnej sfére vedeli prirovnať k činnosti riaditeľa, respektíve manažéra spoločnosti. Jeho úlohou je správne a efektívne prerozdelenie práce – *činnosti aplikácií*, medzi svojich zamestnancov – *súčasti hardvéru*, na základe aktuálneho dopytu – *interakcií používateľa*.



Obr. 2.1: Úrovne počítača

Aby sa zabezpečila správna funkcionalita, tak OS musí riešiť množstvo s tým súvisiacich problémov. Ide napríklad o **plánovanie úloh**[1], **správu pamäte zariadenia**, **monitorovanie hardvérového príslušenstva**, **spracovanie vstupov a výstupov z/na príslušné zariadenia** (klávesnica, myš, ...), a iné. Dôležitým aspektom pri návrhu alebo voľbe OS je závislosť od miesta jeho nasadenia. Niektoré

[1] Z ang. *Task sheduling*.

operačné systémy sú navrhnuté so zameraním na jednoduché používateľské použitie. Ide o tzv. *user-friendly*, teda používateľsky prívetivé systémy. Ich použitie je bežné a príkladom môže byť samotný OS Windows 10. Ďalšiu kategóriu tvoria systémy zamerané na poskytnutie čo najlepšej efektívnosti nejakej služby. Príkladom sú serverové zariadenia. Väčšinou je však nutná vyššia odbornosť pri práci s týmito OS. V súčasnosti má už používateľ k dispozícii aj vysoko efektívne, a zároveň aj používateľsky prívetivé operačné systémy. Viac pozornosti k tejto problematike je venované v [22] a [23].

Súčasťou každého dobrého operačného systému je implementácia kryptografických modulov a ich aplikačných rozhraní (ďalej API). Ich súčasťou sú okrem iných aj funkcie, ktoré vykonávajú služby RNG. **Náhodné dáta** sú dôležitou a zároveň potrebnou súčasťou viacerých kryptografických algoritmov, respektíve systémov. Tie sú použité **na zabezpečenie** samotného **systému** a **sieťovej komunikácie** s inými zariadeniami. Bezpečnosť proti útokom je teda priamo úmerná kvalite náhodných dát a správnej implementácii kryptografických algoritmov vo vnútri týchto modulov. Okrem RNG sú ďalším príkladom obsahu kryptografických modulov tzv. hašovacie algoritmy, ktoré majú v kryptografii taktiež široké uplatnenie. Najznámejšími a v súčasnosti najpoužívanejšími sú algoritmy z rodiny SHA[2]. Viac informácií k tejto problematike sa uvádza v [24].

V súvislosti s generovaním náhodných dát je potrebné definovať pojem – **náhodnosť**, respektíve **entropia**[3]. Národný inštitút štandardov a technológií NIST v dokumente [25] a [26] definuje tento pojem. Následné vety budú parafrázou definícií v uvedených publikáciách. **Entropia** je matematický pojem pre náhodnú premennú X. Jej hodnota nám určuje množstvo vopred očakávaných informácií, poskytnutých zdrojom tejto premennej. Hodnota entropie je vždy viazaná na zdroj. Jej znalosť je výsledkom pozorovania, respektíve analýzy zdroja tejto náhodnosti. Okrem uvedenej slovnej definície je definovaná aj matematická formulácia dostupná na [27]. Pre pochopenie súvislostí nám vystačí aj slovná definícia.

2.1 História kryptografických API v OS Windows

Operačný systém Windows bol prvýkrát implementovaný 27. júla v roku **1993**. Niesol označenie „NT – New Technology“. Od vzniku až po súčasnosť došlo k viacerým viditeľných grafickým zmenám. Priebežným úpravám sa nevyhla ani

[2] Z ang. *Secure Hash algorithm*.
[3] Z ang. *Entropy*.

architektúra systému. Dôležité z hľadiska obsahu práce sú roky **1996** a **2006**.

2.1.1 CryptoAPI

V roku 1996 bola publikovaná novú verziu OS – Windows NT 4.0, ktorej obsahom bolo tzv. **rozhranie na programovanie kryptografických aplikácií**[4]. Pre jeho názov sa zaužívala skratka **CryptoAPI**, resp. **CAPI**. Oficiálny návrh použitého riešenia v rozhraní však spoločnosť Microsoft nezverejnila.

Základom kryptografického modulu sú tzv. *kryptografické primitíva*. Tento pojem trochu rozšírime. Pomenovanie **kryptografické primitíva** označuje jednotlivé nízko-úrovňové algoritmy, ktoré sa **často používajú** v počítačovej bezpečnosti pri rôznych kryptografických protokoloch alebo aplikáciach. Realizujú ich dynamické knižnice (`.dll` súbory). Tieto algoritmy úzko spolupracujú s tzv. **poskytovateľmi kryptografických služieb** (ďalej CSP).

CSPs predstavujú nezávislé softvérové knižnice, ktoré slúžia na kódovanie a dekódovanie kryptografických algoritmov. Inými slovami implementujú kryptografické algoritmy a štandardy. Pred každým použitím musí byť zvolený CSP digitálne podpísaný spoločnosťou Microsoft, pričom tento podpis je overovaný pri každom načítaní daného poskytovateľa. Proces overovania digitálneho podpisu prebieha aj periodicky pri používaní CSP. Týmto spôsobom je vykonané **zabezpečenie systému** pred vírusmi a možnými útokmi na kryptografický modul. Viac informácií je dostupných v [28].

Primitíva tvoria základ pre správne fungovanie ďalších funkcií alebo programov. V tejto súvislosti uvádzame pojem **kryptografické aplikácie**. Ten predstavuje označenie pre algoritmy, ktoré zabezpečujú bezpečný prenos a utajenie údajov. Uvedené programy realizujú dva základné úkony. Na základe toho rozdeľujeme aj funkcie opisovaného CAPI, na:

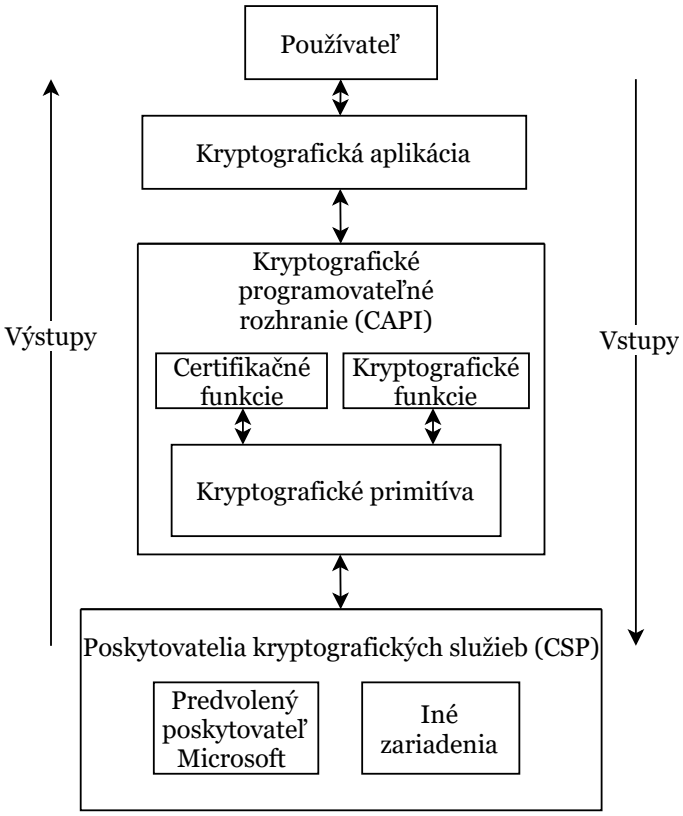
- certifikačné,
- kryptografické.

Príkladom použitia certifikačných je proces autentizácie[5] pomocou digitálneho certifikátu. V prípade druhého bodu je typické šifrovanie a dešifrovanie dát. Vyššie uvedené rozdelenie funkcií rozhrania na základe ich využitia, je znázornené pomocou schémy číslo 2.2. Každá z nich slúži na nejakú operáciu. Vykonávanie jednotlivých operácií je zabezpečené postupným volaním základných algoritmov – kryptografických primitív.

[4] Z ang. *Cryptographic Application Programming Interface*.

[5] Autentizácia – proces overenie identity.

CryptGenRandom je funkcia z kategórie primitív a slúži na generovanie kryptograficky bezpečných náhodných dát. Výstupy sú následne použité napríklad ako zdroj tajnej hodnoty – seed.



Obr. 2.2: Schéma architektúry CryptoAPI v OS Windows NT 4.0

Pomocou analýz boli v minulosti odhalené rôzne chyby a zraniteľné miesta CAPI implementácie. **Nedostatky** boli zistené aj pri CSPRNG. CryptGenRandom používal na generovanie výstupu systémovú entropiu a hashovací algoritmus SHA-1´[6] v kooperácii s RC4 šifrou ([20]). Spomenuté riešenie však neposkytovalo žiadnu spätnú ani do-prednú bezpečnosť, ktorá bola opísaná pri CSPRNG. Podrobný opis použitých metód, nástrojov, útokov a dosiahnutých výsledkov analýzy tohto generátora je dostupný v dokumente [29].

2.1.2 Kryptografické rozhranie novej generácie

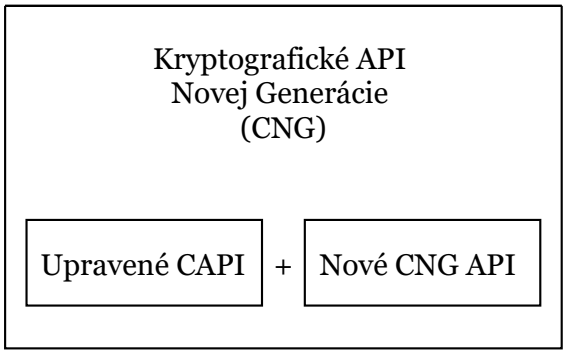
Vzhľadom na problémy návrhu kryptografického rozhrania CryptoAPI bola **nutná** jeho **inovácia**. Ta bola uverejnená v roku **2006**, spoločne s novou verziou OS – Windows Vista. Rozhranie zmenilo názov na kryptografické API novej generácie (ďalej CNG[7]). Dôležité je, že došlo k rozšíreniu a taktiež úprave pôvodného

[6] Rozdiel oproti klasickému SHA-1 bol v zmenenom poradí inicializačných vektorov.
[7] Z ang. *Cryptography API: Next Generation*.

rozhrania. Dôsledkom toho sa zabezpečila aj spätná **kompatibilita** voči už nasadeným systémom. Schéma 2.3 znázorňuje uvedenú skutočnosť.

Rozhranie CNG prinieslo množstvo nových funkcií, pričom ich konštrukcia bola vytvorená na základe noriem definovaných v dokumente FIPS 140-2[30]. Príkladmi sú:

- nový spôsob konfigurácie rozhrania,
- nové kryptografické rozhranie pre jadro systému – *z ang. kernel-mode*,
- oddelenie úložiska dát od operácií algoritmov,
- vylepšený proces izolácie od operácií s dlhodobým kľúčom,
- vylepšenie v oblasti uloženia, obnovenia, importu a exportu kľúčov,
- podpora kryptografie pomocou eliptických kriviek,
- podpora piatich režimov[8] v šifrovacom rozhraní, pri šifrovaní pomocou symetrických blokových šifier
- a iné.

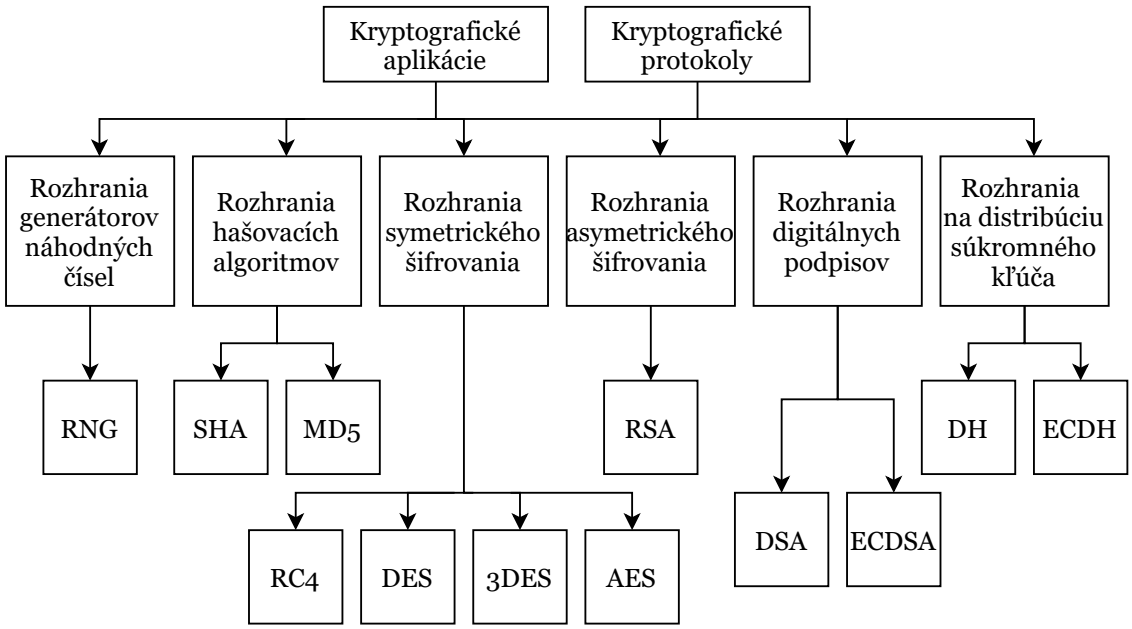


Obr. 2.3: Schéma architektúry po inovácií

Zmeny nastali aj v oblasti RNG. V rozhraní je k dispozícií možnosť zmeniť predvolený generátor náhodných čísel. Taktiež je možné zameniť ho v CSP. Avšak táto zmena nie je možná v prípade hlavného poskytovateľa týchto služieb, ktorým je Microsoft Base CSP. Vďaka tomu je tiež možné špecifikovať určitý typ generátorov k určitému volaniu príslušnej funkcie.

V prípade CNG rozhrania bola uverejnená aj architektúra, ale len niektorých častí. Spoločnosť v oficiálnej dokumentácii k rozhraniu zverejnila návrh i použité algoritmy. Na ich základe je vytvorená schéma 2.4.

[8]Elektronická kódovaná kniha (ECB), Zreťazené blokové šifrovanie (CBC), Šifrovanie pomocou zašifrovaného textu CFB, CBC čítačový režim (CCM) a Galoisov čítačový řezim (GCM).



Obr. 2.4: Schéma architektúry CNG primitív

2.2 Aktuálny prístup k rozhraniu RNG

Postupom času dochádzalo k úprave a vylepšeniam API. Dokument [31, kap. 2], opisuje prístup ku generátoru náhodných čísel v CNG module[9]. Základné mechanizmy systému boli lepšie vysvetlené práve v skorších verziách. Nasledujúce myšlienky sú parafrázami vyššie uvedenej publikácie a dokumentu [32]. Publikácia [32] opisuje obdobne prístup k RNG, avšak v aktuálnom OS Windows 10.

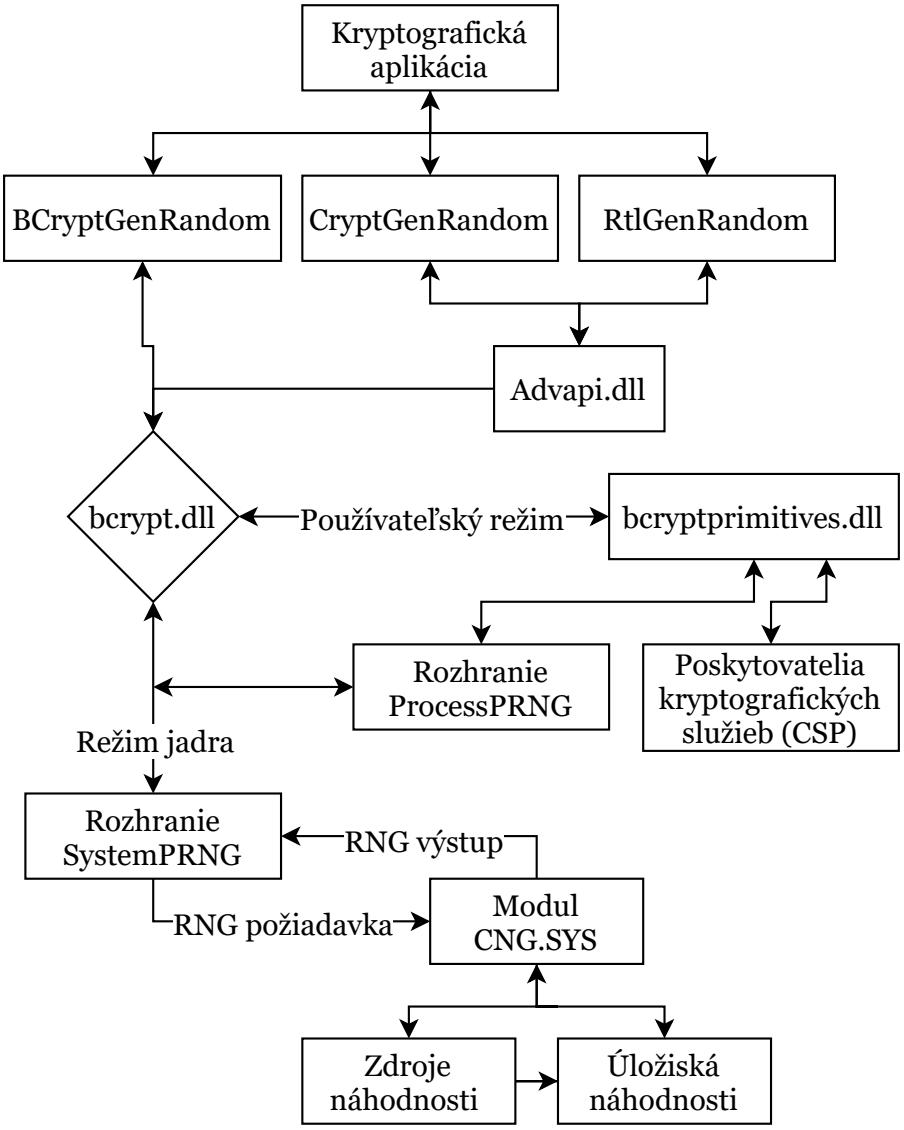
Funkcie rozhrania sú prístupné v dvoch režimoch:

- režim jadra – z ang. *kernel mode*,
- používateľský režim – z ang. *user mode*.

V oboch režimoch je prístup k službám rozhrania identický. Rozdiel je v spôsobe, účeloch volania a prerušeniach systému. Režim jadra je využívaný výhradne aplikáciami a komponentami jadra OS[10]. Na realizáciu tohto úkonu slúži rozhranie **SystemPRNG**, ktoré je súčasťou ovládača, resp. modulu – **CNG.SYS**. V prípade používateľa je prístup k generátoru zložitejší. Je riešený pomocou rozhrania **ProcessPRNG**. To je súčasťou dynamicky pripojenej knižnice – **bcryptprimitives.dll**. Tá následne získa prístup k softvérovému RNG a entropií pomocou SystemPRNG v module CNG.SYS. Schéma 2.5 opisuje tento prístup už aj s primárne určenými funkciami a rozhraniami, ktoré aplikácie volajú pri žiadosti o náhodné dáta.

[9] Prístup bol demonštrovaný v OS Windows 8.

[10] Prístup používateľa k tomuto módu je možné realizovať implementáciou vlastného modulu.



Obr. 2.5: Schéma prístupu kryptografických aplikácií k náhodným číslam

Používateľ má možnosť zvoliť iného CSP[11] na získanie prístupu k softvérovému RNG. Teda OS nie je nevyhnutne odkázaný iba na prednastavené riešenie a poskytuje užívateľovi väčšiu voľnosť pri voľbe CSP.

2.3 Infraštruktúra použitých PRNG

OS Windows používa na generovanie náhodných čísel viacero typov a označení pre jednotlivé PRNG. V tejto podkapitole je uvedená ich charakteristika. Informácie boli čerpané z [32, str. 1].

Ako prvý opíšeme tzv. **základný PRNG** [12]. Na všetko generovanie náhodných

[11] Opísaný v 2.1.1.

[12] Z ang. *Basic PRNG*.

čísel je použitý algoritmus AES-256 v čítačovom režime, ktorý bol spomenutý v prvej kapitole. Jeho konštrukcia je založená podľa normy SP800-90 [2] a generátor nesie označenie AES_CTR_DRBG. Zároveň patrí do kategórie CSPRNG. Poskytuje ochranu na úrovni 256-bitov. Na inicializáciu tajnej počiatočnej hodnoty (ďalej ako seed) používa funkciu `df()`.

Ďalšie označenie, ktoré sa používa v systéme je tzv. **buffer-ovaný PRNG** [13]. Základný RNG nie je používaný priamo, ale prostredníctvom ochrannej vrstvy [14]. Tá poskytuje generátoru niekoľko vlastností. Konkrétne:

- pridáva malý zásobník – ďalej buffer,
- uzatvára podporu viac-vlaknovosti (z ang. *multi-threading*),
- poskytuje verziu seed-u.

Ukladanie dát do buffer-a sa realizuje priamo a zlepšuje výkon najmä pri generovaní menšieho množstva čísel. Jeho veľkosť je 128 bajtov. Plní sa vždy po použití uvedenej veľkosti dát. Predstavme si, že obsahuje napríklad 10 bajtov. Systém vyšle požiadavku na väčší objem dát ako sa nachádza v buffer-i. V tomto prípade sa použijú spomínané bajty uložené v buffer-i a zvyšok sa dodá po doplnení zásobníka. Tento úkon zabezpečí vyššie opísaný základný generátor. Po použití bajtov dochádza k vynulovaniu príslušných pozícií. Aktualizácia tejto vyrovnávacej pamäte sa uskutočňuje obdobne pri každej aplikácii nového seed-u (ďalej ako **reseed**). Všetky údaje sa prepíšu. V prípade, že požiadavka o náhodné dáta prekračuje veľkosť 128 bajtov, tak generovanie výstupov je uskutočnené priamo základným PRNG. Prístup k stavu buffer-ovaného PRNG je zablokovaný. Tým sa zabezpečuje, že ostatné vlákna nemôžu čítať ani modifikovať rovnaké stavy v rovnakom čase.

Koreňový PRNG[15]. S takto označeným generátorom sa stretávame pri tvorbe a ukladaní náhodnosti v entropickom systéme[16]. Je súčasťou CNG.SYS modulu a ide o už vyššie opísaný buffer-ovaný PRNG.

V režime jadra je použitý opäť buffer-ovaný PRNG s daným stavom pre logický procesor. To znamená, že ak máme 8 jadrový procesor s podporou tzv. **viacerých vlakien**[17], tak máme k dispozícií 16 logických procesorov. V prípade, že aplikácia požiada o náhodne dáta v režime jadra, tak algoritmus skontroluje, ktorý

[13] Z ang. *Buffered PRNG*.
[14] Z ang. *Wrapping layer*.
[15] Z ang. *Root PRNG*.
[16] Charakterizovaný v podkapitole číslo 2.4.
[17] Z ang. *Hyperthreading*.

z procesorov je aktívny a overí stav generátora. Ak bol generátor alokovaný v tomto procesore, tak následne sa použije na generovanie náhodných dát. V opačnom prípade dochádza k návratu na pôvodný procesor. Tento CPU sa následne pokúša o alokovanie nového stavu PRNG. Dodanie seed-u zabezpečí zdrojový generátor. Ak proces alokácie nebude úspešný, tak o službu generovania sa postará zdrojový generátor. V praxi však k neúspešnej alokácii nedochádza a náhodné dáta dodá PRNG z CPU. Anglické označenie vyššie opísaného PRNG v OS Windows 10 je tzv. *Kernel per-processor PRNG*.

Pre používateľa sa mení iba spôsob prístupu ku generátoru. Každý proces vytvorený užívateľom, ktorý vytvorí požiadavku o náhodné dáta, je spracovaný dynamickou knižnicou bcryptprimitives.dll. Tá vytvorí tzv. *ProcessPRNG*. Ním žiada režim jadra o generovanie náhodných bitov, prostredníctvom vyššie opísaného spôsobu. Tento celý proces dokáže zlyhať jedine v prípade, že načítanie knižnice bcryptprimitives nebude úspešné. V tomto prípade je celý proces zrušený. V praxi sa na bežiacom počítači tento postup opakuje pri každom procese, ktorý vyžaduje náhodné dáta.

Výsledný počet celkovo aktívnych stavov PRNG je možne vyjadriť ako súčin $(N+1)*(M+1)$. N nám označuje množstvo jadier systému a M počet spustených procesov, ktoré vyžadujú náhodné dáta. Takýto súčin nám vytvára celkom veľký výsledok aktívnych PRNG s príslušným stavom. To však už v súčasnosti nie je problém, pretože výkonné počítače obsahujú dostatok pamäte, aby to bez problémov zvládli. V odbornej literatúre sa definuje vyššie uvedený prístup jedného procesu k generátoru pomocou anglického výrazu *Process base PRNG*. Pohľad z vyššej perspektívy definuje výraz *Process per-processor PRNG*.

2.4 Entropický systém

Nasledujúci opis vychádza z dokumentu [32, str. 6]. Dodanie náhodnosti má za úlohu špeciálny nástroj – **Entropický systém**. V OS Windows 10 pozostáva z niekoľkých častí, resp. komponentov. Tvoria ho konkrétne:

- zdroje náhodnosti – z ang. *Entropy sources*,
- úložiská náhodnosti – z ang. *Entropy pools*, slúžia ako úložisko dát zdrojov entropie, ktoré sa následne používajú ako seed hodnota pre PRNG.
- obnovovacia, resp. resetovacia logiky seed-u [18]. Jej úlohou je rozhodovať ako

[18] Z ang. *Reseed logic*.

a kedy dôjde k aktualizácii seed hodnoty v zdrojovom PRNG z polí náhodnosti.

Proces reseed-u zdrojového generátora sa opakuje periodicky pomocou časového plánovača. Spúšťa sa jednu sekundu po inicializácii OS a následne každým trojnásobkom predchádzajúcej hodnoty (3, 9, 27, ...). Koniec nastane po dosiahnutí časového limitu. Ten je nastavený na jednu hodinu teda 3600 sekúnd. Plánovanie je nastavené tak, aby k reseed-u došlo iba v prípade, ak je CPU prebudený. Dôvodom sú vysoké energetické nároky na prebudenie.

2.4.1 Zdroje náhodnosti

Ich úlohou je poskytovať náhodné dáta. Výstup je následne uložený v úložiskách entropie.

Súčasťou jadra rozhrania systému je rozhranie na vytváranie nových zdrojov entropie. API vo všeobecnosti podporuje dva typy zdrojov:

- nízke – z ang. *low pull*,
- vysoké – z ang. *high pull/push*.

Rozdiely v ich spracovaní sú minimálne. Nízke zdroje označujú nepodmienené udalosti ako je napríklad pohyb myši. Tie však môžu byť reprodukovateľné. Vysokokvalitné náhodné dáta zabezpečujú vysoké zdroje. V prípade potreby dokážu ihneď (okamžite) poskytnúť entropiu všetky z uvedených zdrojov. Rozdiel medzi typom **pull** a **push** je v tom, že prvý spomenutý neobsahuje vlastnú logiku časovania na obnovu entropie. Tieto zdroje sú dodatočne informované pri každom použití zdrojovým RNG.

Prerozdelenie dát v rámci jednotlivých úložísk entropie sa riadi pomocou tzv. z ang. *Round-Robin*[19] plánovania. Pri vysokých zdrojoch sa vždy prvých 32 bajtov ukladá do prvého úložiska entropie a zvyšné sa uložia do poľa, ktoré nasleduje podľa plánu. Viac informácii k zdrojom náhodnosti je dostupných v [32, str. 8].

Použité zdroje náhodnosti v OS Windows 10 [32, str. 8]

Windows 10 používa viacero zdrojov entropie za účelom poskytnúť dostatočnú entropiu v každej situácii. V odbornej literatúre sa uvádzajú tieto zdroje.

[19] `https://en.wikipedia.org/wiki/Round-robin_scheduling`.

- **Časovače prerušení** – primárny zdroj. Každé prerušenie je spracované pomocou TSC[20]. Je to počítadlo, ktoré beží v procesore. V x86 a x64 architektúrach sa na získanie jeho hodnoty používa inštrukcia RDTSC ([33], [34]). V rámci inštrukcie dochádza k rotovaniu a následnému xorovaniu hodnôt predtým než sú poskytnuté ďalej.

- **Štart** – pri štarte systému dochádza k zhromažďovaniu TSC údajov pred načítaním CNG modulu. Tento proces typicky sprevádza niekoľko stoviek prerušení. Po inicializácii tohto ovládača sa údaje použijú hneď ako seed pre koreňový generátor. Teda polia entropie sú v toto kroku vynechané. Tento krok zabezpečuje, že systém má po štarte k dispozícii dostatok entropie.

- **TPM**[35] – pri štarte dodáva 40 bajtov. Po registrácii zdroja poskytuje 64 bajtov pri každom reseede. Ten sa opakuje raz za 40 minút.

- **RDRAND/RDSEED** – procesorové inštrukcie na tvorbu náhodných dát.

- **Seed súbor** – register v podsystéme registrov. Zapisuje sa pri štarte a vypnutí systému. Vytvára ho OS a je použitý pri ďalšom štarte systému (z ang. *boot*). Nová hodnota je zapísaná pomocou výstupu systémového PRNG. Jeho veľkosť je 64 bajtov. Ak však dôjde k netradičnému vypnutiu OS, tak zápis prebieha aj počas behu. Štyri minúty po obnovení systému dochádza každým trojnásobkom tohto času k periodickému zápisu až do dosiahnutia limitu ôsmich hodín. Pri vypnutí dochádza k použitiu všetkej uloženej náhodnosti zdrojovým generátorom. Výstup je uložený v tomto súbore a pri nasledujúcom štarte použitý pomocou winload modulu.

- **Externý zdroj** – používateľ má možnosť pridať k zdrojom aj vlastný. Použije sa pri bootovaní systému pomocou winload modulu a po nasadení CNG.SYS ovládača sa tento zdroj po štarte odstraňuje.

- **ACPI-OEM0** – je to ACPI[21] [36] tabuľka s názvom OEM0[22]. Vytvára ju hypervízor Hyper-V. Jej obsahom je 64 bajtov náhodných dát.

- **Firmvérové údaje**.

- **UEFI**[37] protokol.

- **Čas spustenia**.

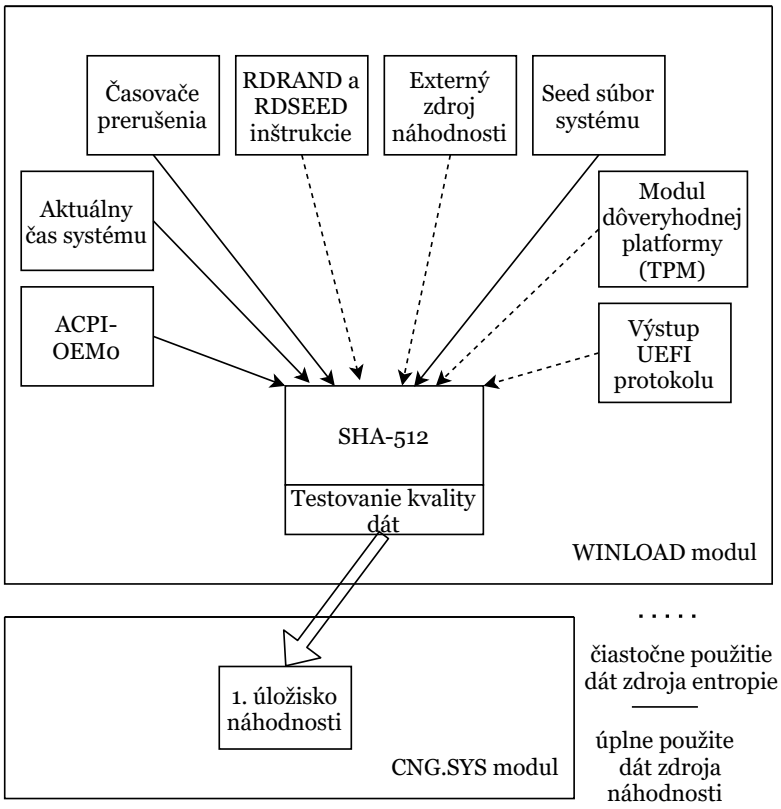
[20] Z ang. *Time Stamp Counter*.
[21] Z ang. *Advanced Configuration and Power Interface*.
[22] Z ang. *Original Equipment Manufacturer*.

2.4.2 Úložiská entropie – Entropy pools

Každé pole, respektíve úložisko entropie je implementované pomocou hašovacej funkcie SHA-512, ktorá patrí do rodiny SHA2[38]. To znamená, že všetky dáta tohto poľa sú výstupom SHA funkcie. Ten je pridaný na koniec daného poľa. Vstup hashovacej funkcie tvoria zdroje entropie. Pri štarte systému existuje len jedno takéto úložisko. Po dosiahnutí limitu reseed-u sa zapne možnosť viacerých, resp. dodatočných úložísk entropie. A povolí sa vytvorenie ďalšieho poľa. Dizajn viacnásobného úložiska entropie systému opisuje príloha v dokumente [32].

2.4.3 Štart systému

Pri štarte systému dochádza k prvotnému seedu. Uskutočňuje ho modul Winload pred štartom ovládača Ntoskml. Proces vytvorenia tajnej hodnoty je opísaný v schéme 2.6. Výstup týchto dát je použitý generátorom **AES-CTR-DRBG** podľa



Obr. 2.6: Schéma vytvorenia prvotného seed-u pri inicializácií systému

normy SP 800-90[2]. 48 bajtov výstupu je použitých modulom CNG.SYS a zvyšok je presmerovaný do jadra na budúce použitie. Spomenuté bajty sú použité zdrojovým PRNG. Následne sa spúšťa vytváranie vlastnej entropie pomocou vyššie opísaných zdrojov. Informácie boli čerpané z [32, str. 8].

2.5 Zhrnutie bezpečnosti operačného systému

Aktuálne platnú validáciu FIPS 140-3, OS Windows 10[23] spĺňa aj napriek tomu, že kryptografické rozhranie obsahuje, v niektorých prípadoch aj používa, neschválené, resp. neodporúčané algoritmy. Tie sa v dnešnej dobe naďalej nepovažujú za kryptograficky bezpečné. OS však ponúka riešenie aj pre užívateľov, ktorí vyžadujú použitie aktuálne platných štandardov. Ide o tzv. **FIPS režim**. Ten je dostupný v každej edícií Windows-u 10, okrem Home verzie. Postup, ako spustiť daný režim, je dostupný na stránke spoločne so zoznamom systémov, ktoré sú schválené.

[23] Zoznam schválených produktov spoločnosti Microsoft je dostupný na stránke: `https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation`.

3 Metodika testovania dát a merania rozhraní

Pred samotným generovaním dát je nutné zvoliť meracie a testovacie metódy. V tejto kapitole ich popíšeme. Znázornené demo príklady zdrojových kódov sú obsahom prílohy A, spoločne s makefile-om.

Implementáciu rozhraní sme realizovali v programovacom jazyku C. Preklad do strojového kódu zabezpečil prekladač GCC vo verzii 10.2.0 (04. 03. 2021). **Overenie kvality dát** vykonáme pomocou NIST štatistickej testovacej sady (ďalej NIST STS), opísanej v podkapitole 4.2. Pri experimentálnych meraniach implementácií RNG rozhraní sme sa zamerali na tri údaje:

- čas vykonania samotnej implementácie – T_A,
- dobu vykonania, vrátane bežných úkonov[1] – T_B,
- priemerný počet cyklov API – ANC. Získa sa ako pomer súčtu všetkých vykonaných cyklov a počtu opakovaní volania testovanej funkcie. Pri výsledkoch uvádzame ANC iba z hodnôt meraní T_A.

Pomocou týchto nameraných hodnôt sme vypočítali priepustnosť dát. Použili sme vzťah (3.1).
Nech

$X = \{A, B\}$,
T_X – čas zvoleného procesu,
NI – počet opakovaných volaní[2],
BS – veľkosť buffer-a[3],
$V_D = NI * BS$ – celkový objem vygenerovaných dát,

[1] Ukladanie dát, overenie úspešnosti generovania, ...
[2] Z ang. *Number of Iterations*.
[3] Z ang. *Buffer Size*.

P_X – priepustnosť procesu,
potom,

$$P_X = \frac{V_D}{T_X}. \tag{3.1}$$

Uvedené experimenty boli aplikované na troch zariadeniach. Špecifikáciu prenosných počítačov znázorňuje tabuľka 3.1. Všetky notebook-y sme počas testovania pripojili do elektrickej siete. Režim napájania sme zmenili na „Vysoký výkon“.

Označenia hodnôt a meracích nástrojov, ktoré vznikli v tejto kapitole **sú použité pri interpretácii výsledkov**.

Komponenty	**Konfigurácia**		
	A – ASUS TUF A15	**B – ASUS Vivo15**	**C – LENOVO IdeaPad**
Model	F506IU-AL006T	X510UN-BQ148R	S540-15IML
Verzia OS	Win 10 Home; 64-bit.; v.20H2	Win 10 Pro; 64-bit.; v.2004	Win 10 Home; 64-bit.; v. 20H2
Zostava OS	19042.964	19041.928	19042.985
CPU	AMD Ryzen 7 Mobile 4800H	Intel Core i5-8250U	Intel Core i5-10210U
RAM	16 GB DDR4 2x1600 MHz	8 GB DDR4 1x2400 MHz	8 GB DDR4 2x1200 MHz
Úložisko	SSD OM8PCP3512F-AB	HDD MQ04ABF100	SSD SSDPEKNW512G8L

Tabuľka 3.1: Technická špecifikácia použitých počítačov

V tomto bode treba spomenúť určitú odchýlku experimentálnych meraní od skutočných hodnôt. Najväčším faktor, ktorý mohol spôsobiť chybu merania sú prerušenia operačného systému[4]. Uvedenému procesu je možné sa vyhnúť jedine implementáciou modulu pre prácu v režime jadra. Následne by sme vyhradili procesor pre vlastné účely. Kladom módu by mohol byť aj menší počet inštrukcií potrebných na vykonanie funkcionality. S týmto postupom sa však používateľ často nestretáva. Uvedený fakt je dôvodom, prečo sme sa pri meraniach zamerali iba na spúšťanie rozhraní v používateľskom režime. Pripomenieme však, že rozhrania v oboch režimoch poskytujú **identické služby**.

3.1 Časové meranie rozhraní

Pri meraní dĺžky vykonávania rozhraní sme implementovali Windows API:

- `QueryPerformanceCounter()` –QPC [39],
- `QueryPerformanceFrequency()` –QPF [40].

[4]Viď. napríklad elapsedTime v 3.1.

Informácie o funkciách sú dostupné vo forme webovej dokumentácie [41]. Použitie týchto rozhraní pri meraní znázorňuje zdrojový kód 3.1. Uvedeným postupom dokážeme odmerať čas vykonania algoritmu s rozlíšením nanosekúnd[5]. Presnosť metódy sa dá overiť jednoducho. Napríklad pomocou funkcie `Sleep()`[6]. Stačí ju vložiť do 18. riadku v zdrojovom kóde 3.1.

Zdrojový kód 3.1: Ukážka použitia QPC/QPF

```
1  #include<windows.h>
2
3  #define TIMER_INIT \
4    LARGE_INTEGER frequency; \
5    LARGE_INTEGER t1,t2; \
6    double elapsedTime; \
7    QueryPerformanceFrequency(&frequency);
8    // Use to start the performance timer
9  #define TIMER_START QueryPerformanceCounter(&t1);
10   // Use to stop the timer
11 #define TIMER_STOP \
12   QueryPerformanceCounter(&t2); \
13 elapsedTime=(double)(t2.QuadPart-t1.QuadPart)/frequency.QuadPart;
14
15 int main(){
16   TIMER_INIT
17   {TIMER_START
18     functionMeasurment(); // code for measurment
19   TIMER_STOP}
20   printf("Time␣of␣execution:␣%f␣sec\n", elapsedTime);
21   return 0;
22 }
```

Odporúčanie pri pretypovaní dát

V jazyku C môže pri pretypovaní dát dochádzať k zmene poradia vygenerovaných reťazcov. Dôvodom je zmena endianity[7], respektíve uloženia dát v pamäti. S týmto problémom sme sa stretli pri zmene z **unsigned int** na **unsigned char** vo funkcii `rand_s`. Dáta v pamäti boli uložené metodikou malý endián, ale správne

[5] API je možné implementovať s presnosťou piko-sekúnd. Viď [41, Using QPC in native code].
[6] ***Sleep*(*1000*)** reprezentuje jednu sekundu.
[7] `https://sk.wikipedia.org/wiki/Endianita`.

poradie vygenerovaných dát reprezentoval veľký endián. V tomto prípade bolo nutné vykonať konverziu endianity. Použili sme na to funkcie znázornené pomocou zdrojového kódu 3.2.

Zdrojový kód 3.2: Ukážka pretypovania premenných

```
#include<stdio.h>
#define IS_BIG_ENDIAN (!*(unsigned char *)&(uint16_t){1})
int main (){
  unsigned int a = 2343352;
  unsigned char b;
  if(!IS_BIG_ENDIAN) castUintToByte(a,b);
  else b=(unsigned char)a;
  printf("%lu",b);
  return 0;
}
```

3.2 Meranie počtu cyklov jednotlivých funkcií

Obdobne sme sa zamerali aj na počet cyklov jednotlivých funkcií. Tento údaj nám z pohľadu dlhodobého vývoja predstavuje kvalitnejšiu informáciu ako čas vykonania. Na základe týchto údajov vieme určiť napríklad či v priebehu času došlo k zefektívneniu algoritmov rozhrania[8]. Ďalším príkladom je určenie pomeru prerušení OS a mnoho iných. Meranie sme prvotne realizovali pomocou funkcie `cpucycles()`[9]. Zdrojový kód 3.3, ju definuje. Táto metóda však neposkytovala dostatočne presné výsledky.

Zdrojový kód 3.3: Meranie počtu cyklov pomocou funkcie cpucycles()

```
int64_t cpucycles(void){
  uint64_t result;
  __asm__ volatile(".byte␣15;.byte␣49;shlq␣$32,
␣␣␣␣%%rdx;orq␣%%rdx,%%rax"
  : "=a" (result) ::  "%rdx");
  return result;
}
```

Meranie počtu vykonaných cyklov sme, kvôli tomuto faktoru, uskutočnili podľa

[8] Zmenší sa počet cyklov, potrebných na vykonanie.

[9] Funkcia prebraná z git archívu: `https://github.com/newhopecrypto/newhope/tree/master/ref`.

metód v Intel dokumente [42, kap.3.2.1]. Ukážkou aplikovaného riešenia je kód 3.4.

Zdrojový kód 3.4: Meranie počtu cyklov Intel metódou

```
#include <stdio.h>

//cpucyclesS -- Start measure
static __inline__ uint64_t cpucyclesS(){
  unsigned cycles_low, cycles_high;
  __asm__ volatile ("CPUID\n\t"
    "RDTSC\n\t"
    "mov␣%%edx,␣%0\n\t"
    "mov␣%%eax,␣%1\n\t": "=r" (cycles_high), "=r" (cycles_low)::
    "%rax", "%rbx", "%rcx", "%rdx");
  return  (((uint64_t)cycles_high << 32) | cycles_low );
}
//cpucyclesE -- End measure
static __inline__ uint64_t  cpucyclesE(){
  unsigned cycles_low, cycles_high;
  __asm__ volatile ("RDTSCP\n\t"
    "mov␣%%edx,␣%0\n\t"
    "mov␣%%eax,␣%1\n\t"
    "CPUID\n\t": "=r" (cycles_high), "=r" (cycles_low)::
    "%rax", "%rbx", "%rcx", "%rdx");
  return (((uint64_t)cycles_high << 32) | cycles_low );
}
int main(){
  uint64_t tick,tock;
  tick=cpucyclesS();
  codeForMeasurment();
  tock= cpucyclesE() - tick;
  printf("Executed:␣%llu␣cycles\n", tock);
  return 0;
}
```

4 Štatistické testovanie generátorov

Pri používaní RNG môže dôjsť k jeho vnútorným zmenám. Na overenie kvality generátorov náhodných čísel sa používajú statické testy náhodnosti. Väčšina z nich preveruje jednu alebo hneď niekoľko štatistických vlastností. Cieľom je odhaliť anomáliu bitov v danej postupnosti dát. Pre používateľa sú k dispozícii vo forme jednotlivých testov alebo štatistických testovacích sád. Takáto zbierka predstavuje implementáciu viacerých štatistických testov, pripravených na jednoduché použitie, modifikáciu a vyhodnotenie. Najznámejšími sú:

- **Dieharder**[43],
- **NIST STS**[25] – z ang. *The NIST Statistical Test Suit*.

Okrem vyššie uvedených je však možné vyhľadať, vytvoriť, a aj upraviť, respektíve optimalizovať rôzne implementácie štatistických testov. Dôkazom toho je aj vznik modifikovanej NIST testovacej sady – [44]. Podľa vyjadrení autorov uvedeného projektu sa im podarilo zefektívniť rýchlosť testovania o vyše 50 percent v porovnaní s originálom. Ďalšie príklady testovacích sád sú napríklad **ENT** [45], **TestU01** [46] a mnohé iné.

Je však nutné uvedomiť si, že úspešné absolvovanie ľubovoľného počtu štatistických testov nezabezpečuje kryptografickú bezpečnosť daného generátora. Dôvodom je, že niektoré z nástrojov na generovanie náhodných dát môžu byť pripravené tak, aby boli úspešné pri vykonávaní týchto testov. Avšak platí, že ak má byť RNG označený za CSPRNG, tak musí úspešne zvládnuť testovanie štatistickými testami.

4.1 Testovacia sada Dieharder

Sád Diehard vznikla v roku 2003 a je aj naďalej udržiavaná. Slúži na testovanie dát z RNGs. Používateľ môže pomocou sady generovať dáta a následne ich otestovať. Okrem toho je možné testovať aj vlastný súbor s náhodnými dátami.

Výstupom sady je výpis vo forme tabuľky s výsledkami použitých štatistických testov.

Aktuálna verzia – 3.31.1 (03. 06. 2020), pozostáva z:

- 17 Diehard,
- 3 NIST STS,
- 10 testov, autora tejto zbierky – *Róberta G. Brown-a*.

Podrobnejší opis týchto testov je dostupný online[1,2]. Inštalácia sady je jednoduchá najmä v prostredí OS Linux. Realizuje ju príkaz:
`sudo apt-get install -y dieharder`[3].
V prostredí systému Windows je potrebná zložitejšia konfigurácia.

4.2 NIST – Štatistická testovacia sada

Tento balíček bol vytvorený inštitúciou NIST. Pozostáva z 15 testov. Slúžia na overenie kvality výstupných dát z hardvérových, a aj softvérových generátorov náhodných čísel. V súčasnosti sa práve táto sada používa aj pri testovaní CSPRNG. Na rozdiel od spomenutej kolekcie *Dieharder* táto zbierka vyšla s podrobnou dokumentáciou [25]. Uvádzame stručný opis metód a použitých testov.

4.2.1 Obsah sady a opis implementovaných testov

Sada je pre používateľa dostupná vo forme archívu – „**sts-2.1.2.zip**“[4](09. 07. 2014). Jeho obsahom sú implementácie štatistických testov v jazyku C. **Makefile** na jednoduché vytvorenie spustiteľného programu a príklady vstupných dát z rôznych typov generátorov. Následné spustenie je opísané v [25, kap. 5.6] a pomocou ukážky 4.1.

Opis testov štatistickej sady:

1. **Frekvenčný test – *The Frequency/Monobit Test* [25, kap. 2.1]**

   Výsledok je určený pomerom jednotkových a nulových bitov v danej testovanej postupnosti. Úspešný je vtedy, ak sa obsah jednotiek v našich dátach

[1] `https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions`.

[2] `https://en.wikipedia.org/wiki/Diehard_tests`.

[3] Príkaz pre distribúciu Ubuntu.

[4] `https://csrc.nist.gov/CSRC/media/Projects/Random-Bit-Generation/documents/sts-2_1_2.zip`.

blíži k $\frac{1}{2}$ z celkového počtu. Odporúčaná dĺžka testovanej postupnosti je minimálne 100 bitov. Úspešné zvládnutie tohto testu je nutnou podmienkou pre ďalšie pokračovanie testovania.

2. **Blokový frekvenčný test – *Frequency Test within a Block*[25, kap. 2.2]**

   Podobný prvému zo spomenutej sady. Namiesto testovania celej postupnosti dát dochádza k frekvenčnému testu m-bitových blokov. Úspešný je vtedy, keď sa výsledný pomer rovná $\frac{m}{2}$, pričom m je počet bitov jedného bloku. Obdobne aj v tomto prípade sa odporúča testovať minimálne 100 bitovú postupnosť.

3. **Test rovnakých reťazcov – *Run Test*[25, kap. 2.3]**

   Zameraný na celkový počet po sebe idúcich neprerušených a rovnakých bitov – tzv. **behov**. Cieľom tohto testu je overiť rôznorodosť striedajúcich sa postupností pri generovaní. Odporúča sa aplikovať na postupnosť s minimálnou dĺžkou 100 bitov

4. **Test najdlhšej postupnosti jednotiek v bloku – *Test for the longest Run of Ones in a Block*[25, k. 2.4]**

   Zameraný na najdlhší beh v m-bitových blokoch. Má za úlohu overiť, či najväčšia vygenerovaná postupnosť je v súlade s najdlhšou sériou, aká sa očakáva, resp povoľuje v náhodnej postupnosti bitov[5]. Odporúča sa aplikovať test na minimálne 128 bitov náhodnej postupnosti. [6]

5. **Test hodností binárnych matic – *Binary Matrix Rank Test*[25, kap. 2.5]**

   Zameraný na poradie disjunktných[7] submatíc celej testovanej postupnosti. Overuje lineárne závislosti medzi podreťazcami s pevnými veľkosťami. Tento test je obdobne súčasťou sady Dieharder. Pre správne fungovanie je minimálna dĺžka postupnosti stanovená na 38 912 bitov.

6. **Test diskrétnej Fourierovej transformácie – *Discrete Fourier Transform* (*Spectral*) *Test*[25, kap. 2.6]**

   Zameraný na výšky vrcholov – **amplitúd**, jednotlivých bitov testovanej postupnosti v diskrétnej Fourierovej transformácii[8]. Cieľom je detekcia periodických znakov, ktorá by naznačovala odchýlku od predpokladu náhod-

[5] Povolený počet jednotiek závisí od veľkosti testovanej postupnosti.
[6] V tomto prípade sa dovoľuje maximálne 8, po sebe idúcich, jednotkových bitov.
[7] Disjunktné množiny sú také, ktoré nemajú žiaden spoločný prvok.
[8] `https://cs.wikipedia.org/wiki/Fourierova_transformace`.

nosti. Zisťujú sa prahové hodnoty[9] 95% znakov a zvyšných 5% testovanej postupnosti. Následne sa overuje, či nedochádza k významnej odlišnosti medzi týmito hodnotami. 1 000 bitov sa odporúča ako minimálna veľkosť postupnosti.

7. **Test neprekrývajúcich sa vzorov – *Non-overlapping Template Matching Test*[25, kap. 2.7]**

   Zameraný na počet vopred určených bitov v testovanom reťazci – tzv. okne. Cieľom je detegovať generátor, ktorý generuje veľa neperiodického vzoru. Test používa M-bitové okno na vyhľadanie konkrétneho m-bitového vzoru. Ak ho nenájde, nastane posun okna o jednu bitovú pozíciu. Ak sa nájde, okno sa resetuje na bity po nájdenom vzore a vyhľadávanie pokračuje. Nemá odporúčanú dĺžku vstupnej postupnosti. Parametre pre tento test sa vypočítajú na základe požiadavky.

8. **Test prekrývajúcich sa vzorov – *Overlapping Template Matching Test*[25, kap. 2.8]**

   Pracuje na rovnakom princípe ako 7. test. Rozdiel je len pri zhode vzoru s oknom. V tomto prípade sa okno posúva o jeden bit a následne pokračuje prehľadávanie.

9. **Mauerov „univerzálny štatistický“ test – *Maurer's „Universal Statistical“ Test*[25, kap. 2.9]**

   Zameraný na počet bitov medzi zhodnými vzormi. Overuje, či je možné danú postupnosť komprimovať bez straty informácií. Ak je možná veľká kompresia, tak daná postupnosť sa nepovažuje za náhodnú. Vstupná postupnosť môže mať variabilnú dĺžku. Odporúča sa aplikovať na minimálne 1 000 000-bitovú postupnosť.

10. **Test lineárnej zložitosti – *Linear Complexity Test*[25, k. 2.10]**

    Zameraný na dĺžku posuvného registra s lineárnou spätnou väzbou – LFSR[10]. Cieľom je zistiť, či je daná postupnosť dostatočne zložitá, aby sa mohla považovať za náhodnú. Platí, že dlhšie LFSRs spĺňajú tento predpoklad.

[9] Prahová hodnota – krajná, resp. hraničná hodnota. Získava sa prahovaním. Viac informácií o tejto metóde je dostupných v dokumente [47, k. 1.3].

[10] Výstup tohto registra je lineárne závislý od jeho počiatočného stavu a predchádzajúcich výstupov. Používa sa napríklad v PRNG a prúdových šifrách.

11. **Test Sérií – *Serial Test*[25, kap. 2.11]**

    Zameraný na frekvenciu všetkých možných prekrývajúcich sa m-bitových vzorov v celej testovanej postupnosti. Cieľom je zistiť, či počet výskytov je približne rovnaký, ako by mal byť v náhodnej postupnosti. Tá by mala byť rovnomerná[11]. Vyžaduje sa zvoliť m také, pre ktoré platí $m < [log_2 n] - 2$. Veľkosť vstupnej postupnosti v bitoch reprezentuje premenná n.

12. **Približný test Entropie – *Approximate Entropy Test*[25, kap. 2.12]**

    Podobný ako Test Sérií. Zameraný na frekvenciu všetkých možných prekrývajúcich sa vzorov s dĺžkou m-bitov. Porovnanie sa aplikuje na frekvencie dvoch po sebe nasledujúcich blokov, ktoré sa prekrývajú. Výsledok je následne porovnaný s ekvivalentom pre náhodnú postupnosť. Vyžaduje sa zvoliť m také, pre ktoré platí $m < [log_2 n] - 5$. Veľkosť vstupnej postupnosti v bitoch reprezentuje premenná n.

13. **Test kumulatívnych súčtov – *Cumulative Sums Test*[25, kap. 2.13]**

    Zameraný na maximálnu odchýlku od nuly pri kumulatívnom súčte všetkých bitov. Pri tomto postupnom hromadnom sčítaní predstavujú jednotkové bity kladné jednotky. Nuly na druhej strane záporné. Cieľom testu je určiť, či takýto kumulatívny súčet testovaných dát zodpovedá náhodnej sekvencii. Test je úspešný, ak sa výsledok sčítania blíži k nule. Odporúča sa, aby každá testovaná postupnosť mala minimálnu dĺžku 100 bitov.

14. **Test náhodných návštev – *Random Excursions Test*[25, kap. 2.14]**

    Zameraný na počet cyklov, ktoré majú presne K náhodných návštev v kumulatívnom súčte. Všetky cykly majú dĺžku zvolenú náhodne. Cieľom je zistiť či sa počet návštev odlišuje od hodnoty platnej pre náhodné dáta. Test pozostáva z 8 čiastočných testov. Každý otestuje jeden zo stavov: - 4, - 3, - 2, - 1, 1, 2, 3, 4. Úspešný je iba ak dáta uspejú vo všetkých čiastočných testoch. Odporúča sa testovať postupnosť s minimálnou dĺžkou 1 000 000 bitov.

15. **Test variantov náhodných návštev – *Random Excursions Variant Test*[25, kap. 2.15]**

    Zameraný na celkový počet návštev jednotlivých stavov pri kumulatívnom súčte. Pozostáva z 18 testov. Postupne sa testujú stavy: - 9, - 8, ..., - 1, 1, ..., 8, 9. Odporúčaná minimálna veľkosť vstupu je rovnaká ako v 14. teste.

[11] Rovnomernosť, resp. uniformita, náhodnej postupnosti znamená, že každý m-bitový vzor ma rovnakú pravdepodobnosť objaviť sa, ako ktorýkoľvek iný.

4.2.2 Interpretácia výsledkov sady štatistických testov

Nasledujúce vety vychádzajú z dokumentu [25, kap. 1.1.5]. Testovanie pomocou sady je založené na overení dvoch predpokladov, tzv. **nulovej** a **alternatívnej** hypotézy. Prvá z uvedených, ozn. H_0, tvrdí, že testovaná postupnosť je náhodná. Druhú definujeme ako inverznú voči nulovej, teda výstup z RNG je nenáhodný. Len jeden z týchto predpokladov je prijatý v priebehu aplikácie sady. V súvislosti s týmto postupom môže dôjsť k 2 typom chýb v priebehu testovania.

1. Postupnosť je náhodná, ale H_0 nebola akceptovaná.

2. Postupnosť nie je náhodná, avšak nulová hypotéza je prijatá.

Pravdepodobnosť, že dôjde k prvému zo spomenutých dejov definuje pojem – **hladina významnosti**, označíme α. Jej veľkosť závisí od konkrétneho štatistického testu. Typicky sa volí z intervalu $\langle 0.001, 0.01 \rangle$

Na vyhodnotenie uvedených predpokladov dochádza pri každom teste k porovnaniu **výslednej štatistickej hodnoty** S[12] a **kritickej** hodnoty, ďalej CV[13].
Každý z testov má definované štatistické hodnoty, na základe ktorých môžeme prijať alebo odmietnuť H_0. CV je hodnota získaná štatistickým rozdelením hodnôt. Platí, že táto hodnota predstavuje výsledok testu, ktorý pochádza na 99% z nenáhodnej postupnosti testovaných dát. Inými slovami, predstavuje hodnotu, pri ktorej už neakceptujeme H_0 ako pravdivé. Program vytvorí výpis s medzivýpočtami pre každý test a zároveň aj finálny výsledok testu. Používateľ si však môže všimnúť tzv. **pravdepodobnostnú hodnotu** – P-VALUE. Viď tabuľku 4.1.

Pri testovaní sú vstupné hodnoty testovaných postupnosti rozdelené na 10 približne rovnakých častí $(C1 - C10)$. Následne dochádza k výpočtu jednotlivých $p\text{-}values$ v každej z týchto častí. Tieto čiastkové $p\text{-}values$ sú vypočítané podľa daného štatistického testu a jeho výsledkov. Výsledná hodnota P-VALUE je vypočítaná pomocou uvedených čiastkových hodnôt $(p\text{-}values)$, dosadených do vzťahov a funkcií v [25, kap. 4.2.2]. Pri úspešnom testovaní platí pre výslednú hodnotu P-VALUE vzťah (4.1).
Nech

$$X = P\text{-VALUE},$$

potom,

$$(X \in (\alpha < X \geq 1)) \Leftrightarrow (H_0 = pravda). \tag{4.1}$$

[12] Hodnota získaná aplikovaním daného testu na naše dáta.
[13] Z ang. *Critical Value*.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STAT. TEST
486	487	459	423	439	450	479	487	443	447	0.306049	4546/4600	Frequency
427	474	476	456	482	470	464	443	460	448	0.776260	4553/4600	BlockFrequency
495	490	449	425	462	438	426	475	457	483	0.172899	4550/4600	CumulativeSums
458	450	468	461	456	474	495	457	425	456	0.718863	4553/4600	Runs
438	500	445	484	495	462	417	473	440	446	0.101732	4554/4600	LongestRun
464	440	481	484	459	428	442	455	477	470	0.642552	4555/4600	Rank
490	501	454	452	436	452	476	421	451	467	0.251503	4544/4600	FFT
462	425	448	488	466	514	494	443	426	434	0.032592	4560/4600	NonOverlappingTemplate
496	489	473	468	457	454	442	437	434	450	0.456758	4536/4600	OverlappingTemplate
483	453	461	456	445	445	450	438	469	500	0.616757	4553/4600	Universal
512	448	431	439	485	471	454	447	451	462	0.260244	4543/4600	ApproximateEntropy
254	289	272	308	272	249	326	310	291	293	0.022539	2833/2864	RandomExcursions
283	285	295	280	275	284	294	320	269	279	0.706626	2840/2864	RandomExcursionsVariant
452	472	468	441	475	449	477	439	467	460	0.922748	4563/4600	Serial
463	419	445	478	448	456	485	461	470	475	0.607722	4553/4600	LinearComplexity

Tabuľka 4.1: Príklad skráteného výpisu sady NIST STS po otestovaní náhodných dát

V tabuľke 4.1 sa nachádza taktiež stĺpec *PROPORTION*. Ten znázorňuje pomer úspešných a všetkých vykonaných testov pri testovaní sadou NIST.

4.2.3 Doba vykonávania testovacej sady

Obsahom tejto podkapitoly sú informácie súvisiace s dobou vykonávania sady NIST STS. Testovanie sme uskutočnili na konfigurácii A[14].

Na meranie dĺžky času vykonávania sme modifikovali zdrojové kódy sady NIST STS. Použili sme Windows rozhranie na meranie času – **QueryPerformanceCounter()**. Taktiež sme zisťovali počet cyklov jednotlivých testov vzhľadom na veľkosti vstupov m_1 a m_2. K tomu nám dopomohli inštrukcie **RDTSC** a **RDTSCP**. Opis použitých metód je obsahom kapitoly 3. Takto upravená testovacia sada je súčasťou prílohy A.

Experimentálne výsledky sú znázornené v tabuľke 4.2. Výsledná doba vykonania daného testu zavisí od veľkosti testovanej postupnosti. Pre celkový čas platí (4.2). Ak by sme vyhradili procesor iba pre nás, tak potom jednotlivé časy t_i by boli konštantné. Dôvodom je deterministický charakter procesu. Tento úkon však nie je v používateľskom režime možný. Uvedený vzťah ráta s týmto faktom.

Nech

m – veľkosť vstupnej testovacej postupnosti,

t_i – doba aplikovania všetkých testov na m-bitovú postupnosť,

[14]Vid. tabuľku 3.1 v kapitole 3.

n – počet testovaných prúdov s veľkosťou m,

T – výsledný čas celého testovania.

Potom

$$T = \sum_{i=1}^{n} t_i. \tag{4.2}$$

Pri meraní času v tabuľke 4.2 sme zvolili $n = 1$ pre obidve náhodné postupností (m_1, m_2).

Test č.	**Veľkosť vstupnej testovanej postupnosti** $m_1 =$ **122kB** $\approx$ **1 000 000 b**		$m_2 =$ **1,165GB** $\approx$ **10 000 000 000 b**	
	ČAS [s]	**CYKLY**	**ČAS [s]**	**CYKLY**
1	0,004 055	11 730 471	5,387 021	15 593 089 692
2	0,002 740	7 921 031	3,591 694	10 396 392 294
3	0,009 644	27 907 135	13,367 284	38 692 505 860
4	0,014 938	43 230 329	21,562 572	62 414 330 729
5	0,007 866	22 760 418	10,895 430	31 537 547 299
6	0,074 364	215 241 654	104,454 277	302 350 042 366
7	0,226 567	655 801 389	0,036 371	105 257 157
8	1,763 558	5 104 725 551	2 485,426 270	719 423 6071 645
9	0,056 390	163 214 697	78,888 596	228 348 426 960
10	0,045 201	130 830 194	67,123 001	194 292 108 778
11	0,203 056	587 748 800	285,328 339	825 902 376 471
12	0,004 447	12 862 109	10,411 473	30 136 705 606
13	0,005 211	15 068 574	76,817 490	222 353 456 121
14	0,564 533	1 634 064 740	781,071 777	2 260 865 638 374
15	3,703 921	10 721 233 139	5 222,643 066	15 117 297 898 720
Sumár	6,686 501	19 354 524 033	9 167,004 883	26 534 522 105 621

Tabuľka 4.2: Meranie testovania sady pomocou konfigurácie A

Informácie ku spusteniu sady NIST STS

Na spustenie sady je potrebná inicializácia. Používateľ ju realizuje vstupmi z príkazového riadku. Zdrojový kód 4.1, znázorňuje tento úkon. Veľkosť otestovaných dát pomocou takejto konfigurácie je 11,92 MB[15].

Na základe faktov súvisiacich s výpočtom P-VALUE je potrebné pre správne vyhodnotenie otestovať viacero postupností. Pri určení veľkostí vstupnej postup-

[15] 1 000 000 b $*$ 100 $=$ 100 000 000 b $\approx$ 11,92 MB.

nosti preto odporúčame zvoliť $m = 1000000$ bitov ($\approx$ 122kB) a počet prúdov stanoviť na číslo deliteľné 10. Dôvodom je rozdelenie testovaných postupností práve na 10 rovnakých častí. Parametre jednotlivých testov nemeníme. Týmto postupom zamedzíme chýbam pri inicializácii sady a vykonané testy poskytnú presnejšie výsledky.

Zdrojový kód 4.1: Príklad inicializácie testovacej sady NIST STS

```
1 > ./assess.exe 1000000   // size of random sequence in bits
2 > 0                      // apply tests on input file
3 > myRandomDataFile.bin   // name of file with random numbers
4 > 1                      // apply all tests
5 > 0                      // no changes of default tests values
6 > 100                    // number of bit streams
7 > 1                      // choosen bin format of input file
```

5 Generovanie náhodných dát

Na generovanie náhodných dát má používateľ k dispozícii hneď niekoľko možností. V rámci tejto práce rozhrania rozdelíme do dvoch kategórií:

- Hardvérové API – závisí od technického vybavenia počítača.
- Softvérové API – implementáciu služby RNG zabezpečuje OS alebo knižnice zvoleného jazyka.

V tejto kapitole uvedené rozdelenie viac charakterizujeme pomocou nasledujúcich podkapitol. Súčasťou každej opísanej funkcie sú aj tabuľky s výsledkami experimentálnych meraní jednotlivých funkcií. Merania boli uskutočnené pomocou metód opísaných v kapitole 3. Obdobne sme použili aj opísané označenia. Konkrétne:

- počet opakovaní volania testovanej funkcie – NI,
- veľkosť buffer-a – BS,
- celková veľkosť vygenerovaných dát – V_D,
- priemerný počet cyklov pri vykonávaní testovanej funkcie – ANC,
- čas vykonávania (bez ukladania dát) – T_A,
- čas vykonávania (s ukladaním dát) – T_B,
- priepustnosť dát procesu T_A – P_A,
- priepustnosť dát procesu T_B – P_B.

Posledný riadok každej konfigurácie v tabuľkách, bol meraný pri maximálnej možnej veľkosti buffer-a, ktorú funkcia dokáže poskytnúť.

5.1 Hardvérové rozhrania

Súčasťou moderných procesorov sú aj implementácie generátorov skutočne náhodných čísel. Ich použitie je realizované pomocou procesorových inštrukcií `RDRAND` a `RDSEED`. Práca s inštrukciami vyžaduje znalosti nízko-úrovňových programovacích jazykov, akým je napríklad *Asembler*[1]. Tie nie sú v dnešnej dobe veľmi populárne. Aj dôsledkom toho výrobcovia procesorov sprístupňujú programátorom tzv. **programovateľné rozhrania – API**. Štandardne sú napísané pomocou vysoko-úrovňových jazykov[2]. Ich transformáciu do strojového kódu zabezpečuje prekladač.

V tejto súvislosti spomenieme dvoch výrobcov procesorových čipov s najväčším zastúpením na trhu – **AMD a Intel**. Obe firmy sprístupnili používateľom svoje API v programovacom jazyku C. Najpodstatnejším rozdielom pri používaní je vzájomná **kompatibilita**. Intel rozhranie poskytuje podporu iba pre vlastné procesory, zatiaľ čo spoločnosť AMD uverejnila sadu, ktorá má podporu aj iných výrobcov procesorových čipov. Podmienkou úspešného generovania však ostáva potrebná implementácia inštrukcií `RDRAND` a `RDSEED`. Z tohto dôvodu sme použili pri generovaní náhodných dát, pomocou vyššie spomenutých inštrukcií, práve riešenie firmy AMD.

V súvislosti s inštrukčnými sadami, implementovanými na dnešných mikroprocesoroch, uvádzame dokument [48]. Jeho obsahom sú špecifikácie inštrukcií, používaných na rôznych typoch procesorov. Dokument je pravidelne aktualizovaný a udržiavaný.

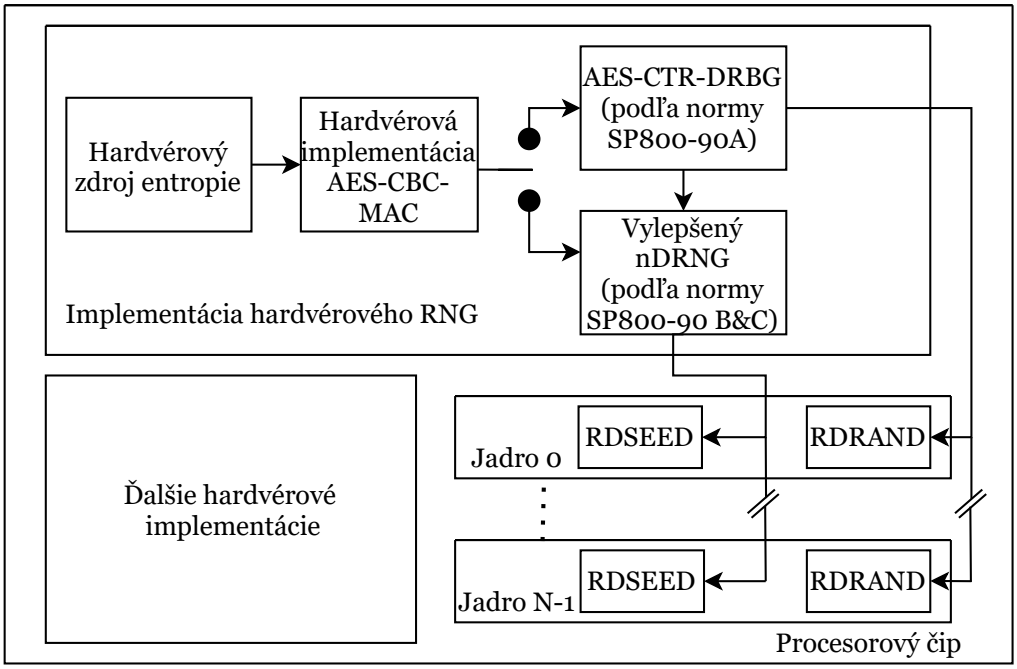
5.1.1 RDRAND a RDSEED

Inštrukcie vytvorila spoločnosť Intel. Ich vznik sa datuje do roku **2012**. Samotní autori ([49]), odôvodnili implementáciu TRNG v procesorových čipoch ako nutný bezpečnostný prvok, ktorý dokáže v akejkoľvek situácii dodať kvalitné a skutočne náhodné dáta. Samozrejme, nezávislé od deterministických procesov bežiaceho softvéru.

Inštrukcia RDSEED je implementáciou n-DRNG. Na druhej strane RDRAND realizuje CSPRNG, ktorého zdrojom entropie je výstup RDSEED inštrukcie. Obidva uvedené typy RNG boli opísané v kapitole 1 tejto práce. Proces generovania

[1] `https://en.wikipedia.org/wiki/Assembly_language`.
[2] Jazyk C, Java, C Sharp a iné.

náhodných dát pomocou týchto príkazov je znázornený na obrázku 5.1[3,4]. In-



Obr. 5.1: Implementácia procesorových inštrukcií RDRAND a RDSEED v procesoroch Intel

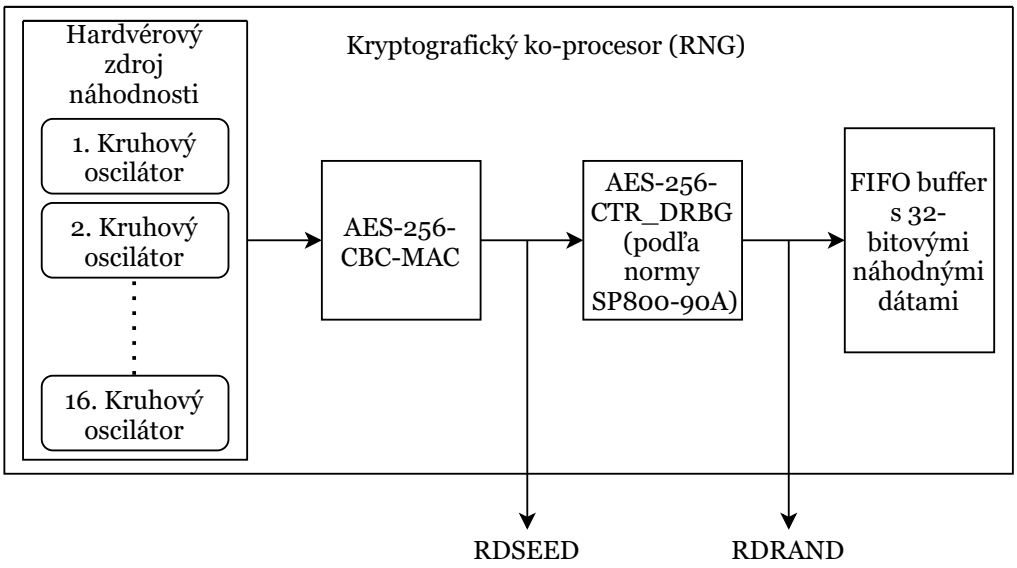
tel procesory používajú ako hardvérový zdroj entropie tepelný šum procesora pri rýchlosti 3 GHz. Nepotrebujú žiaden externý zdroj napájania, pretože používajú rovnaký zdroj ako jadro, na ktorom generovanie prebieha. Podrobný opis aplikovaných metód v generátore je dostupný v dokumentácii [49].

Podporu týchto inštrukcií doplnila aj spoločnosť AMD v roku 2015. Do svojich procesorov implementovali kryptografické ko-procesory. Architektúra je znázornená pomocou schémy 5.2. Zdrojom entropie sú tzv. **kruhové oscilátory**[5], ktoré počas behu ko-procesora neustále dodávajú náhodné dáta. Uvedené AMD ko-procesory používajú aj tzv. FIFO buffer [50] na zabezpečenie podpory rýchleho čítanie 32-bitových hodnôt[6]. Viac podrobností o riešení sa nachádza v [51].

AMD Secure RNG API [51, str. 1-7]

Voľné dostupné rozhranie[7] v programovacom jazyku C. Aktuálna verzia je 3.0.6 (15. 03. 2021), obsahuje celkovo 14 funkcií. Z toho dve na kontrolu implementácie inštrukcií a 12 na vytváranie náhodných dát. Programátor má možnosť

[3]MAC – z ang. *Message Authentication Code*.
[4]Prebraté z: [49, kap. 3.1].
[5]Z ang. *Ring Oscillators*.
[6]Z ang. *Bursts*.
[7]`https://developer.amd.com/amd-aocl/rng-library/`.



Obr. 5.2: Kryptografické ko-procesory v AMD procesoroch

vygenerovať jedno 16/32/64-bitové číslo alebo ich dátovú štruktúru. Obdobne je k dispozícii možnosť zvoliť si požadovanú veľkosť vygenerovaných dát v bajtoch. Pri našom testovaní sme implementovali možnosť generovania náhodných dát pomocou polí v jazyku C. Zdrojový kód, 5.1, uvádza deklarácie spomenutých implementovaných funkcií. Pomenovania sú jednoznačné. Argument N označuje počet opakovaní a *retry_count* udáva počet pokusov v prípade zlyhania.

Obsahom AMD balíka je aj zdrojový kód **secrng_test.c**. Ten realizuje príklad použitia každého AMD rozhrania.

Zdrojový kód 5.1: Funkcie v AMD Secure RNG rozhraní

```
1  int is_RDRAND_supported();
2  int get_rdrand16u(uint16_t* rng_val, unsigned int retry_count);
3  int get_rdrand32u(uint32_t* rng_val, unsigned int retry_count);
4  int get_rdrand64u(uint64_t* rng_val, unsigned int retry_count);
5  int get_rdrand32u_arr(uint32_t* rng_val,
6                        unsigned int N,
7                        unsigned int retry_count);
8  int get_rdrand64u_arr(uint64_t* rng_arr,
9                        unsigned int N,
10                       unsigned int retry_count);
11 int get_rdrand_bytes_arr(unsigned char *rng_arr,
12                          unsigned int N,
13                          unsigned int retry_count);
14 int is_RDRAND_supported();
15 int get_rdseed16u(uint16_t* rng_val, unsigned int retry_count);
16 int get_rdseed32u(uint32_t* rng_val, unsigned int retry_count);
17 int get_rdseed64u(uint64_t* rng_val, unsigned int retry_count);
18 int get_rdseed32u_arr(uint32_t* rng_arr,
19                       unsigned int N,
20                       unsigned int retry_count);
21 int get_rdseed64u_arr(uint64_t* rng_arr,
22                       unsigned int N,
23                       unsigned int retry_count);
24 int get_rdseed_bytes_arr(unsigned char *rng_arr,
25                          unsigned int N,
26                          unsigned int retry_count);
```

Výsledky experimentálnych meraní funkcií rozhrania AMD

Dosiahnuté namerané výsledky sú reprezentované formou tabuliek:

- 5.1 – RDRAND a
- 5.2 – RDSEED.

Jednotlivé označenia stĺpcov sú charakterizované v úvode tejto kapitoly (5). Zdrojový kód `amdSPRNG.c` implementuje testované funkcie a je obsahom prílohy A.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	5 628 973	1,991 472	1,953 684	8,034	8,190
	1 024	16 MB	16 GB	5 676 879 262	2 008,291 504	2 015,468 384	8,158	8,129
	1 024	32 MB	32 GB	11 354 590 480	4 016,876 709	4 050,146 973	8,158	8,091
	8	ULONG_MAX	32 GB	1 453 162 191 677	4 016,253 662	4 051,736 572	8,159	8,087
B	1 024	16 kB	16 MB	6 241 786	3,551 908	3,639 144	4,505	4,397
	1 024	16 MB	16 GB	6 440 954 785	3 664,190 186	3 685,320 801	4,471	4,445
	1 024	32 MB	32 GB	12 897 904 573	7 337,478 516	7 486,955 566	4,466	4,377
	8	ULONG_MAX	32 GB	1 650 387 205 662	7 335,057 129	7 598,942 383	4,467	4,312
C	1 024	16 kB	16 MB	4 805 709	2,330 819	2,351 972	6,865	8,803
	1 024	16 MB	16 GB	4 810 432 963	2 332,328 369	2 338,702 881	7,025	7,006
	1 024	32 MB	32 GB	9 617 320 340	4 662,936 523	4 707,775 879	7,029	6,960
	8	ULONG_MAX	32 GB	1 232 428 756 993	4 668,283 203	4 790,067 383	7,019	6,841

Tabuľka 5.1: Výsledky meraní funkcie get_rand_bytes_arr

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	24 876 680	8,800 674	8,673 107	1,818	1,845
	1 024	16 MB	16 GB	25 529 272 494	9 031,408 203	8 902,758 789	1,814	1,840
	1 024	32 MB	32 GB	50 962 566 174	17 528,861 328	17 621,734 375	1,869	1,860
	8	ULONG_MAX	32 GB	6 374 627 781 381	17 618,212 891	18 180,208 984	1,860	1,802
B	1 024	16 kB	16 MB	6 390 445	3,636 515	4,014 131	4,400	3,990
	1 024	16 MB	16 GB	6 447 545 248	3 667,939 209	3 675,636 963	4,467	4,457
	1024	32 MB	32 GB	13 112 239 637	7 459,411 621	7 479,892 578	4,393	4,381
	8	ULONG_MAX	32 GB	1 688 133 062 298	7 502,824 219	7 615,551 758	4,367	4,303
C	1 024	16 kB	16 MB	4 811 400	2,333 592	2,367 678	6,856	6,758
	1 024	16 MB	16 GB	4 819 373 956	2 336,663 330	2 338,929 688	7,012	7,005
	1 024	32 MB	32 GB	9 615 872 296	4 662,234 375	4 706,395 508	7,028	6,962
	8	ULONG_MAX	32 GB	1 231 308 014 422	4 664,037 598	4 769,422 363	7,026	6,870

Tabuľka 5.2: Výsledky meraní funkcie get_rdseed_bytes_arr

5.2 Rozhrania operačného systému Windows

Používateľ má v aktuálnom OS Windows[8] prístup k trojici funkcií realizujúcich službu RNG[32, str. 5].

- `RtlGenRandom` [52],
- `CryptGenRandom` [53],
- `BCryptGenRandom` [54].

Vyššie uvedené realizujú generovanie náhodných čísel pomocou používateľského rozhrania – `ProcessPrng`. V prípade kernel režimu je vytváranie náhodných dát vykonané primárne pomocou `SystemPrng` API. Tie boli spomenuté v 2.2. Obsahom tejto podkapitoly je opis týchto funkcií. Ten bol vytvorený na základe dokumentácií spoločnosti Microsoft [52], [53], [54].

[8]Windows 10 Home – 64-bit, verzia 20H2, zostava OS – 19042.964.

5.2.1 RtlGenRandom

Funkcia je deklarovaná v hlavičkovom súbore **ntsecapi.h**, ale nemá knižnicu, ktorá ju vykonáva. Slúži na generovanie pseudonáhodných čísel. Pomocou makra je definovaná ako `SystemFunction036` a až táto je realizovaná v dynamickej knižnici **Advapi32.dll**. Pri použití je teda potrebné načítať tento modul. V súčasnosti je tento krok automatizovaný. Generovanie je realizované použitím rozhrania **ProcessPrng**. Microsoft však odporúča namiesto používania tejto funkcie použitie `CryptGenRandom`.

Špecifikácia funkcie RtlGenRandom [52]

```
RtlGenRandom(PVOID RandomBuffer, ULONG RandomBufferLength);
```

1. `PVOID RandomBuffer` – adresa premennej na uloženie náhodnosti.
2. `ULONG RandomBufferLength` – veľkosť prvého parametra (max. `ULONG_MAX`).

Funkcia je typu `boolean`. Teda návratové hodnoty sú `TRUE/FALSE` pri úspechu, resp. neúspechu generovania. Príkladom použitia je zdrojový kód 5.2. Tento demo príklad je súčasťou prílohy A. Nachádza sa v priečinku *DemoExamples*.

Zdrojový kód 5.2: Príklad použitia funkcie RtllGenRandom

```
1  #include<stdio.h>
2  #include<windows.h>     // for variable types definition
3  #include<ntsecapi.h>    // declaration RtlGenRandom()
4
5  int main(){
6    BYTE *pbData=(BYTE*)malloc(sizeof(BYTE) * 10);
7    if(RtlGenRandom(pbData,10) == TRUE){
8      for (int i = 0; i < 10; i++){
9        printf("%u",pbData[i]);
10     }
11     free(pbData);
12     return 0;
13   }
14   free(pbData);
15   return -1;
16 }
```

Výsledky experimentálnych meraní

Dosiahnuté výsledky meraní sú znázornené pomocou tabuľky 5.3.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	15 074	0,005 451	0,020 702	2 935,241	772,872
	1 024	16 MB	16 GB	16 907 537	5,982 000	166,227 746	2 738,883	98,564
	1 024	32 MB	32 GB	33 698 517	11,922 174	524,159 006	2 758,492	62,515
	8	ULONG_MAX	32 GB	4 489 111 941	12,407 025	500,792 203	2 641,084	65,432
B	1 024	16 kB	16 MB	10 580	0.006 771	0.021 564	2 363,019	741,977
	1 024	16 MB	16 GB	14 072 032	8,006 607	136,472 897	2 046,310	120,053
	1 024	32 MB	32 GB	29 400 765	16,726 991	984,013 108	1 958,990	33,300
	8	ULONG_MAX	32 GB	3 940 128 574	17,511 696	275,192 794	1 871,207	119,073
C	1 024	16 kB	16 MB	14 630	0,007 994	0,024 424	2 001,501	655,093
	1 024	16 MB	16 GB	11 823 989	5,733 827	217,114 585	2 857,428	75,462
	1 024	32 MB	32 GB	23 913 126	11,595 265	513,567 815	2 825,981	63,805
	8	ULONG_MAX	32 GB	3 347 779 853	12,680 974	561,609 795	2 584,029	58,347

Tabuľka 5.3: Výsledky meraní funkcie RtlGenRandom

5.2.2 CryptGenRandom

Funkcia na generovanie kryptograficky bezpečných náhodných dát. Vznikla pri prvom riešení kryptografického rozhrania – CAPI. **Zastaraná**, ale zatiaľ podporovaná aj v súčasnom CNG. Microsoft však **neodporúča** jej používanie z dôvodu možného odstránenia v budúcnosti. Jej deklarácia je obsahom hlavičkového súboru **wincrypt.h**. Realizuje ju dynamická knižnica **Advapi32.dll**. Následne je použitý modul **brcyptprimitives.dll**.

Špecifikácia funkcie CryptGenRandom [53]

```
CryptGenRandom(HCRYPTPROV hProv, DWORD dwLen, BYTE *pbBuffer);
```

1. `HCRYPTPROV hProv`[9] – opis CSP[10], vytvorený funkciou `CryptAcquireContext`,
2. `DWORD dwLen` – veľkosť výstupu (max. `ULONG_MAX`),
3. `BYTE *pbBuffer` – adresa úložiska, veľkosť musí byť najmenej `dwLen`.

Funkcia `CryptGenRandom` je typu `BOOL`. Návratová hodnota je `TRUE`, resp. `FALSE`. V prípade zlyhania je dôvod zapísaný do CSP premennej `hProv`.

[9] Nutná inicializácia tejto premennej.
[10] Opísané v 2.1.1.

Zdrojový kód 5.3: Príklad použitia funkcie CryptGenRandom

```
#include<stdio.h>
#include<windows.h>

int main(){
  HCRYPTPROV   hCryptProv;
  BYTE   *pbData=(BYTE*)malloc(sizeof(BYTE)* 10);
  CryptAcquireContext(&hCryptProv,NULL,
    "Microsoft Base Cryptographic Provider v1.0",
                      PROV_RSA_FULL,
                      CRYPT_VERIFYCONTEXT);
  if(CryptGenRandom(hCryptProv,10,pbData)!=0){
    printf("Random sequence generated. \n");
  }
  else
  {
    printf("Error during CryptGenRandom.\n");
    free(pbData);
    return -1;
  }
  free(pbData);
  return 0;
}
```

Výsledky experimentálnych meraní funkcie CryptGenRandom

Výsledky experimentov funkcie `CryptGenRandom` znázorňuje tabuľka 5.4. Zdrojové kódy `winAPIprng.c` a ukážka 5.3 implementujú túto funkciu a sú obsahom prílohy A.

5.2.3 BCryptGenRandom

Implementácia služby generovania náhodných čísel v druhej verzii kryptografického rozhrania. Ako jediná z uvedených je navrhnutá na použitie v používateľskom aj kernel režime. **Deklarácia** je uvedená v hlavičkovom súbore **bcrypt.h**. Funkcia je následne vykonaná pomocou **bcrypt.dll**. Knižnicu je potrebné pri kompilácií programu prilinkovať. Tento úkon realizujeme pomocou prepínača `-lbcrypt`.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	14 242	0,006 430	0,022 011	2 488,336	726,909
	1 024	16 MB	16 GB	16 991 496	6,013 020	143,722 924	2 724,754	113,997
	1 024	32 MB	32 GB	33795736	11,957515	480,253462	2 740,369	68,231
	8	ULONG_MAX	32 GB	4 525 421 725	12,508601	452,834846	2 619,637	72,362
B	1 024	16 kB	16 MB	10 545	0,008 252	0,021 986	1 938,924	727,736
	1 024	16 MB	16 GB	14 174 624	8,066 505	120,477 247	2 031,115	135,992
	1 024	32 MB	32 GB	29 485 483	16,776 389	982,773 749	1 953,221	33,342
	8	ULONG_MAX	32 GB	3 969 183 958	17,642 095	283,746 065	1 857,376	115,484
C	1 024	16 kB	16 MB	10 884	0.007 010	0.024 739	2 282,454	646,752
	1 024	16 MB	16 GB	11 819 580	5,733 101	213,303 019	2 857,790	76,811
	1 024	32 MB	32 GB	23 865 849	11,573 146	509,545 807	2 831,382	64,308
	8	ULONG_MAX	32 GB	5 285 174 011	20,022 825	561,612 424	1 636,532	58,346

Tabuľka 5.4: Výsledky meraní funkcie CryptGenRandom

Špecifikácia funkcie BCryptGenRandom[54]

`BCryptGenRandom(BCRYPT_ALG_HANDLE hAlgorithm, PUCHAR pbBuffer, ULONG cbBuffer, ULONG dwFlags);`

1. `BCRYPT_ALG_HANDLE hAlgorithm` – opis algoritmu CSP. Vytvára sa použitím funkcie `BCryptOpenAlgorithmProvider`. Tú však nie je nutné inicializovať. Pri použití makra `NULL` sa použije predvolený poskytovateľ[11], ktorý poskytuje služby generovania náhodných čísel.
2. `PUCHAR pbBuffer` – adresa úložiska dát. Veľkosť musí byť najmenej cbBuffer.
3. `ULONG cbBuffer` – veľkosť vygenerovaných dát (max. `ULONG_MAX`).
4. `ULONG dwFlags` – prepínač na modifikovanie správania funkcie.

Parameter `dvFlags` môže nadobúdať hodnoty:

- nula – je nutné inicializovať CSP,
- `BCRYPT_RNG_USE_ENTROPY_IN_BUFFER`[12] – obsah dát v pbBuffer sa použije ako dodatočný zdroj entropie.
- `BCRYPT_USE_SYSTEM_PREFERRED_RNG`[13] – v prípade, že CSP je rovný NULL, volíme tento parameter.

[11] Microsoft Cryptographic Service Provider.
[12] Od verzie Windows 8 a vyššie je ignorovaný.
[13] Windows Vista nepodporuje tento prepínač.

Funkcia `BCryptGenRandom` je typu `NTSTATUS`. Návratové hodnoty môžu byť `STATUS_SUCCESS`, `STATUS_INVALID_HANDLE` a `STATUS_INVALID_PARAMETER`.

Zdrojový kód 5.4: Príklad použitia funkcie BCryptGenRandom

```
1  #include<stdio.h>
2  #include<windows.h>
3  #include<ntstatus.h>
4  int main(){
5    BYTE   *pbData=(BYTE*)malloc(sizeof(BYTE)* 10);
6    if (STATUS_SUCCESS!=BCryptGenRandom(NULL,pbData,10,
7                          BCRYPT_USE_SYSTEM_PREFERRED_RNG))
8            printf("BCryptGenRandom error.\n");
9    else    printf("Random sequence generated.\n");
10   free(pbData);
11   return 0;
12 }
```

Výsledky experimentálnych meraní funkcie BCryptGenRandom

Výsledky meraní funkcie sú znázornené pomocou tabuľky 5.5. Programy použité pri experimentoch sú obsahom prílohy A, spoločne so zdrojovým kódom 5.4 a implementáciou Windows rozhraní (`winAPIprng.c`).

Počítač	**Špecifikácie meraní**							
	NI	BS	V_D	ANC	T_A **[s]**	T_B**[s]**	$\approx P_A$ **[MB/s]**	$\approx P_B$ **[MB/s]**
A	1 024	16 kB	16 MB	15 376	0,005 560	0,020 745	2 877,698	771,270
	1 024	16 MB	16 GB	16 905 128	5,981 131	236,919 392	2 739,281	69,154
	1 024	32 MB	32 GB	33 805 640	11,960 016	506,876 314	2 739,796	64,647
	8	ULONG_MAX	32 GB	4 538 998 240	12,544 902	568,900 696	2 612,057	57,599
B	1 024	16 kB	16 MB	11 011	0,007 250	0,066 015	2 206,897	242,369
	1 024	16 MB	16 GB	14 093 652	8,018 871	117,026 229	2 043,180	140,003
	1 024	32 MB	32 GB	29 396 766	16,724 702	845,170 274	1 959,257	38,771
	8	ULONG_MAX	32 GB	3 951 000 676	17,560 018	252,854 179	1 866,057	129,592
C	1024	16 kB	16 MB	10 912	0,005 963	0,025 895	2 683,213	617,880
	1 024	16 MB	16 GB	11 825 926	5,734 744	102,225 511	2 856,971	160,273
	1 024	32 MB	32 GB	23 915 195	11,596 222	437,033 046	2 825,748	74,978
	8	ULONG_MAX	32 GB	3 346 909 253	12,677 676	591,363 044	2 584,701	55,411

Tabuľka 5.5: Výsledky meraní funkcie BCryptGenRandom

5.3 Rozhrania na generovanie náhodných čísel v jazyku C a knižnici OpenSSL

Používateľ pracujúci na platforme Windows môže okrem softvérových riešení OS použiť aj implementácie rôznych knižníc. Uvedenou metódou môžeme vytvoriť aplikáciu, resp. službu, nezávislú od operačného systému. Tzv. **mutiplatformové programy**. V tejto kapitole demonštrujeme vyššie opísaný postup. Použijeme štandardné knižničné rozhrania jazyka C[14] a celosvetovo známu kryptografickú knižnicu – OpenSSL[15] [55].

Zdrojové kódy v tejto podkapitoly sú obsahom prílohy A.

5.3.1 rand() a srand()

Funkcia `rand()` je zrejme najznámejšou funkciou na generovanie pseudonáhodných výstupov, s ktorou sa používateľ stretne. Vo väčšine prípadov je implementáciou lineárne kongruentného generátora. Jeho inicializačná hodnota sa mení pomocou funkcie `srand()`. Pri tejto metóde je teda nutnosťou kooperácia týchto funkcií. Ak by sme neurčili seed generátora, výstup by bol vždy rovnaký[16]. Výstupom funkcie je hodnota z číselnej množiny prvkov 0 až 32767. Použitie rozhrania na kryptografické účely sa **neodporúča**. Z toho dôvodu nevykonáme testovanie tohto rozhrania. Príkladom použitia je zdrojový kód 5.5.

Zdrojový kód 5.5: Príklad použitia funkcie rand a srand

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main () {
5    time_t t;
6    /* Intializes random number generator */
7    srand((unsigned) time(&t));
8
9    /* Print random numbers from 0 to 32767 */
10   printf("%d\n", rand());
11   return(0);
12 }
```

[14] Funkcie ***rand, srand*** a ***rand_s***.
[15] Funkcie ***RAND_bytes*** a ***RAND_priv_bytes***.
[16] Rovný ***srand(1)***.

Viac informácia čitateľ nájde v [56].

5.3.2 rand_s

Funkcia `rand_s`[17] je vylepšením rozhrania `rand`. Výstupom sú **pseudonáhodné** dáta v rozmedzí od 0 až `UINT_MAX`[18]. Minimálna a zároveň aj maximálna veľkosť generovaných dát pri jednom volaní funkcie je 32 bitov (`u_int`). Funkcia používa operačný systém, aby vygenerovala kryptograficky bezpečné náhodné dáta. Nepotrebuje `srand()` funkciu.

Špecifikácia funkcie rand_s

```
rand_s(unsigned int* randomValue);
```

Funkcia používa jeden vstupný parameter. Ním je adresa na uloženie náhodnosti. Premenná musí byť typu `unsigned int`. `Rand_s` je typu `errno_t`. Návratové hodnoty sú **nula** pri úspechu alebo chybový kód pri zlyhaní. Príkladom použitia je zdrojový kód 5.6, ktorý je obsahom priečinku *DemoExamples* v Prílohe A. Obdobne je funkcia implementovaná v `winAPIprng.c`.

Zdrojový kód 5.6: Príklad použitia funkcie rand_s

```
1  #include <stdio.h>
2  #define _CRT_RAND_S
3  #include <stdlib.h>
4
5  int main () {
6    unsigned int data;
7    //generating 10*32bit value
8    for (int i = 0; i < 10 ; i++)
9      if(rand_s(&data)==0) printf("%u\n",data);
10     else printf("Rand_s error\n");
11   return 0;
12 }
```

Pred použitím je nutné definovať makro `_CRT_RAND_S` pred linkovaním knižnice **stdlib.h**. Tým je zabezpečená deklarácia `rand_s`.

Pri skúmaní dokumentácie sme zistili, že **používa rozhranie** `RtlGenRandom`, konkrétne makro `SystemFunction036`. Uvedená funkcia je obsahom podkapitoly 5.2.1.

[17] Kompatibilné iba s OS Windows.

[18] ***UINT_MAX*** = ***ULONG_MAX*** = 4 294 967 295.

Výsledky experimentálnych meraní funkcie rand_s

Výsledky dosiahnuté pri meraní tejto funkcie sú znázornené pomocou tabuľky 5.6. Programy spoločne s výsledkami, na základe ktorých táto tabuľka vznikla, sú obsahom prílohy A.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	4 194 304	32 b	16 MB	767 988 443 354 161	0,505 931	0,555 474	31,625	28,804
	4 294 967 296	32 b	16 GB	768 430 967 898 097	513,802 964	562,708 093	31,888	29,116
	8 589 934 592	32 b	32 GB	768 535 482 013 732	1 025,192 557	1 166,065 347	31,963	28,101
B	4 194 304	32 b	16 MB	15 547 492 712 804 349	4,381 201	4,313 881	3,652	3,709
	4 294 967 296	32 b	16 GB	5 446 628 539 601 029	4 481,040 633	4 521,708 060	3,656	3,623
	8 589 934 592	32 b	32 GB	5 447 757 045 127 528	8 955,748 443	9 164,667 793	3,659	3,575
C	4 194 304	32 b	16 MB	1 729 007 496 224 506	5,719 874	3,098 462	2,797	5,164
	4 294 967 296	32 b	16 GB	690 952 124 931 332	4 147,831 673	3 149,553 623	3,950	5,202
	8 589 934 592	32 b	32 GB	689 962 359 510 427	6 229,372 939	11 885,556 796	5,260	2,757

Tabuľka 5.6: Výsledky meraní funkcie rand_s

5.3.3 Kryptografická knižnica OpenSSL

OpenSSL [55] je široko použiteľná a modifikovateľná knižnica, určená pre kryptografické aplikácie. Jej obsahom sú kvalitné a udržiavané algoritmy. Obdobne obsahuje aj funkcie, ktorých úlohou je generovanie kryptograficky bezpečných náhodných dát. Podľa dokumentácie [57] je ich implementácia skonštruovaná podľa odporúčania [2], teda knižnica **používa** na generovanie čísel **AES256-DRBG v čítačovom režime**. Ekvivalentná bezpečnosť výstupov je teda na úrovni 256 bitov.

Špecifikácia RNG funkcií v knižnici OpenSSL[58, kap. 5]

V tejto práci budeme pracovať s OpenSSL verzia 1.1.1k (25. 03. 2021). Predvolený generátor sa inicializuje pri spustení a automaticky vykonáva reseed-ovanie. Pri tomto procese používa dôveryhodné zdroje daného operačného systému. V prípade OS Windows nimi sú výstupy funkcie `BCryptGenRandom` a `CryptGenRandom`. Tie sú opísané v podkapitole 5.2

Prístup k CSPRNG sprostredkúva dvojica funkcií:

- `int RAND_bytes(unsigned char * buf, int num)`,
- `int RAND_priv_bytes(unsigned char * buf, int num)`.

Obidve sú deklarované v **rand.h** a poskytujú výstup z rovnakého CSPRNG. Druhá z uvedených funkcií používa unikátnu inštanciu generátora. Odporúča sa ju používať pri tvorbe citlivých dát. Napríklad pri procese generovania kľúčov. Ukážku

jednoduchého použitia v jazyku C znázorňuje zdrojový kód 5.7. Viac informácií o metódach RNG rozhraní je možne nájsť v riporte [58, kap. 5]

Zdrojový kód 5.7: Príklad použitia RNG funkcií knižnice OpenSSL

```
1  #include <stdio.h>
2  #include <openssl/rand.h>
3
4  int main(){
5    unsigned char * data =(unsigned char*)malloc(
6                            sizeof(unsigned char)*10);
7
8    if(RAND_bytes(data,10))
9      for (int i = 0; i < 10; i++)
10       printf("%u", data[i]);
11   else printf("RAND_bytes error\n");
12   printf("\n");
13   if (RAND_priv_bytes(data,10))
14     for (int i = 0; i < 10; i++)
15       printf("%u", data[i]);
16   else printf("RAND_priv_bytes error\n");
17
18   free(data);
19   return 0;
20 }
```

Výsledky experimentálnych meraní

Aj napriek faktu, že knižnica čerpá zdroj náhodnosti z rozhraní operačného systému Windows, sme sa rozhodli aplikovať naše testovacie metódy na funkcie v OpenSSL. Dôvodom je vysoká miera používania v bežnej prevádzke. Prehľad nameraných hodnôt pri časovom meraní znázorňujú tabuľky č.:

- 5.7 – funkcia `RAND_bytes`,
- 5.8 – funkcia `RAND_priv_bytes`.

Funkcie sú implementované v súbore `openssl_rng.c`. Spoločne s ukážkou 5.7, sú obsahom prílohy A.

Počítač	**Špecifikácie meraní**							
	NI	BS	V_D	ANC	T_A **[s]**	T_B**[s]**	$\approx P_A$ **[MB/s]**	$\approx P_B$ **[MB/s]**
A	1 024	16 kB	16 MB	12 700	0,008 139	0,022 769	1 965,843	702,710
	1024	16 MB	16 GB	12 371 900	7,039 314	114,814 303	2 327,500	142,700
	1 024	32 MB	32 GB	23 384 922	13,304 316	806,312 717	2 462,960	40,640
	16	INT_MAX	32 GB	1 662 721 364	9,190 874	814,555 646	3 565,276	40,228
B	1 024	16 kB	16 MB	12 700	0,008 139	0,022 769	1 965,843	702,710
	1 024	16 MB	16 GB	12 371 900	7,039 314	114,814 303	2 327,500	142,700
	1 024	32 MB	32 GB	23 384 922	13,304 316	806,312 717	2 462,960	40,639
	16	INT_MAX	32 GB	2 079 574 215	18,485 125	252,108 430	1 772,669	129,976
C	1 024	16 kB	16 MB	14 043	0,007 516	0,021 354	2 128,792	749, 274
	1 024	16 MB	16 GB	10 805 372	5,239 904	232,817 471	3 126,775	70,373
	1 024	32 MB	32 GB	21 915 623	10,626 684	517,460 051	3 083,558	63,325
	16	INT_MAX	32 GB	1 515 908 023	11,484 146	564,056 276	2 853,325	58,093

Tabuľka 5.7: Výsledky meraní funkcie RAND_bytes

Počítač	**Špecifikácie meraní**							
	NI	BS	V_D	ANC	T_A **[s]**	T_B**[s]**	$\approx P_A$ **[MB/s]**	$\approx P_B$ **[MB/s]**
A	1 024	16 kB	16 MB	16 877	0,006 082	0,005 921	2 630,714	2 702,246
	1 024	16 MB	16 GB	12 359 089	4,372 758	255,915 867	3 746,834	64,021
	1 024	32 MB	32 GB	24 664 863	8,726 003	650,850 067	3 755,213	50,346
	16	INT_MAX	32 GB	1 261 401 535	6,972 535	436,126 318	4 699,582	75,134
B	1 024	16 kB	16 MB	12 676	0,007 968	0,008 675	2 008,032	1 844,380
	1 024	16 MB	16 GB	12 450 851	7,084 268	116,442 791	2 312,730	140,704
	1024	32 MB	32 GB	23 206 632	13,202 893	1 006,623 945	2 481,880	32,552
	16	INT_MAX	32 GB	1 566 820 326	13,927 308	279,844 212	2 352,788	117,094
C	1 024	16 kB	16 MB	13 622	0,007 342	0,007 162	2 179,243	2 234,013
	1 024	16 MB	16 GB	10 483 219	5,083 700	217,651 658	3 222,849	75,276
	1 024	32 MB	32 GB	21 078 845	10,220 949	514,842 818	3 205,965	63,647
	16	INT_MAX	32 GB	2 431 597 459	18,421 180	613,641 577	1 778,822	53,399

Tabuľka 5.8: Výsledky meraní funkcie RAND_priv_bytes

6 Bezpečnostný problém pri RNG v prostredí VM

Virtuálny stroj (ďalej VM) je softvér, pomocou ktorého dokážeme vytvoriť abstraktné, respektíve virtuálne prostredie medzi našim počítačom a inou platformou. Celý tento proces je nezávislý od aktuálne použitého operačného systému. Používateľ dokáže takto spustiť aj aplikačné sady, ktoré nie su určené práve pre jeho OS. VM umožní inštaláciu virtuálneho OS na aktuálnom systéme pomocou zdieľania hardvérového vybavenia počítača. V dnešnej dobe je virtuálizácia vysoko populárna. Svoje uplatnenie našli v rôznych sférach. Príkladom použitia sú spoločnosti zamerané na virtuálizáciu sieti. Prostredníctvom jednoduchej náhrady alebo rozšírenia softvéru dokážu poskytnúť väčšiu spoľahlivosť a flexibilitu svojich služieb, bez nutnosti nákupu viacerých zariadení. Obdobne zasahuje virtuálizácia aj do procesu vývoja a testovania aplikácií.

Problematika VM je natoľko rozsiahla, že by mohla byť predmetom osobitnej práce. Avšak vzhľadom k aktuálnej problematike je opis základných princípov a metód týchto nástrojov vynechaný. Čitateľ má možnosť získať informácie prostredníctvom odkazu na video [59]. Autor v priebehu úvodných minút opisuje základné princípy virtuálizácie. Následne aj demonštruje spustenie konkrétneho linuxového operačného systému.

V publikácií [60], autori zrealizovali úspešný útok vo virtuálnom prostredí s OS Windows. Dokument opisuje útok na implementáciu TLS protokolu pri štandardne zabezpečenej komunikácii medzi serverom a klientom. TLS[1] sa používa na zabezpečenie komunikácie s využitím certifikátov pri distribúcií verejného kľúča. Absolútnym základom útoku je vedomosť tzv. bezpečnostnej zraniteľnosti pri obnovení snímky obrazu[2]. Následne autori dokázali pri opakovanom obnovení snímky extrahovať tajný kľúč servera. Ten vznikne deterministickým procesom pričom sa použijú náhodné dáta z RNG. Vlastník kľúča sa môže vydá-

[1] Z ang. *Transport Secure Layer*.

[2] Z ang. *VM reset vulnerabilities*.

vať za server a klient nedokáže rozoznať rozdiel.

Táto kapitola demonštruje aktuálny bezpečnostný problém v prostredí OS Windows na platforme VM, avšak so zameraním na proces generovania náhodných dát po obnovení snímky obrazu s uvedeným OS. Vzhľadom na využitie výstupov z RNG je teda teoreticky možné použiť uvedenú chybu aj pri pokuse o sieťový útok. Overenie aktuálnosti problému realizujeme experimentom. Použijú sa RNG rozhrania systému Windows[3] a knižnice OpenSSL[4], na generovanie kryptograficky bezpečných pseudo-náhodných čísel.

6.1 Opis bezpečnostného problému pri generovaní náhodných čísel po obnovení snímky obrazu VM s OS Windows

Pred opisom problému je nutné vysvetliť pojem snímka obrazu VM. Virtuálne stroje poskytujú možnosť spustiť ľubovoľný OS na počítači, bez nutnosti zmeny aktuálneho systému. Ďalšou z vymožeností tohto prostredia je možnosť vytvorenia kópie aktuálneho stavu pomocou tzv. snímky obrazu[5]. Pri tomto úkone dochádza k úplnému uloženiu stavu daného systému. Výsledkom je súbor, vďaka ktorému je možné kedykoľvek obnoviť systém do stavu, aký bol počas vytvorenia snímky. Vrátane všetkých údajov v operačnej pamäti. Využitie je výhodné najmä pri spúštaní alebo inštalácií programov z neznámych zdrojov. Tie môžu poškodiť OS. Ďalším príkladom je aktualizácia systému. Po znehodnotení OS je takto možné obnoviť systém do bodu kedy bolo všetko v poriadku. Po načítaní snímky je možné ďalej pokračovať v práci avšak iba s dátami z obdobia vzniku snímky.

VM reset vulnerabilities pomenúva pôvodný anglický názov pre bezpečnostné riziko. Doslovný preklad do slovenčiny nie je celkom výstižný. Preto zavedieme pomenovanie problému ako – Zraniteľnosť pri obnovení snímky obrazu VM. Po reštarte uvedeným spôsobom, je možné zreprodukovať aj výstupy systémových CSPRNG. Táto skutočnosť rapídne znižuje bezpečnosť algoritmov ako AES [17, kap. 5], ktorý sa používa pri symetrickom šifrovaní, resp. dešifrovaní dát práve pomocou tajného kľúča. Ďalším príkladom je DSA[6][17, kap. 11], používaný pri digitálnych podpisoch. Ak má útočník k dispozícií kľúče, s ktorými uvedené nástroje pracujú, ich použitie za účelom zabezpečenia komunikácie je bezvýznamné.

[3] Funkcia *BCryptGenRandom*.
[4] *Rand* príkaz v prostredí príkazového riadku.
[5] Z ang. *Snapshots*.
[6] Z ang. *Digital Signature Algorithm*.

Pri častom používaní rovnakej snímky obrazu si teda útočník môže všimnúť podobnosti kľúčov.

6.2 Experimentálne overenie zraniteľnosti

Realizáciu experimentu zabezpečí prenosný počítať A. Jeho konfigurácia je obsahom tabuľky 3.1. Pri tomto experimente je použitá iná metodika práce ako v [60]. Vykonanie jedného z pokusov zabezpečí CSPRNG API operačného systému Windows 10. Tento proces je vykonaný pomocou funkcia BCryptGenRandom [54]. Výstupy následne potvrdia bezpečnostné riziko.

Prvé kroky sú spojené s voľbou hypervízora. Voľne dostupný nástroj je VirtualBox [61][7]. V čase experimentu je dostupná 64-bitová verziu 6.0.24 (14. 07. 2020). Nasleduje voľba OS. Použili sa 64-bitový systém. Konkrétne Windows 10 Pro vo verzii 1909 (zostava 18363.592). Konfigurácia systému prebieha podľa video návodu [62]. V tomto návode autor priradil OS v prostredí VM 2 procesory (ďalej CPU[8]) My sme navýšili počet CPU na 4. Dôvodom je samotný OS, ktorý sa v prípade použitia konfigurácie so 4 CPU zdal rýchlejší. Následne sme ešte vykonali úpravu možnosti zdieľania súborov na obojsmernú. Tieto zmeny však neovplyvnia výsledok experimentu.

Prostredie Guest VM

Po úspešnej inštalácii obrazu a sprístupnení OS je nutná príprava prostredia.

Vykonané inštalácie:

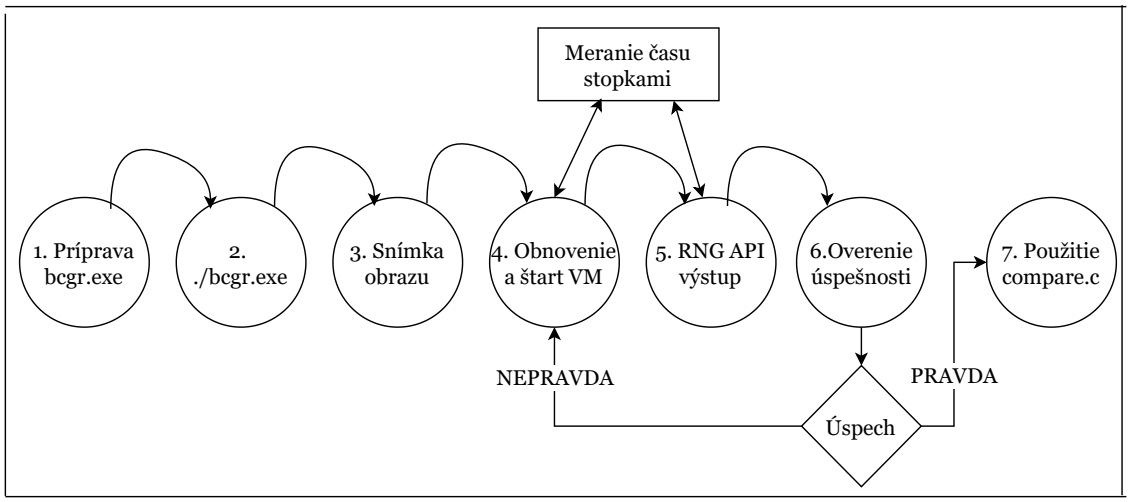
- GCC prekladač – Winlibs[9] GCC, 10.2.0, 64-bitová verzia, pre potreby prekladu programov do strojového kódu.
- VirtualBox Extension Pack – 6.0.24, zabezpečí ovládače pre VM.
- Visual Studio Code [63] – voľne dostupné prostredie na úpravu, kompiláciu, a aj spustenie programov pomocou príkazového riadka.

[7] Dostupný na: `https://www.virtualbox.org/wiki/Downloads`.
[8] Z ang. *Central Processing Unit*.
[9] Dostupný na: `https://winlibs.com/`.

6.2.1 Postup pri realizácií experimentu

Pokus je síce realizovaný prostredníctvom odporúčaného rozhrania na generovanie náhodných čísel – `BCryptGenRandom` [54]. Avšak akékoľvek systémové RNG API (`CryptGenRandom` i `RtlGenRandom`) dosahuje rovnaké výsledky. Toto tvrdenie sme aj úspešne overili.

Podstata nášho experimentu spočíva v presnom zopakovaní tých istých krokov od načítania snímky obrazu v približne rovnakom čase. Grafický postup znázorňuje schéma 6.1. V nasledujúcej časti je slovný opis.



Obr. 6.1: Grafické znázornenie krokov postupu

Postup:

1. Príprava a preklad jednoduchého programu `bcgr.exe` (`BCryptGenRandom`). Jeho úlohou je generovanie náhodných čísel pomocou systémového rozhrania Windows. Viď. ukážku zdrojového kódu tohto programu 6.1. Veľkosť výstupu vygenerovaných dát je v kóde stanovená na 1 920 B. Táto hodnota bola vykonaná pri poslednom experimente. Získaná je postupným násobením dvomi od veľkosti buffer-a 15 bajtov. Teda 15-30-60-120-...-1920. Pri hodnote 1920 B sme s navyšovaním skončili.

Zdrojový kód 6.1: Ukážka zdrojového kódu bcrypt.c

```
#include<stdio.h>
#include<windows.h>
#include<ntstatus.h>
//gcc bcrypt.c -o bcrypt -lbcrypt
int main(){
  int size= 1920;//size of data to generate in bytes
  BYTE   pbData[size];
  BCryptGenRandom(NULL,pbData,size,
                  BCRYPT_USE_SYSTEM_PREFERRED_RNG);
  for( int i = 0; i < size;i++ )
    printf("%02X",pbData[i]); //02X for same printing
  printf("\n");
  return 0;
}
```

2. Po príprave algoritmu je nutné aby sa používateľ presunul do bodu, tesne pred spustením programu `bcgr`, v príkazovom riadku. Pri experimente je použitý Windows PowerShell, prostredníctvom editora Visual Studio Code [63], v ktorom sme vykonali prípravu aj preklad programu.

3. V uvedenom bode je nutné vytvoriť snímku obrazu. Operačný systém bežiaci vo virtuálnom prostredí je možné následne vypnúť.

4. V tomto bode používateľ potrebuje pre dosiahnutie úspešnosti používať merací nastroj, napríklad stopky. Meranie je potrebné začať v ľahko identifikovateľnom mieste. Pri pokuse je aktivácia stopiek vykonaná v okamihu spustenia obnovenej snímky OS. Po sprístupnení systému sme spustili program `bcgr.exe`. V čase spustenie sme zastavili meranie času. Dôležité pre úspešný experiment je dosiahnuť spustenie `bcgr.exe` vždy v rovnakom čase. Tento krok nie je celkom jednoduchý a vyžaduje zručnosť používateľa.

   Pri pokuse sme program spúšťali približne 2 sekundy po sprístupnení systému. V čase od začiatku načítavania obrazu až po zapnutie programu nebolo počas experimentu vykonané nič. Žiaden pohyb myši ani stlačenia klávesnice. Toto je aj dôvod, prečo je snímka obrazu zaznamenaná pred zapnutím programu `bcgr.exe`.

5. Takto vzniknuté dáta je nutné uložiť mimo VM. Dôvodom je, že pri obno-

vení pomocou snímky dôjde k zahodeniu všetkých dát, ktoré nie sú jej obsahom. Vrátane dát vzniknutých v kroku 4.

6. V prípade neúspešnej realizácií experimentu je potrebné opakovať tento proces od kroku 4. Opakovanie je ukončené v prípade, že daná postupnosť už bola vygenerovaná. Stačí, ak sa používateľ zameria na prvé znaky výstupu. Dôvodom je, že ak by došlo k zmene len jedného bitu pred generovaním, tak výsledná postupnosť bude úplne iná. Tato vlastnosť je charakteristická pre generátor AES, ktorý OS Windows používa.

7. Hneď po zaznamenaní zhody v počiatočných hodnotách podrobíme dáta testovaniu zhody. Za účelom tohto procesu je vytvorený program `compare.exe`. Jeho úlohou je kontrola zhody v dvoch rovnako dlhých reťazcoch. Zdrojový kód je znázornený v ukážke 6.2. Program `compare.exe` je nutné pri každej zmene kompilovať pomocou GCC prekladača. Za zhodu je považovaná rovnosť hodnoty a polohy vo vygenerovanej postupnosti. Výstup poskytuje hodnotu nájdenej zhody v percentách. V prípade úplnej zhody je používateľ upozornený slovným výstupom.

Zdrojový kód 6.2: Ukážka zdrojového kódu compare.c

```
#include <stdio.h>
#include <string.h>
//gcc compare.c -o compare
/*
place for generated data
*/
int main(){
  char string1[]={"1.dataToCompare"};
  char string2[]={"2.dataToCompare"};
  int size=sizeof(string1)/sizeof(string1[0]);
  if(strcmp(string1, string2)==0){
    printf("ABSOLUTE␣MATCH\n");
  }else {
    int i=0,counter=0;
    while(i<size){
      if(string1[i]==string2[i])
      counter++;
      i++;
    }
    printf("MATCH␣in␣%.2f␣percent\n",
    (double)(counter*100)/size);
  }
  return 0;
}
```

Experiment s knižnicou OpenSSL v.1.1.1k

Po úspešnom prvom experimente so systémovými rozhraniami je overená i knižnica OpenSSL. Pri generovaní náhodných dát je použitý ako **zdroj náhodnosti** výstup z CSPRNG API systému, na ktorom je naimplementovaná zvolená funkcionalita. Teda ak je niektorá z funkcií na RNG implementovaná v OS Windows, tak vstupom OpenSSL generátora **je výstup funkcie BCryptGenRandom**. Periodicky dochádza k reseed procesu, pričom prvotná inicializácia OpenSSL generátora prebehne pri spustení.

Pri experimente boli aplikované rovnaké metódy ako pri softvérových rozhraniach. Generovanie sme vykonali v rozhraní OpenSSL, konkrétne cez prostredie `Command Prompt` (ďalej `cmd`). Knižnicu OpenSSL sme spustili príkazom `openssl`

v prostredí `cmd`. V spustenej aplikácii sme použili príkaz:

`rand -out result.txt -base64 100`.

Pred vykonaním sme vytvorili snímku obrazu. Úlohou príkazu je vygenerovať náhodné dáta s veľkosťou 100 bajtov a uložiť ich do textového súboru `result.txt` vo formáte *Base64* [64]. Pokus sme zopakovali 30-krát avšak ani raz sa nám pri experimente nepodarilo zreprodukovať výstup z generovania.

6.3 Vyhodnotenie výsledkov testovania

Vzhľadom k dobe, ktorá ubehla od uverejnenia článku [60], bolo očakávané úplné vyriešenie tohto bezpečnostného rizika. Počas experimentu sa podarilo vyvrátiť túto hypotézu. Dôkazom je videozáznam, ktorý demonštruje postup od 4. do 7 kroku experimentu. Záznam je obsahom prenosného média prílohy A, spoločne so zdrojovými kódmi použitými pri experimente. Videozáznam je možné vyhľadať aj na webovej platforme Youtube pomocou odkazu [65]. Vďaka videu je zrejmé, že pri používaní systémových RNG rozhraní isté bezpečnostný problém pretrváva.

Je však nutné poukázať na skutočnosť, že zopakovať výstup generovania pomocou systémových RNG API s rovnakým výsledkom, vyžaduje splnenie podmienok, ktoré užívateľ pri bežnej práci nevykonáva. Viď vyššie uvedený postup. Obdobne bolo potrebné počas testovania vykonať veľké množstvo pokusov. Niekedy bolo nutné počas generovaní konkrétnej postupnosti absolvovať reštart obrazu aj 20-krát.

Uvedený bezpečnostný problém však neplatí v prípade použitia kryptografickej knižnice OpenSSL. Pri testovaní bezpečnostného problému pri obnovení snímky obrazu v prostredí VM s OS Windows 10 sa ani raz nepodarilo zreprodukovať výstup z RNG rozhrania v tejto knižnici. Tento fakt potvrdzuje kvalitu spracovania kryptografickej knižnice OpenSSL, ktorá je aj pravidelne udržiavaná a aktualizovaná. V prípade zistenia nejakej chyby dochádza často k jej rýchlemu odstráneniu.

V porovnaní s [60], je situácia v dnešnej dobe pravdepodobne nenapodobiteľná. Je nutné uvedomiť si, že istý potenciál na útočenie existuje. Ak Windows 10 beží prostredníctvom hypervízora, tzv. Hyper-V, tak po obnovení následne dochádza k reseed-u stavov koreňového generátora systému. Týmto sa značne redukuje čas, kedy je možné uvedený bezpečnostný problém využiť. Jeho implementácia je dnes štandardná. Okrem toho väčšina softvérových hypervízorov pracuje aj v kooperácií s hardvérovými ako Intel VM-x a AMD-V. Jedinou podmienkou

je ich povolenie v BIOS-e. Na dnešných počítačoch je však možnosť virtuálizácie hardvéru štandardne zapnutá. Používateľ teda nie je nútený zasahovať do týchto nastavení.

Tento problém je dlhodobo známy fakt. Vzhľadom k veľkej popularite a rozvoju VM očakávame v blízkej dobe úplne odstránenie uvedeného bezpečnostného rizika

Aktualizácia softvérového vybavenia

V čase tvorby práce došlo k aktualizáciám softvérov použitých v tejto kapitole. Zmeny sa týkajú:

1. aktualizácia OS Windows – 10, Pro, 64-bitová verzia, v20H2, 19042.985,
2. GCC prekladač – Winlibs, 64-bitový, v11.1.0,
3. VirtualBox – 6.1.22 (vrátane doplnkov).

Testovanie bolo vykonané aj pomocou aktualizovaných nástrojov. Bezpečnostné riziko aj napriek uvedeným zmenám stále pretrváva. Aktualizovaný videozáznam ([66]) spoločne s údajmi o nástrojoch je taktiež obsahom prílohy A.

Je nutné informovať čitateľa o možnom probléme pri realizácii vyššie uvedených experimentov. Ak je na natívnom OS Windows použitý Windows Subsystém pre Linux vo verzii 2 (WSL2), tak je vysoko pravdepodobné, že nedosiahnete výsledky uvedené v tejto publikácií. Pred aplikovaním metód je potrebné deaktivovať túto funkciu. Dôvodom je interferencia medzi Hyper-V a týmto systémom. Občasne môže spôsobovať aj pády, respektíve nespoľahlivý výkon systému.

7 Vyhodnotenie dosiahnutých výsledkov

Obsahom tejto kapitoly je vyhodnotenie dosiahnutých výsledkov, ktoré sme získali počas experimentálnych meraní a štatistického testovania náhodných dát. Následne pomocou týchto dát vytvoríme odporúčanie pre použitie funkcií v prostredí OS Windows.

Vyhodnotenie štatistického testovania dát

V tabuľke 7.1 sú znázornené výsledky štatistického testovania náhodných dát pomocou sady NIST STS. Dáta z meraní jednotlivých funkcií sú obsahom prílohy A. Pomocou vyhodnotenia sady sme vypočítali celkovú úspešnosť na základe pomeru počtu úspešných a všetkých testov.

Pri spustení štatistickej sady na každý výstup náhodných dát testovanej funkcie sme použili rovnaké inicializačné parametre. Konkrétne, vstupná postupnosť $m = 1000000$bitov, počet prúdov $n = 4600$ a preddefinované hodnoty testov sme ponechali nezmenené. Pri uvedenej konfigurácii sady trvalo jedno testovanie okolo 10 hodín. Testy boli vykonané na počítači A. Špecifikácia tohto zariadenia je obsahom tabuľky 3.1.

Vzhľadom k výsledkom je potrebné informovať čitateľa, že aj napriek nižšej úspešnosti niektorých funkcií, boli dáta vygenerované z CSPRNG a môžeme ich považovať za kvalitné. Dôvodom neúspešnosti je citlivosť testov a snaha generátorov o rovnomerné rozloženie vygenerovaných dát (**uniformita dát**). Citlivosť sady je priamo úmerná veľkosti testovaných dát. Výsledná úspešnosť väčšiny testov musí byť približne 99 percent z celkového počtu vykonaných testov. Napríklad pri vykonaní 1000 testov je testovanie považované za úspešne, ak bolo úspešných aspoň 990 testov. Zároveň však platí, že pri veľkom počte testovaných dát sa dokonca očakáva, že dôjde k istým odchýlkam z množiny hodnôt, ktoré sú typické pre náhodné dáta. Tento jav súvisí s rovnomerným rozložením výstupov RNG. Uvedené fakty platia aj pre P-VALUE. Opis tejto hodnoty je obsahom kapitoly

Funkcia	Hodnoty			
	API	Testy celkovo	Úspešné testy	Úspešnosť [%]
RDSEED	AMD	188	188	100
RDRAND	AMD	188	186	98,94
RtlGenRandom	Windows	188	188	100
CryptGenRandom	Windows	188	186	98,94
BCryptGenRandom	Windows	188	187	99,47
rand_s	jazyk C	188	187	99,47
RAND_bytes	OpenSSL	188	188	100
RAND_priv_bytes	OpenSSL	188	188	100

Tabuľka 7.1: Vyhodnotenie kvality výstupov funkcií pomocou NIST STS

4.2.

Vyhodnotenie experimentálnych meraní

Na základe označenia prvkov 3. a priebežných experimentálnych meraní v 5. kapitole, je vytvorená tabuľka 7.2.

Poradie rozhraní v uvedenej tabuľke vzniklo nasledujúcim spôsobom. Vypočítali sme súčet ANC všetkých počítačov pri maximálnej možnej veľkosti BS. Takto vzniknuté číslo sme následne vydelili počtom meracích zariadení. Hodnoty jednotlivých rozhraní sme usporiadali vzostupne.

Funkcia	Hodnoty					
	Poradie	API	NI	BS	V_D	$\approx$ Priemer cyklov
RAND_bytes	1	OpenSSL	16	UINT_MAX	32 GB	$1{,}752 * 10^9$
RAND_priv_bytes	2	OpenSSL	16	UINT_MAX	32 GB	$1{,}753 * 10^9$
RtlGenRandom	3	Windows	8	ULONG_MAX	32 GB	$3{,}925 * 10^9$
BCryptGenRandom	4	Windows	8	ULONG_MAX	32 GB	$3{,}945 * 10^9$
CryptGenRandom	5	Windows	8	ULONG_MAX	32 GB	$4{,}593 * 10^9$
RDRAND	6	AMD	8	ULONG_MAX	32 GB	$1{,}445 * 10^{12}$
RDSEED	7	AMD	8	ULONG_MAX	32 GB	$3{,}098 * 10^{12}$
rand_s	8	jazyk C	8 589 934 592	uint32_t	32 GB	$2{,}302 * 10^{15}$

Tabuľka 7.2: Vyhodnotenie funkcií podľa výsledkov meraní

Celkové vyhodnotenie získaných výsledkov

Vzhľadom k faktu, že každá testovaná funkcia poskytuje kvalitné a kryptograficky bezpečné náhodné dáta, tak odporúčanie vykonáme na základe poradia tabuľky 7.2.

Knižnica OpenSSL ponúka kvalitne spracované rozhranie na generovanie kryptograficky bezpečných náhodných čísel. Úspešne zvládla experimenty, ktoré sú obsahom tejto práce. Vrátane bezpečnostného problému v prostredí VM s OS Windows. Generovanie náhodných čísel vykonáva s najmenším počtom potrebných inštrukcií. Ďalšími plusmi sú pravidelné aktualizácie, ľahká implementácia, možnosť použitia na rôznych typoch OS a voľná dostupnosť pre používateľa. Na základe uvedených faktov odporúčame používateľovi pri tvorbe kryptografických aplikácií na platforme Windows, použiť práve RNG rozhrania tejto knižnice. Výsledky oboch funkcií sú takmer rovnaké. `RAND_priv_bytes` by sme však mali implementovať v prípade, že nedochádza k ukladaniu vygenerovaných dát. Napríklad pri generovaní seed hodnoty a podobne.

V prípade systémových rozhraní dosiahla najlepšie výsledky pri meraní cyklov funkcia `RtlGenRandom`. Výhodou je jednoduchá implementácia. Problémom však ostáva jej podpora systému v budúcnosti. Prí tvorbe kryptografickej aplikácie je nevyhnutne nutné ošetriť kompatibilitu systému pomocou funkcie `BCryptGenRandom` alebo pomalšej `CryptGenRandom`. Ak dochádza k častému obnovovaniu snímok obrazu v prostredí virtuálnych strojov s OS Windows, tak použitie systémových RNG rozhraní neodporúčame vzhľadom k existujúcemu bezpečnostnému problému.

Používateľ má k dispozícií ešte procesorové inštrukcie `RDSEED` a `RDRAND`. Obidve poskytujú dostatočne kvalitné dáta. Najmä prvá z uvedených. Ich najväčšou nevýhodou je dĺžka potrebného času pri generovaní náhodných dát. Ak pri implementácií nie je proces náchylný na rýchlosť vykonávania tak odporúčame použitie funkcie `RDSEED`. V prostredí VM je nutná dodatočná konfigurácia hypervízora na povolenie prístupu k uvedeným inštrukciám.

Funkciu `rand_s` **neodporúčame** na použitie v kryptografických aplikáciach. Dôvodom sú dosiahnuté výsledkom meraní, ktoré naznačujú, že použitie hardvérových generátorov pomocou inštrukčných sád, je v tomto prípade lepšia alternatíva. Dôvodom je obmedzená veľkosť buffer-a na hodnotu 32 bitov. Táto funkcia je vhodná ako príklad na generovania náhodných dát.

8 Záver

Cieľom tejto práce bolo postupné uvedenie čitateľa do problematiky generovania náhodných čísel so zameraním na platformu Windows. Najprv sme vysvetlili základné pojmy súvisiace s témou práce. Následne sme pomocou systematickej klasifikácie charakterizovali nástroje na tvorbu náhodných dát, ktoré sú bežne využité v oblasti informačnej bezpečnosti. Zároveň sme vytýčili, ktoré z uvedených generátorov sú obsahom práce.

V úvode druhej kapitoly sme stručne opísali základné parametre operačných systémov. Vysvetlili sme dôležitosť kryptografických modulov a kvality výstupov generátorov pri bežnej prevádzke počítača. Následne sme charakterizovali historický vývoj systému Windows z hľadiska kryptografického vývoja. Vysvetlili sme základné princípy aplikované v prvom rozhraní – CAPI. Podrobnejší opis použitých metód sme však vykonali pri aktuálnom CNG API. So zameraním na tému tejto práce sme charakterizovali základné prvky systému. Opísali sme možnosti prístupu k rozhraniam. Najprv sme vysvetlili základné pojmy súvisiace s témou práce. Definovali sme čerpanie náhodnosti pri štarte a proces obnovovania počiatočnej inicializačnej hodnoty generátorov pomocou slova reseed.

Od praktík aplikovaných v kryptografických moduloch platformy Windows sme sa presunuli k špecifikácií metód, ktorými realizujeme experimenty na rozhraniach. Vytvorili sme označenia kvôli prehľadnejšej reprezentácii výsledkov a určili odchýlku meraní. Následne sme načrtli možnosti overovania výstupov pomocou sád štatistických testov. Opísali sme metódy vyhodnocovania v štatistickej testovacej sade NIST STS. Špecifikovali sme testy v súprave spoločne s meraním času potrebného na ich vykonanie. Pomocou uvedenej sady sme vykonali kontrolu kvality výstupných dát z RNG funkcií.

V piatej kapitole sme prešli k opisu možností generovania náhodných čísel. Popísali sme hardvérové a softvérové možnosti, ktoré má používateľ k dispozícii. Počas vykonávania jednotlivých funkcií sme aplikovali meracie metódy a výsledky reprezentovali vo forme tabuliek. V rámci overenia bezpečnosti generovania v prostredí virtuálnych stojov sme vykonali experiment. Špecifikácia prob-

lému, realizácia pokusu a následné zhodnotenie sú obsahom predposlednej kapitoly.

V poslednej časti tejto práce sme vyhodnotili naše experimentálne merania spoločne s výsledkami štatistických testov. Následne sme vo forme odporúčania určili funkcie, ktoré by mal čitateľ pri implementácií kryptografických aplikácii na platforme Windows použiť.

Pre účely rozšírenia tejto práce by bolo vhodné aplikovať zvolené merania viac z vnútra operačného systému. Konkrétne implementáciou kernel modulov a následným experimentom. V ďalšej fáze definovať presné správanie rozhraní pri ich volaní. Tieto údaje následne schematicky načrtnúť a opísať. Uvedený postup sa dá rozšíriť na viacero systémových modulov. V prípade zlepšenia aktuálnej epidemiologickej situácie vykonať štatistické testovanie s viacerými testovacími sadami a väčším množstvom prúdov.

Obsah 6. kapitoly tejto práce bol spracovaný a následne odoslaný ([67]) do Zborníka vedeckých prác TUKE FEI, pri príležitosti publikácie dosiahnutých výsledkov pri experimentoch v prostredí VM. Obdobne sme uverejnili celý obsah prílohy A na git stránky TUKE FEI katedier KEMT a KPI. Odkazy sú dostupné v prílohe A. Dôvodom je možnosť jednoduchšieho prístupu k dátam tejto práce v prípade potreby ich použitia. Archívy budú dostupné pre verejnosť až po registrácií tejto práce.

Literatúra

1. L'ECUYER, Pierre. Uniform random number generation. *Annals of Operations Research*. 1994, roč. 53, č. 1, s. 77–120. Dostupné tiež z: http://www.iro.umontreal.ca/~lecuyer/myftp/papers/tutaor-1994.pdf. [Online; Citované: 12.3.2021].
2. BARKER, Elaine; KELSEY, John; STANDARDS, National Institute of; TECHNOLOGY; COMMERCE, U.S. Department of. *NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2012. ISBN 1478169311. Dostupné tiež z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.228.2294. [Online; Citované: 12.3.2021].
3. HASINOFF, Samuel W. *Photon, Poisson Noise.* 2014. Dostupné tiež z: https://people.csail.mit.edu/hasinoff/pubs/hasinoff-photon-2012-preprint.pdf/. [Online; 6.3.2021].
4. QUANTIQUE, Company ID. *What is Q in the QRNG - Random number generation White Paper*. 1227 Carouge/Geneva, Switzerland, 2020. Dostupné tiež z: https://marketing.idquantique.com/acton/attachment/11868/f-0226/1/-/-/-/-/What%20is%20the%20Q%20in%20QRNG_White%20Paper.pdf/. [Online; 6.3.2021].
5. MYRVOLD, Wayne; GENOVESE, Marco; SHIMONY, Abner. Bell's Theorem. In: ZALTA, Edward N. (ed.). *The Stanford Encyclopedia of Philosophy*. Fall 2020. Metaphysics Research Lab, Stanford University, 2020. Dostupné tiež z: https://plato.stanford.edu/archives/fall2020/entries/bell-theorem/. [Online; 6.3.2021].
6. WATT, AD; MAXWELL, EL. Characteristics of atmospheric noise from 1 to 100 kc. *Proceedings of the IRE*. 1957, roč. 45, č. 6, s. 787–794. Dostupné tiež z: https://ieeexplore.ieee.org/abstract/document/4056603. [Online; 6.3.2021].

7. MCINTYRE, R. J. Multiplication noise in uniform avalanche diodes. *IEEE Transactions on Electron Devices*. 1966, roč. ED-13, č. 1, s. 164–168. Dostupné z DOI: 10.1109/T-ED.1966.15651.

8. MAROUANI, Hicham; DAGENAIS, Michel R. Internal clock drift estimation in computer clusters. *Journal of Computer Systems, Networks, and Communications*. 2008. Dostupné tiež z: https://www.hindawi.com/journals/jcnc/2008/583162/. [Online; 6.3.2021].

9. CORPORATION, SiTime. Clock Jitter Definitions and Measurement Methods. *SiT-AN10007 Rev 1.21*. 2019. Dostupné tiež z: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjRlN3hxefwAhUYAxAIHaotCCsQFjAJegQIAxAD&url=https%3A%2F%2Fwww.sitime.com%2Fapi%2Fgated%2FAN10007-Jitter-and-measurement.pdf&usg=AOvVaw1MIXIeLi3RSYMjCRhLRPIn. [Online; 6.3.2021].

10. BARKER, Elaine; DANG, Quynh. Nist special publication 800-57 part 1, revision 5:Recommendation for key management: Part 1–general. *NIST, Tech. Rep*. 2020. Dostupné tiež z: https://blkcipher.pl/assets/pdfs/NIST.SP.800-57pt1r5.pdf. [Online; Citované: 12.3.2021].

11. CHEON, Audrey Chaeyoung. The Influence of Pseudorandom Number Generators. *ANALYSIS OF APPLIED MATHEMATICS*. 2017, s. 80. Dostupné tiež z: http://www.analysisofappliedmathematics.org/wp-content/uploads/2016/09/AAM2.pdf#page=80. [Online; Citované: 12.3.2021].

12. SCHRIFT, Avital W; SHAMIR, Adi. On the universality of the next bit test. In: *Conference on the Theory and Application of Cryptography*. 1990, s. 394–408. Dostupné tiež z: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_29.pdf. [Online; Citované: 12.3.2021].

13. Initialization vector. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Initialization_vector. [Online; Citované: 12.3.2021].

14. Data masking. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Data_masking. [Online; Citované: 12.3.2021].

15. BARKER, Elaine. Recommendation for Digital Signature Timeliness. *NIST Special Publication*. 2009, roč. 800, s. 6. Dostupné tiež z: https://csrc.nist.rip/library/NIST%20SP%20800-102%20Recommendation%20for%20Digital%20Signature%20Timeliness,%202009-09.pdf. [Online; Citované: 12.3.2021].

16. DANG, Quynh; WOLFF, Otto J; CHANG, Shu-jen; DODSON, Donna F; KELSEY, John; PERLNER, Ray; POLK, W Timothy. NIST Special Publication 800-106 Randomized Hashing for Digital Signatures. 2009. Dostupné tiež z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.2967. [Online; Citované: 12.3.2021].

17. LEVICKÝ, Dušan. *Kryptografia v informačnej bezpečnosti*. Elfa, 2005.

18. DORRENDORF, Leo; GUTTERMAN, Zvi; PINKAS, Benny. Cryptanalysis of the random number generator of the windows operating system. *ACM Transactions on Information and System Security (TISSEC)*. 2009, roč. 13, č. 1, s. 16–17. Dostupné tiež z: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.1401&rep=rep1&type=pdf. [Online; Citované: 12.3.2021].

19. MCEVOY, Robert; CURRAN, James; COTTER, Paul; MURPHY, Colin. Fortuna: cryptographically secure pseudo-random number generation in software and hardware. 2006.

20. JEEVA, AL; PALANISAMY, Dr V; KANAGARAM, K. Comparative analysis of performance efficiency and security measures of some encryption algorithms. *International Journal of Engineering Research and Applications (IJERA)*. 2012, roč. 2, č. 3, s. 3033–3037. Dostupné tiež z: https://d1wqtxts1xzle7.cloudfront.net/28320314/SG2330333037-with-cover-page.pdf?Expires=1621881470&Signature=J80bk6ZBcDIdwBshWwXYaIokl7LXA6UocqttV5n6yKOYRj-3yhWHXWAdimDAPYMPoElfqSJWb2qaY1HlfSoEvmH6zNJb-ffXEu6ZIM4a94MppR~wTaRmSjngcV-ee6uOnvCDAS7mrjQREJjnfTEmaCqOBfk9gYewMpsHB4kdfzvemzIdo6aifC19JXpdFjLELIifUu6F6pyFC6AGDE-L3aNxKNRD8t8AH73NEk6VWdb9UvSJdAveQghaGJlcH3bb8-9lUGuTE4Aq-qI1CuGefflMpKZV-SVEcRJOQj23S8ZIv9WnNyCJwc8V3qGgwszYLezA6K~gyjtWCi5fYA3eHA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. [Online; Citované: 12.3.2021].

21. FIPS, PUB. 197: Advanced encryption standard (AES). *National Institute of Standards and Technology*. 2001, roč. 26.

22. STALLINGS, William. *Operating systems: internals and design principles*. Boston: Prentice Hall, 2012. Dostupné tiež z: https://repository.dinus.ac.id/docs/ajar/Operating_System.pdf. [Online; Citované: 12.3.2021].

23. SILBERSCHATZ, Abraham; PETERSON, James L; GALVIN, Peter B. *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc., 1991. Dostupné tiež z: http://www.uobabylon.edu.iq/download/M.S%202013-

2014/Operating_System_Concepts,_8th_Edition%5BA4%5D.pdf. [Online; Citované: 12.3.2021].

24. LIN, Chu-Hsing; YEH, Yi-Shiung; CHIEN, Shih-Pei; LEE, Chen-Yu; CHIEN, Hung-Sheng. Generalized secure hash algorithm: SHA-X. In: *2011 IEEE EUROCON-International Conference on Computer as a Tool*. 2011, s. 1–4. Dostupné tiež z: https://www.researchgate.net/profile/Chu-Hsing-Lin/publication/221290910_Generalized_secure_hash_algorithm_SHA-X/links/0c96052c81c724ce6a000000/Generalized-secure-hash-algorithm-SHA-X.pdf. [Online; Citované: 12.3.2021].

25. BASSHAM III, Lawrence E; RUKHIN, Andrew L; SOTO, Juan; NECHVATAL, James R; SMID, Miles E; BARKER, Elaine B; LEIGH, Stefan D; LEVENSON, Mark; VANGEL, Mark; BANKS, David L et al. *Sp 800-22 rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010. Dostupné tiež z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf. [Online; Citované: 12.3.2021].

26. BARKER, Elaine; ROGINSKY, Allen; BLANK, Rebecca; GALLAGHER, Patrick D.; SECRETARY, Under. *NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*. 2012. Dostupné tiež z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.252.3846&rep=rep1&type=pdf. [Online; Citované: 12.3.2021].

27. Entropy -infromation theory. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Entropy_(information_theory). [Online; Citované: 12.3.2021].

28. Cryptographic Service Provider (CSP). 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Cryptographic_Service_Provider. [Online; Citované: 12.3.2021].

29. DORRENDORF, Leo; GUTTERMAN, Zvi; PINKAS, Benny. Cryptanalysis of the random number generator of the windows operating system. *ACM Transactions on Information and System Security (TISSEC)*. 2007, roč. 13, č. 1, s. 1–32. Dostupné tiež z: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.1401&rep=rep1&type=pdf. [Online; Citované: 12.3.2021].

30. BROWN, Karen H. Security requirements for cryptographic modules. *Fed. Inf. Process. Stand. Publ*. 1994, s. 1–53. Dostupné tiež z: http://mayor.fri.uniza.sk/krypto/04/fips140-2.pdf. [Online; Citované: 12.3.2021].

31. ALZHRANI, Khudran; ALJAEDI, Amer. Windows and linux random number generation process: A comparative analysis. *International Journal of Computer Applications*. 2015, roč. 113, č. 8. Dostupné tiež z: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.7729&rep=rep1&type=pdf. [Online; Citované: 12.3.2021].

32. FERGUSON, Niels. Going in-depth on the Windows 10 random number generation infrastructure. 2019, s. 1–19. Dostupné tiež z: https://aka.ms/win10rng. [Online; Citované: 12.3.2021].

33. RDTSC – Read Time-Stamp Counter. [B.r.]. Dostupné tiež z: https://c9x.me/x86/html/file_module_x86_id_278.html. [Online; Citované: 12.5.2021].

34. PAOLONI, Gabriele. How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures. *Intel Corporation*. 2010, roč. 123. Dostupné tiež z: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf. [Online; Citované: 12.5.2021].

35. Trusted Platform Module. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Trusted_Platform_Module. [Online; Citované: 12.3.2021].

36. Advanced Configuration and Plan Interface. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface. [Online; Citované: 12.3.2021].

37. Unified Extensible Firmware Interface. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface. [Online; Citované: 12.3.2021].

38. Secure Hash Algorithm 2 (SHA-2). [B.r.]. Dostupné tiež z: https://en.wikipedia.org/wiki/SHA-2. [Online; Citované: 21.5.2021.

39. QueryPerformanceCounter function (profileapi.h). 2020. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter. [Online; Citované: 15.5.2021].

40. QueryPerformanceFrequency function (profileapi.h). 2020. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancefrequency. [Online; Citované: 15.5.2021].

41. Acquiring high-resolution time stamps. 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps. [Online; Citované: 12.5.2021].

42. HAMBURG, Mike; KOCHER, Paul; MARSON, Mark E. *Analysis of Intel's Ivy Bridge digital random number generator*. 2012. Tech. spr. Technical Report, Cryptography Research INC. Dostupné tiež z: https://cdn.atraining.ru/docs/Intel_TRNG_Report_20120312.pdf. [Online; Citované: 12.3.2021].

43. BROWN, Robert G. Dieharder: A Random Number Test Suite. 2003. Dostupné tiež z: https://webhome.phy.duke.edu/~rgb/General/dieharder.php. [Online; Citované: 11.5.2021].

44. SÝS, Marek; ŘÍHA, Zdeněk; MATYÁŠ, Vashek. Algorithm 970: optimizing the NIST statistical test suite and the berlekamp-massey algorithm. *ACM Transactions on Mathematical Software (TOMS)*. 2016, roč. 43, č. 3, s. 1–11. Dostupné tiež z: https://dl.acm.org/doi/abs/10.1145/2988228. [Online; Citované: 11.5.2021].

45. ENT A Pseudorandom Number Sequence Test Program. 2008. Dostupné tiež z: http://www.fourmilab.ch/random/. [Online; Citované: 25.5.2021].

46. TestU01-v.1.2.3 (18.08.2009). 2009. Dostupné tiež z: http://simul.iro.umontreal.ca/testu01/tu01.html. [Online; Citované: 25.5.2021].

47. RÁČEK, Ing. Tomáš. *Prahovací pravidla pro potlačovční šumu ve zvukových signálech*. 2010. Dostupné tiež z: https://www.vutbr.cz/studenti/zav-prace?zp_id=32092. [Online; Citované: 12.5.2021].

48. FOG., Agner. *Instruction tables*. 2021. Dostupné tiež z: https://www.agner.org/optimize/instruction_tables.pdf. [Online; Citované: 16.5.2021].

49. MECHALAS, John P. Intel Digital Random Number Generator (DRNG) Software Implementation Guide. *Intel Corporation*. 2014. Dostupné tiež z: https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html. [Online; Citované: 12.5.2021].

50. FIFO (computing and electronics). [B.r.]. Dostupné tiež z: https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics). [Online; Citované: 21.5.2021].

51. AMD. AMD Secure Random Number Generator Library. 2019, s. 1–10. Dostupné tiež z: https://developer.amd.com/wp-content/resources/AMD%20Secure%20Random%20Number%20Generator%20Library%202.0%20-Whitepaper.pdf. [Online; Citované: 12.5.2021].

52. RtlGenRandom function (ntsecapi.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-rtlgenrandom. [Online; Citované: 12.5.2021].

53. CryptGenRandom function (wincrypt.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom. [Online; Citované: 15.5.2021].

54. BCryptGenRandom function (bcrypt.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom. [Online; Citované: 15.5.2021].

55. OpenSSL Cryptography and SSL/TLS Toolkit. [B.r.]. Dostupné tiež z: https://www.openssl.org/. [Online; Citované: 15.5.2021].

56. ISO C Random Number Functions. [B.r.]. Dostupné tiež z: https://www.gnu.org/software/libc/manual/html_node/ISO-Random.html. [Online; Citované: 15.5.2021].

57. RAND – the OpenSSL random generator. [B.r.]. Dostupné tiež z: https://www.openssl.org/docs/man1.1.1/man7/RAND.html. [Online; Citované: 15.5.2021].

58. SAS, Quarkslab. OpenSSL Security Assessment – Technical Report – Ref. 18-04-720-REP. 2019, s. 1–35. Dostupné tiež z: https://eprint.iacr.org/2016/367.pdf. [Online; Citované: 15.5.2021].

59. NETWORKCHUCK. You need to learn Virtual Machines RIGHT NOW!! (Kali Linux VM, Ubuntu, Windows). 2021. Dostupné tiež z: https://www.youtube.com/watch?v=wX75Z-4MEoM%5C&ab%5C_channel=NetworkChuck. [Online; Citované: 23.5.2021].

60. RISTENPART, Thomas; YILEK, Scott. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In: *NDSS*. 2010. Dostupné tiež z: http://pages.cs.wisc.edu/~rist/papers/sslhedge.pdf. [Online; Citované: 16.5.2021].

61. Oracle VM Virtual Box. [B.r.]. Dostupné tiež z: https://en.wikipedia.org/wiki/VirtualBox. [Online; Citované: 21.5.2021].

62. MIKETHETECH. Installing Windows 10 on Virtualbox 6.1.12 – FULL PROCESS, 2020 – video tutorial. 2020. Dostupné tiež z: `https://www.youtube.com/watch?v=gKQvaPejxpc&ab_channel=MikeTheTech`. [Online; Citované: 17.5.2021].

63. Visual Studio Code. 2021. Dostupné tiež z: `https://code.visualstudio.com/`. [Online; Citované: 15.5.2021].

64. Base64. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/Base64`. [Online; Citované: 21.5.2021.

65. ROHAČ, Marek. Proof for security problem in Windows 10 v.1909 VM enviroment. 2021. Dostupné tiež z: `https://youtu.be/NTmtWbgm0gw`. [Online; Citované: 17.5.2021].

66. ROHAČ, Marek. Proof for security problem in Windows 10 VM enviroment – Update for version 20H2. 2021. Dostupné tiež z: `https://youtu.be/eaPwpNJcQr0`. [Online; Citované: 28.5.2021].

67. MAREK ROHAČ, Miloš Drutarovský. Bezpecnostné riziko pri generovaní nahodných dát v prostredí virtualnych strojov s OS Windows. 2021.

Zoznam príloh

Príloha A CD médium – viď. obsah CD média

A Obsah CD Média

Obsah tohto média je dostupný na gite:

- `https://git.kpi.fei.tuke.sk/marek.rohac/bc`
- `https://git.kemt.fei.tuke.sk/mr171hg/BachelorWork`

```
bc-master/ ..... Koreňový priečinok
└── appendixes/ ..... Prílohy
    └── BC_ZK/ ..... Zdrojové kody
        ├── amdINC/ ..... Knižnice pre AMD API
        │   ├── secrng.c
        │   └── secrng.h
        ├── demoExamples/ ..... Príklady demo kódov
        │   ├── openssl/ ..... Knižnice OpenSSL 1.1.1k
        │   │   ├── include/
        │   │   └── lib/
        │   ├── bcryptgenrandom.c
        │   ├── cryptgenrandom.c
        │   ├── cycleMeasure.c
        │   ├── libcrypto-1_1-x64.dll
        │   ├── makefile
        │   ├── openssl.c
        │   ├── rand.c
        │   ├── rands.c
        │   ├── README.md
        │   ├── rtlgenrandom.c
        │   └── timeMeasuring.c
        ├── openssl/ ..... Knižnice pre OpenSSL
        │   ├── include/
        │   │   └── openssl/
        │   └── lib/
        │       ├── pkgconfig/
        │       ├── libcrypto.a
        │       ├── libcrypto.dll.a
        │       ├── libssl.a
        │       └── libssl.dll.a
        └── Results/ ..... Výsledky experimentov
            └── Measurment_Results/ ..... Výsledky meraní
```

```
        ├── PCA/ ..... Výsledky kofigurácie A
        ├── PCB/ ..... Výsledky kofigurácie B
        └── PCC/ ..... Výsledky kofigurácie C
    ├── NIST_STS_Results/ ..... Výsledky štatistického testovania
        ├── bcryptgenrandom/
        ├── cryptgenrandom/
        ├── openssl/
        ├── rand_s/
        ├── rd_instructions/
        └── rtlgenrandom/
├── testingPrograms/ ..... Programy použité pre meranie
    ├── MEASURMENT/
        ├── bcryptgenrandom/
        ├── cryptgenrandom/
        ├── openssl/
        ├── rand_s/
        ├── rdinstructions/
        ├── rtlgenrandom/
        ├── README.md
        └── runall.bat ..... Skript pre automatizované spustenie
    ├── Modified_NIST_STS/ ..... Upravená sada NIST
        ├── experiments/
        ├── include/
        ├── obj/
        ├── src/
        ├── templates/
        ├── assess.exe ..... Spustiteľný program pre NIST STS
        └── makefile
├── VM_Experimnt/ ..... Dokumentácia k experimentu vo VM
    ├── experiment ossl/
        ├── ossl1/
        └── ossl2/
    ├── source_codes/ ..... Použité zdrojové kódy
        ├── bcrypt.c
        ├── compare.c
        └── makefile
    ├── Proof_OSv1909.mkv
    ├── Proof_UpdatedOS_v20H2.mkv
    └── README.txt
├── .gitignore
├── amdSPRNG.c ..... Implementácia AMD API
├── libcrypto-1_1-x64.dll
├── makefile
├── openssl_rng.c ..... Implementácia OpenSSL API
├── README.md ..... Dokumentácia k zdrojovým kódom
└── winAPIprng.c ..... Implementácia Windows API
├── prilohaa.tex ..... Zdrojový .tex prílohy A
└── prilohy.tex
```

- chapters/ ..... Zdrojové .tex kapitol práce
  - analysis.tex
  - bibliography.bib
  - evaluation.tex
  - introduction.tex
  - summary.tex
  - synthesis.tex
- figures/ ..... Obrázky a schémy práce v .pdf formáte
  - addcngtowin.pdf
  - cngprimarch.pdf
  - cryptoapi_arch.pdf
  - dizajnrdinstrukcii.pdf
  - os.pdf
  - osinit.pdf
  - rng sumar.pdf
  - rngapi.pdf
  - schema prístupu generátora.pdf
  - vm.pdf
  - zl.pdf
- .gitignore
- acronyms.tex ..... Zdrojový .tex zoznamu skratiek
- compile_template.bat ..... Skript pre kompiláciu šablóny
- glossary.tex
- kithesis.cls
- kithesis.pdf
- kithesis.sty
- kithesis.synctex(busy)
- latexmkrc
- README.md
- thesis.pdf ..... Bakalárska práca v .pdf
- thesis.tex ..... Zdrojový .tex koreňu šablóny