Univerzita Pardubice

Fakulta elektrotechniky a informatiky

# Implementace RNG pomocí dostupných zdrojů entropie v systémech Linux a Windows

Marek Farkas

Diplomová práce

2017

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
Akademický rok: **2016/2017**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Marek Farkas** |
| Osobní číslo: | **I15200** |
| Studijní program: | **N2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Implementace RNG pomocí dostupných zdrojů entropie v systémech Linux a Windows** |
| Zadávající katedra: | **Katedra softwarových technologií** |

Zásady pro vypracování:

Práce navazuje na bakalářskou práci *Dostupné zdroje entropie a jejich kvalita v systémech Linux a Windows*. Teoretická část vysvětlí pojmy týkající se entropie, hustoty informací a náhodnosti, popíše známé metodiky hodnocení kvality generátorů náhodných čísel a bude rozšířena o využití entropie v praxi, zejména v oblasti zabezpečení (kryptografie, kryptografické algoritmy a protokoly). Na základě výsledků výše uvedené BP bude pomocí dostupných zdrojů entropie implementován generátor náhodných čísel (RNG). Jeho výstup pak bude posouzen pomocí vhodných metod popsaných v teoretické části.

Rozsah grafických prací:

Rozsah pracovní zprávy: **40–50 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

***CACHIN, Christian: *Entropy Measures and Unconditional Security in Cryptography.* Zürich: Swiss Federal Institute of Technology, 1997. Disertační práce, školitel prof. Dr. U. Maurer. Dostupné též jako součást cyklu: Maurer, U.: ETH Series in Information Security and Cryptography. ISBN 3-89649-185-7. Dostupné též online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1016&rep=rep1&type=pdf**

***SUCIU, Alin – CAREAN, Tudor: *Benchmarking the True Random Number Generator of TPM Chips?* [online]. 2010. [cit. 2013-10-13]. URL:**

***CryptGenRandom Entropy. *Stack Overflow?* [online]. 2013. [cit. 2013-10-13]. URL: http://stackoverflow.com/questions/3486995/cryptgenrandom-entropy**

Vedoucí diplomové práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2016**

Termín odevzdání diplomové práce: **17. května 2017**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2016

## Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. 05. 2017

Marek Farkas

## Poděkování

Chtěl bych poděkovat panu Mgr. Tomáši Hudci a Emanueli Komínkovi za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat a také panu Ing. Zdeňku Šilarovi, Ph.D. za cenné rady z oblasti jazyka Java.

## Anotace

Tato diplomová práce vznikla jako rozšíření bakalářské práce jménem „Dostupné zdroje entropie a jejich kvalita v systémech Linux a Windows“. Cílem práce je implementace generátoru náhodných čísel pro operační systémy Linux a Windows za použití dostupných zdrojů entropie, které byly popsány ve výše zmíněné bakalářské práci. V diplomové práci jsou vysvětleny pojmy týkající se entropie v oblasti počítačů, jako je náhodnost, hustota informace a relativita informace. Dále jsou popsány existující metodiky hodnocení generátorů náhodných čísel. V diplomové práci je také popsána historie a využití entropie v praxi v oboru zabezpečení a kryptografie a dále i samotná implementace RNG.

## Klíčová slova

Entropie, náhodnost, bit, bajt, kryptografie, zabezpečení

## Title

Implementation of the RNG using available sources of entropy in the Linux and Windows OS

## Annotation

This diploma thesis was created as an extension of the bachelor thesis named "Available sources of entropy and their quality on Linux and Windows systems". The goal of thesis is to implement the random number generator for operating systems Linux and Windows using available sources of entropy, which were described in the aforementioned bachelor thesis. The thesis explains the concepts related to entropy in computer word, such as randomness, information density and relativity information. The thesis further describes the methodology for evaluation of existing random number generators. The thesis also describes the history and use of entropy in practice in the field of security and cryptography, as well as itself RNG implementation.

## Keywords

Entropy, randomness, bit, byte, cryptography, security

# Obsah

## Seznam obrázků



## Seznam tabulek

## Seznam zkratek a značek

| | |
|---|---|
| OS | operační systém (operating system) |
| RNG | generátor náhodných čísel (random number generator) |
| PRNG | pseudo-náhodný generátor čísel (pseudo random number generator) |
| TRNG | skutečný náhodný generátor čísel (true random number generator) |
| CPU | procesor (central processing unit) |
| DES | Data Encryption Standard |
| AES | Advanced Encryption Standard |
| SSL | Secure Sockets Layer |
| SSH | Secure Shell |
| WPA2 | Wi-Fi Protected Access |
| MD5 | Message-Digest algorithm |
| SHA | Secure Hash Algorithm |

# Úvod

Tato diplomová práce vznikla jako rozšíření bakalářské práce „Dostupné zdroje entropie a jejich kvalita v systémech Linux a Windows“, kde diplomová práce se bude zabývat implementací generátoru náhodných čísel pro operační systémy Linux a Windows, využívající dostupné zdroje entropie popsané v předcházející práci. V diplomové práci budou vysvětleny pojmy týkající se entropie, jako je hustota informací, náhodnost a relativita entropie. Dále budou popsány známé metodiky a statistické testy hodnocení kvality generátorů náhodných čísel.

V diplomové práci bude popsáno využití entropie v praxi, zejména v oblasti zabezpečení, jako je kryptografie. Budou také popsány existující kryptografické algoritmy a protokoly. Dále bude popsána implementace generátoru náhodných čísel a jeho výstup náhodných dat posouzen zmíněnými testy kvality generátorů náhodných čísel.

V kapitole zabývající se pojmy entropie bude popsána entropie v oblasti počítačů a vysvětleny pojmy hustota informací, náhodnost a relativita entropie. V kapitole hodnocení kvality generátorů náhodných čísel budou popsány metodiky hodnocení kvality generátorů náhodných čísel. V kapitole entropie v praxi bude popsána historie kryptografie až do dnešní doby počítačů, budou zmíněny principy kryptografie a šifrování a popsány používané kryptografické algoritmy a protokoly. V další kapitole, zdroje entropie, budou stručně popsány jak vhodné, tak i nevhodné zdroje entropie pro implementaci generátoru náhodných čísel. V poslední kapitole implementace generátoru náhodných čísel bude popsána vlastní implementace RNG za použití vhodných zdrojů entropie na základě výsledků z výše zmíněné bakalářské práce.

Na konci budou uvedeny přílohy k práci, jako je implementace RNG, zdrojové kódy, dokumentace implementace RNG a ukázka části zdrojových kódů v programovacím jazyku Java.

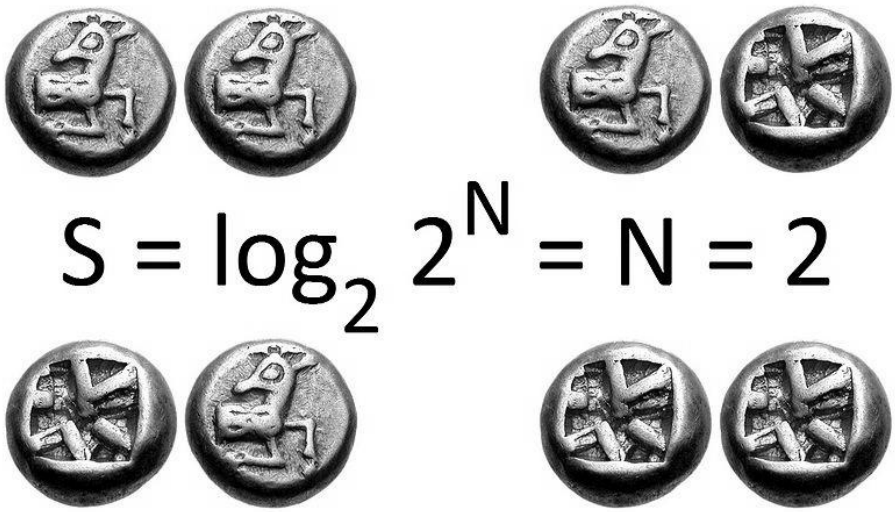
# 1. Pojmy související s entropií

## Entropie

Text této kapitoly (Pojmy související s entropií) byl převzat z mé vlastní bakalářské práce[1] z kapitoly Entropie, na kterou tato práce navazuje. Některé části byly přeformulovány či vypuštěny.

Entropie, v teorii informatiky, je míra nejistoty v náhodné proměnné. V tomto kontextu tento termín obvykle odkazuje na Shannonovu entropii, která kvantifikuje očekávanou hodnotu informací obsažených ve zprávě. Entropie se měří obvykle v bitech, natech (logaritmická jednotka o základu e) nebo banech (logaritmická jednotka o základu 10). Shannonova entropie je průměrná nepředvídatelnost náhodné veličiny, která je ekvivalentní jejímu informačnímu obsahu, poskytuje také absolutní limit na největší možné kódování bezeztrátové komprese nebo jakékoli komunikace za předpokladu, že sdělení je reprezentováno jako posloupnost nezávislých a stejně rozdělených náhodných veličin.[1]

Spravedlivý hod mincí má entropii jednoho bitu, série dvou hodů má entropii dvou bitů, počet reálných hodů mincí je tedy její entropie v bitech. Tento náhodný výběr mezi dvěma výsledky v pořadí v čase (nezáleží, zda jsou výsledky stejně pravděpodobné) se často označuje jako Bernoulliho proces. Výsledek entropie procesu je dán binární entropickou funkcí, míra entropie pro spravedlivé hody je jeden bit na hod, nicméně v případě, že mince není spravedlivá, nastupuje nejistota a tím se snižuje míra entropie, protože jsme-li požádáni o předpovězení dalšího výsledku, můžeme zvolit nejčastější výsledek před méně častým. Neshoda vzniká mezi tím, co víme nebo předvídáme a informací, že nespravedlivý hod nám ukázal méně než jeden rub či líc neboli bit hodu.[1]




**Obrázek 1 – Entropie hodu mincí[27]**

Takto představil definici entropie Claude E. Shannon v jeho díle „A Mathematical Theory of Communication“ v roce 1948.[1]

Při práci s počítači pojem entropie znamená náhodnost, shromážděná operačním systémem nebo aplikacemi, pro použití v kryptografii nebo pro jiné účely, které vyžadují náhodná data. Tato náhodnost je často shromažďována z hardwarových zdrojů, a to včetně například počítačové myši nebo speciálně zavedených hardwarových zařízení za účelem poskytnutí náhodných generátorů.[1]

### Hustota informací

Při šifrování je entropie používána jako míra nepředvídatelnosti krypto-grafického klíče. Například 128bitový klíč, který je náhodně vygenerován, má 128 bitů entropie, což je $2^{128-1}$ nutných kombinací k prolomení hrubou silou. Nicméně entropie nedokáže zachytit počet pokusů nutných k rozšifrování klíče, může být uhodnut na první pokus anebo také ne. Zvažme 1000místné binární číslo (položku). Pokud položka má 1000 bitů entropie, je to skvělé, pokud má 999 bitů entropie rovnoměrně rozložené, lze entropii stále považovat za skvělou. Ale v případě, že položka má 999 bitů entropie, kde první číslice je pevná a zbývajících 999 číslic je naprosto náhodných, je první číslice šifrovaného textu neentropická.[1]

Ačkoli je entropie často používána jako charakteristika informačního obsahu zdroje dat, tento informační obsah není absolutní (záleží na pravděpodobnostním modelu). Zdroj, který vždy generuje stejný symbol má míru entropie 0, ale definice toho co symbol znamená, závisí na abecedě. Například zdroj, který produkuje řetězec A-B-A-B-A-B…, v němž následuje A a B periodicky. Pravděpodobnostní model považujeme za jednotlivá písmena jako nezávislý, míra entropie sekvence je 1 bit na znak, ale pokud je posloupnost považována za AB-AB-AB…, tedy symbol jako dvojmístný blok, pak míra entropie je 0 bitů na znak.[1]

S hustotou informace také souvisí komprese dat. Komprese dat může být bezeztrátová nebo ztrátová, kde bezeztrátová snižuje bity souboru a odstraňuje statickou nadbytečnost dat. Ztrátová komprese snižuje bity souboru odstraněním zbytečné informace. Když máme maximálně zkomprimovaný soubor, jeho entropie dosahuje téměř ideálního zdroje.[1]

### Náhodnost

Náhodnost znamená nedostatek vzoru neboli předvídatelnost akcí, nepořádek nebo nesoudržnost posloupnosti kroků tak, že zde není žádný srozumitelný vzor nebo kombinace. Využití ve vědě, matematice a statistice rozpoznává nedostatek předvídatelnosti odkazem na nahodilost, ale připouští zákonitosti ve výskytu události, jejich výsledky jsou nejisté. Například při hodu dvěma kostkami a počítání celku můžeme říct, že součet 7 se bude náhodně vyskytovat dvakrát častěji, jak 4. V těchto situacích náhodnost znamená určitou míru nejistoty. Náhodnost je často používána ve statistice, kde znamená dobře definované statistické vlastnosti. Monte Carlovy metody, které jsou založeny na náhodném vstupu, jsou důležitými technikami ve vědě (hlavně v počítačové).[1]

### Relativita entropie

Užitečným měřítkem entropie je takzvaná relativita entropie, která funguje stejně dobře jak v diskrétním, tak spojitém případě distribuce relativní entropie. Definicí jest Kullback-Leiblerova odchylka od rozdělení na referenční míru $p$ a je absolutně spojitá vzhledem k opatřením $m$, tedy p formulace $(\mathrm{d}x)=f(x)\ m(\mathrm{d}x)$ pro nezáporné m integrované funkce $f$ s $m$ prvního integrálu, pak relativní entropie může být definována jako:

$$D_{\mathrm{KL}}(p||m) = \int \log\big(f(x)\big)\, p\ \mathrm{d}x = \int f(x) \log\big(f(x)\big)\, m\ \mathrm{d}x$$

kde f(x) – funkce
p – míra
m – opatření

V této podobě relativní entropie zobecňuje jak diskrétní entropie je. Je-li opatření m samo o sobě rozdělení pravděpodobnosti, relativní entropie je nezáporná, nulová pokud p=m. Pro každé opatření je definován prostor a proto koordinace nezávislé a neměnné v souřadných re-parametrizací, pokud jedna řada zohledňuje transformaci opatření m. Relativní, implicitní a diferenciální entropie závisí na referenčním opatření m.[1]

V praxi nás však zajímá, kolik naší entropie útočník zná a kterou znát nemůže, jelikož informace, které útočník zná, je entropie = 0 a je tedy úplně zbytečnou. V praxi také zajímají generátory, které generují náhodnost hned po spuštění, jelikož funkce generující náhodnost v nekonečnu může být náhodná, na začátku svého definičního oboru může být klidně konstantní.[1]


**Obrázek 2 – Relativita entropie[28]**

# 2. Hodnocení kvality generátorů náhodných čísel

Text této kapitoly byl převzat z mé vlastní bakalářské práce[1] s výjimkou částí (odstavců), za kterými je uveden odkaz na jiný zdroj.

Počítačově generovaná náhodná čísla jsou často využívána v aplikacích od her až po komplexní numerické integrace. Pokud se v programu používají jakákoli náhodná čísla nebo pokud se jedná o vlastní implementaci generátoru náhodných čísel, mělo by být zajištěno a ověřeno, že čísla jsou náležitě náhodná.[4]

Je třeba si uvědomit, že všechna počítačově generovaná náhodná čísla nejsou náhodná, vzhledem k výchozímu bodu a generátoru jsou sekvence náhodných čísel generovaných v počítači předvídatelné, proto jsou tato čísla správně nazývána pseudonáhodnými čísly, i když se většinou zjednodušeně nazývají náhodnými čísly.[4]

Jediný způsob, jak zjistit, zda generátor náhodných čísel je dostatečně kvalitní, je skrz pečlivé testování. Pro ověření dostatečnosti náhodnosti generátorů čísel se používají statistické testy, i když test může určit, zda generátor produkuje čísla, která jsou skutečně náhodná, může také určit, zda jsou náhodná čísla dostatečně náhodná pro jejich zamýšlené použití. Například počítačová hra Solitaire používá při zobrazování karet generátor náhodných čísel. Vzhledem k výchozímu bodu a generátoru je sled zobrazovaných karet naprosto předvídatelný. Toto pozorování však není relevantní pro každého, kdo hraje hru a generátor nerozpoznal. Důležitou otázkou je to, zda se zobrazené karty zdají být nepředvídatelné vzhledem k informacím, které uživatel má.[4]

Dobrý generátor náhodných čísel má dva hlavní ukazatele, vysokou kvalitu náhodných dat a vysokou rychlost získávání náhodných dat, měřenou v Mb/s (nebo v MB/s). Kvalita náhodných dat je z hlediska hodnocení mnohem důležitější než rychlost získávání náhodných dat, avšak požadujeme oboje. Pro testování náhodných dat se dá použít program ENT.[1]

### Chi-kvadrát test

Základní myšlenka tohoto testu je podobná testu KS (Kolmogorovův-Smirnovův, popsán níže). Hlavní rozdíl spočívá v tom, že namísto použití každého náhodného čísla jednotlivě jsou čísla seskupena do skupin s netriviálními frekvencemi v každé skupině. Obvyklým pravidlem je, že každá skupina by měla v průměru obsahovat alespoň pět poznatků. Test KS je obecně lepší test, zda je sada náhodných čísel v souladu s rovnoměrným rozdělením, ale test chi-kvadrát může být užitečný, pokud hodláte seskupit pozorování z generátoru.[4]

Tento test je nejběžněji používaný test pro náhodnost dat, je velmi citlivý na chyby v pseudonáhodných sekvencích generátorů. Rozdělení chi-kvadrát testu je vypočten pro proud bajtů v souboru a je vyjádřen jako absolutní číslo a procento, které ukazuje, jak často pravá náhodná sekvence překročila vypočtenou hodnotu. Procento interpretuje, do jaké míry je testovaná sekvence podezřelá, že není náhodná. Pokud je toto procento vyšší než 99 % nebo menší než 1 %, sekvence je téměř jistě nenáhodná. Procento mezi 99 % a 95 %

nebo mezi 1 % a 5 % znamená, že sekvence je podezřelá. Procenta mezi 95 % a 90 % a 5 % a 10 % značí, že sekvence je téměř podezřelá, hodnoty mezi 90 % a 10 % znamenají, že náhodná sekvence je náhodná. [1]

### Aritmetický průměr

Výsledek je prostý součet všech bajtů, který je vydělen délkou souboru. Pokud jsou údaje náhodné, výsledek by měl být kolem 127,5 v případě bajtu (0,5 v případě bitů).[1]

### Monte Carlova hodnota Pi

Každá následující sekvence šesti bajtů se používá jako 24 bitů X a Y souřadnic ve čtverci. Je-li vzdálenost náhodně generovaného bodu menší než poloměr kružnice vepsané do čtverce, sekvence šesti bajtů je považována za „hit“. Podíl zobrazení může být použit pro výpočet hodnoty Pi.[1]

### Sériový korelační koeficient

Tento koeficient měří rozsah, v němž každý bajt v souboru závisí na předchozím bajtu. Pro náhodné sekvence se tato hodnota bude blížit nule.[1]

### Histogram

Histogram je grafické znázornění distribuce dat, je to odhad rozdělení pravděpodobnosti spojité proměnné.[1]

### Kolmogorovův-Smirnovův test

Kolmogorovův-Smirnovův test (KS-test) se snaží zjistit, zda se dva soubory dat významně liší. KS-test má tu výhodu, že nebere v potaz distribuci dat (je bezparametrický a od distribuce oddělený). Tato obecnost má však i svou cenu, další testy (například t test) mohou být přesnější, pokud údaje splňují požadavky testu.[2]

### Spektrální test

Spektrální test může být aplikován na všechny generátory, kde s-rozměrné vektory pseudonáhodných čísel tvoří mřížkovou strukturu, obvykle považován za sadu překrývajících se vektorů. Tato sada vykazuje mřížkovou strukturu pro mnoho generátory pseudonáhodných čísel, jako je lineární kongruentní, roztroušená rekurzivní kongruentní, lagged-Fibonacci, add-with-carry, subtract-with-borrow generátory, kombinované lineární kongruentní generátory a kombinované více rekurzivní generátory. Spektrální test měří maximální vzdálenost mezi sousedními paralelními nadrovinami, tedy maximální vzdálenost všech rodin paralelních nadrovin, které pokrývají všechny vektory.[3]

### Test kolizí

Tento test je inverzní test chi-kvadrát testu. Místo seskupování pozorování k vytvoření pravděpodobnosti velkého počtu pozorování v každé skupině, kolizní test seskupuje

pozorování tak, že opakované hodnoty jsou relativně nepravděpodobné. Opakované hodnoty jsou nepravděpodobné, protože test nastaví velké množství skupin vzhledem k délce subsekvence a možným hodnotám náhodných čísel.[4]

Počet bitů použitých z náhodných čísel je nastaven uživatelem. Počet náhodných čísel je nastaven na 16 384 ($2^{14}$) a počet skupin je nastaven na 1,045,876 ($2^{20}$), tyto konkrétní hodnoty navrhuje Knuth. Tento test se nazývá kolizní test, protože test zkoumá ty skupiny, které mají nadměrný počet opakovaných hodnot nebo kolizí.[4]

### Testy nejbližších hodnot

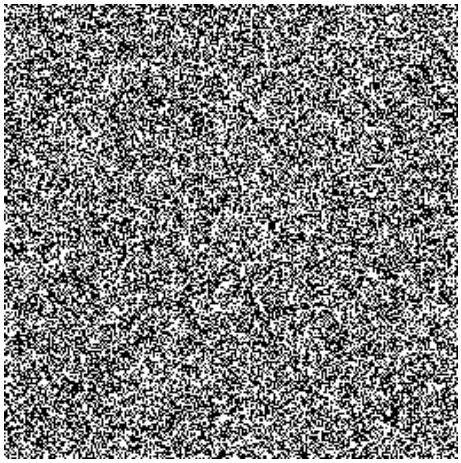
Spíše než zkoumáním frekvence výskytu hodnot v subsekvencích náhodných čísel, tyto statistické testy zkoumají vztah mezi blízkými hodnotami podle vzorů. Tyto testy zkoumají blízké hodnoty, aby zjistili, zda nejsou v rozporu s nezávislými okolními hodnotami. Protože generátory náhodných čísel používají předchozí hodnoty k určení následných hodnot, blízké hodnoty nejsou zjevně nezávislé. Tato postupná generace náhodných čísel neznamená, že se náhodné číslice nezdají nezávislé. Zkoušky v této části zkoumají subsekvence, tedy zda existuje zřejmá sekvenční závislost, kterou mohou testy zjistit.[4]

### The Run Test

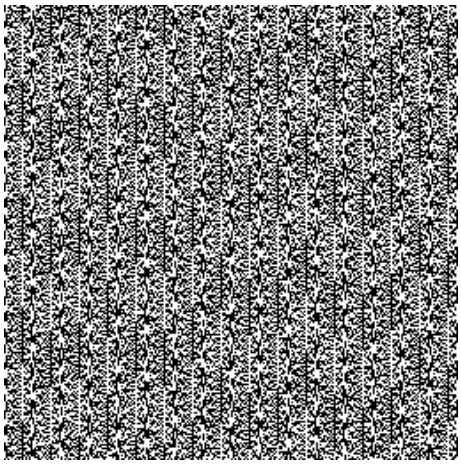
Spíše odlišnou, ale velmi silnou zkouškou pro sekvenční náhodnost je test běhu. Příklad běhu délky tři je 4, 5, 6, následovaný 2 (tento druh běhu je známý jako „run-up“). Test zkoumá subsekvence náhodných čísel a počítá počet běhů, které mají délky 1, 2, 3, 4, 5 a 6 nebo více. Tento test určuje, zda dochází k příliš častým výpadům nebo ne dost často k výskytu sady rovnoměrně distribuovaných nezávislých náhodných čísel.[4]

### Vizuální analýza

Jedním ze způsobů, jak prozkoumat generátor náhodných čísel, je vytvořit vizualizaci čísel, jež produkuje. Lidé jsou opravdu dobří při sledování vzorků a vizualizace vám umožňuje používat oči a mozek přímo pro tento účel. Ačkoli byste tento typ přístupu neměli považovat za vyčerpávající nebo formální analýzu, je to pěkný a rychlý způsob, jak získat hrubý přehled o výkonu daného generátoru. Níže porovnání bitmapy generátoru RANDOM.ORG, který je True Random Number Generator (TRNG) a bitmapou s funkcí rand() z PHP na Microsoft Windows, což je pseudo-náhodný generátor čísel (PRNG).[30]



**Obrázek 3 – TRNG Bitmapa[30]**



**Obrázek 4 – PRNG Bitmapa[30]**

Jak je vidět z obrázků, bitmapa vygenerovaná generátorem pseudonáhodných čísel PHP/Windows ukazuje jasné vzory ve srovnání s generovaným generátorem skutečných náhodných čísel.[30]

# 3. Entropie v praxi

„Pojem entropie se často používá pro označení různých nesouvisejících jevů. V informačních technologiích má entropie vliv na šifrování (TLS/SSL) neboli kryptografii a sestavování (inicializaci) generátorů náhodných čísel, pomocí kterých se testují programy pro náhodné vstupy a vytváří klíče pro zabezpečenou komunikaci po síti.“[1]

Kryptografie je metoda ochrany informací transformací (šifrováním) do nečitelného formátu, nazývaného šifrovaný text. Pouze ti, kteří mají tajný klíč, mohou rozluštit (dešifrovat) zprávu do původního textu. Šifrované zprávy mohou být někdy rozluštěny kryptoanalýzou, ačkoli moderní kryptografické techniky jsou prakticky nerozbitné.[7]

## 3.1 Historie kryptografie

### Historie

Kryptografie je jednou z nejstarších oblastí technického učení, o kterém se dají nalézt záznamy alespoň 4 000 let nazpět. Je zajímavé, že ze všech kryptosystémů vyvinutých za těchto 4 000 let úsilí zůstaly pouze 3 systémy s rozsáhlejším využitím, které dosahují dostatečné odolnosti k prolomení. Jeden z nich potřebuje mnoho prostoru, aby byl prakticky použitelný, druhé je příliš pomalý pro většinu praktických použití a o třetím se obecně předpokládá, že obsahuje vážné slabiny.[5]

Moderní kryptologie je poměrně mladá věda. Ačkoli kryptografie byla používána po tisíce let pro skrývání tajných zpráv, systematické studium kryptologie jako vědy (a možná i umění) začalo asi před sto lety.[6]

Kryptologie byla (a stále je v jisté míře) zahalena do pojmu mystiky pro většinu lidí, právě proto společnost spojovala kryptografii s černým uměním. Často se o kryptografii říkalo, že se zabývá komunikací s temnými duchy a vyvinul si pro svoje učení špatný obraz. Většina prvních kryptografů byli vědci, ale obyčejní lidé byli často přesvědčeni, že jsou také následovníky ďábla.[6]

### Antická kryptografie

Kryptografie se, podle současných indicií, poprvé objevila kolem roku 2000 př. n. l. ve starodávném Egyptě, kde byly použity hieroglyfy k ozdobení hrobů zemřelých vládců a králů. Tyto hieroglyfy vyprávěly příběh o životě krále a hlásaly velké skutky jeho života. Hieroglyfy byly záměrně zašifrovány, ale očividně neměly skrýt text, spíše se zdá, že šifrování bylo zamýšleno hlavně proto, aby byl text krásnější a vypadal důležitě. Během času se tyto spisy stávaly čím dál tím komplikovanějšími a nakonec lidé ztratili zájem o jejich rozluštění a samotné praktikování šifrování také vymizelo.[5] [6]

Důkazy použití kryptografie byly zaznamenány ve většině významných časných civilizací. Antická práce „Arthshashtra“ napsaná Kautalyou popisuje špionážní službu v Indii a zmiňuje udělení úkolů špiónům v šifrovaném textu. V Indii tedy byla kryptografie zjevně pokročilejší a vláda používala tajné kódy pro komunikaci se sítí špionů nacházejících se po celé zemi. Původní indické šifry se většinou skládaly z jednoduchých abecedních substitucí, často založených na fonetice. Některé z nich byly mluvené nebo používané jako znaková řeč.[5] [6]

Starověcí Číňané používali ideografickou povahu svého jazyka, aby skryli význam slov. Zprávy byly často transformovány do ideogramů pro utajení zprávy, ale žádného velkého použití v časných čínských vojenských tažená se nedočkali, například Čingischán nikdy nevyužil kryptografii.[6]

Historie kryptografie v Mezopotámii byla podobná kryptografické historii Egypta, jelikož bylo používáno klínové písmo k šifrování textu. Tato technika byla také použita

v Babylonu a Asýrii. V Bibli se také občas používá hebrejská šifrovací metoda. V této metodě je poslední písmeno abecedy nahrazeno prvním písmenem a naopak, tato šifra se nazývá „atbash“.[6]

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

**Obrázek 5 – Atbash šifra[6]**

V slavném řeckém dramatu Iliad byla použita kryptografie, když byl Bellerophon poslán ke králi se zašifrovaným dopisem, která králi řekla, že má usmrtit Bellerophona, král ho proto poslal bojovat s několika mytickými tvory.[6]

Sparťané používali systém, který sestával z tenkého listu papyru zabaleného kolem tyče. Zprávy byly zaznamenány po délce tyče na papyrus, aby mohla být zpráva přečtena, musel být papyrus zabalen kolem tyče stejného průměru. Tato šifra je nazývaná „skytale“, byla použita v 5. století př. n . l. pro posílání tajných zpráv mezi řeckými válečníky. Bez správné tyče bylo obtížné dekódovat zprávu pomocí technik dostupných v té době.[6]

```
ADGJMPSVY
BEHKNQTWZ
CFILORUX

ADGJMPSVYBEHKNQTWZCFILORUX
```

**Obrázek 6 – Skytale šifra[6]**

Další řeckou kryptografickou metodu vyvinul Polybius (nyní nazvaná Polybiusův čtverec). Písmena abecedy byla rozdělena na čtverec o 5 vertikálních a 5 horizontálních řádcích písmen (podobně jako pozdější metoda Playfair). Řádky a sloupce jsou číslovány 1 až 5, takže každé písmeno má odpovídající pár (řádek, sloupec). Tyto páry mohou být snadno signalizovány pochodněmi nebo ručními signály. Dešifrování se skládá z mapování dvojic číslic zpět do jejich odpovídajících znaků. Tento systém byl první, který snižoval velikost sady symbolů a ve volném slova smyslu by mohl být považován za předchůdce moderních binárních reprezentací znaků.[6]

```
\ 1 2 3 4 5
 \_________
1|A B C D E        T=54
2|F G H I J        H=32    5344 44 4435
3|K L M N O        I=42    4224 24 3211
4|P Q R S T        S=44
5|U V W X Y/Z
```

**Obrázek 7 – Polybiusův čtverec[6]**

Julius Ceasar použil systém kryptografie (Caesarova šifra), který posunul každé písmeno o dvě místa dále v abecedě (např. Y posune na A, R přesune na T atd.). Vzdálenost posunutí není pro princip šifry důležitý a ve skutečnosti ani vybrané lexikální uspořádání. Obecným případem tohoto druhu šifry je monoalfabetická substituční šifra, kde každé písmeno je mapováno do jiného písmena.[6]

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
CDEFGHIJKLMNOPQRSTUVWXYZAB
```

**Obrázek 8 – Caesarova šifra[6]**

### Středověká kryptografie

Arabský matematik Al-Kindi napsal knihu o kryptografii nazvanou Risalah fi Istikhraj al-Mu'amma (rukopis pro dekódování kryptografických zpráv), přibližně v roce 800 n. l. Popsal první techniky kryptoanalýzy (i pro některé polyalfabetické šifry), šifrovací klasifikaci, arabskou fonetiku a syntaxi, ale co je nejdůležitější, popsal použití několika technik kryptoanalýzy, a vysvětlil je na frekvenční analýze.[8]

Arabové byli první, jenž učinili významný pokrok v kryptoanalýze. Arabský autor Ahmad al-Qalqashandi (1355-1418) napsal techniku pro dešifrování, která se používá dodnes. Technikou je zapisovat všechna písmena šifrovaného textu a počítat frekvenci každého symbolu. Při použití průměrné četnosti každého písmena daného jazyka lze získat původní text. Tato technika je dostatečně silná, aby dešifrovala libovolnou monoalfabetickou substituční šifru, pokud je k dispozici dostatek šifrovaného textu.[6]

V podstatě všechny šifry zůstaly zranitelné touto kryptoanalytickou technikou až do vývoje polyalfabetické šifry panem Leone Battista Alberti (cca 1465). Leon Battista Alberti je znám jako „Otec západní kryptologie“ díky jeho vývoji polyalfabetické substituce. Polyalfabetická substituce je technika, která umožňuje, aby různé znaky šifrového textu reprezentovaly stejný symbol původního textu. Tím je obtížnější interpretovat šifrový text pomocí frekvenční analýzy. Aby vyvinul tuto techniku, Alberti analyzoval metody pro lámání šifer a navrhl šifru, která se nedala rozšifrovat těmito technikami.[6] [8]

### Novověká kryptografie

Další významný pokrok byl v roce 1518 německým mnichem Trithemiusem. Napsal sérii šesti knih s názvem Polygrafie a v páté knize navrhl tabulku, která opakovala abecedu duplikující řádek nad ním, který se posunul o jedno písmeno. Chcete-li zakódovat zprávu, první písmeno textu je zakódováno prvním řádkem tabulky, druhé písmeno druhým řádkem a podobně.[6]

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ Plaintext
FGUQHXSZACNDMRTVWEJBLIKPYO T00
OFGUQHXSZACNDMRTVWEJBLIKPY T01
YOFGUQHXSZACNDMRTVWEJBLIKP T02
PYOFGUQHXSZACNDMRTVWEJBLIK T03
    ...
GUQHXSZACNDMRTVWEJBLIKPYOF T25
```

**Obrázek 9 – A Changing Key cipher[6]**

Nejslavnějším kryptografem 16. století byl Blaise de Vigenere (1523-1596). V roce 1585 napsal „Tracte des Chiffres“, ve kterém použil tabulku výše zmíněného mnicha Trithemiuse, ale změnil způsob fungování klíče. Jedna z jeho technik používala prostý text

jako vlastní klíč. Jiná použila šifrovaný text. Způsob, jakým jsou tyto klíče používány, je známý jako plánování klíče a je nedílnou součástí „Data Encryption Standard“ (DES).[6]

V roce 1628 francouz Antoine Rossignol pomohl armádě porazit Huguenoty dekódováním zachycené zprávy. Po tomto vítězství byl mnohokrát povolán, aby vyřešil šifry pro francouzskou vládu. Použil dva seznamy, aby vyřešil své šifry, jeden, ve kterém byly hladké elementy v abecedním pořadí a kódové prvky byly náhodně rozmístěné a druhý pro usnadnění dekódování, v němž byly kódové prvky postaveny v abecedním nebo číselném pořadí, zatímco jejich prosté ekvivalenty byly neuspořádané. Když Rossignol zemřel v roce 1682, jeho syn a později jeho vnuk pokračoval v jeho práci. V té době byla francouzskou vládou zaměstnána řada kryptografů, společně tvořili „Cabinet Noir“ (Černá komora).[6]

Během 1700, černé komory byly běžné v Evropě, jedna z nejznámějších byla ve Vídni, nazývána „Geheime Kabinets-Kanzlei“ a byla řízen Baronem Ignaz de Koch mezi 1749 a 1763. Tato organizace četla celou poštu příchozí ze zahraničních velvyslanectví, kopírovaly dopisy a vrátil je na poště stejného rána. Stejný úřad také zpracovával všechny ostatní politické nebo vojenské zásahy a někdy četl až 100 dopisů denně. Anglickou černou komoru vytvořil John Wallis v roce 1701. Do té doby řešil šifry pro vládu v řadě neoficiálních pozic. Po smrti v roce 1703 převzal jeho vnuk William Blencowe, který byl vyučen jeho dědečkem, a získal titul Decypherer. Anglická černá komora měla dlouhou historii vítězství v kryptografickém světě.[6]

V koloniích neexistovala centralizovaná kryptografická organizace. Dešifrování bylo prováděno převážně zainteresovanými jednotlivci. V roce 1775 byl dopis zachycený od církve Dr. Benjamina podezřelý, že je britským kódovaným poselstvím, nicméně američtí revolucionáři to nemohli rozluštit. Jejich problém vyřešil Elbridge Gerry, který se později stal pátým viceprezidentem a Elisha Porter. Toto poselství prokázalo, že církev je vinná ze snahy o informování konzervativců a byla později vyhoštěna. Benedikt Arnold použil kód, v němž každý korespondent má přesnou kopii téhož kódového seznamu. Každé slovo prostého textu je nahrazeno číslem, které udává jeho polohu v knize (např. 3.5.2, strana 3, řádek 5, slovo 2). Revolucionáři také používali během války šifry, Samuel Woodhull a Robert Townsend poskytli generálovi George Washingtonovi mnoho informací o síle a pohybech britských vojsk v New Yorku a okolí. Kód, který použili, sestával z čísel, které nahradily otevřená slova. Tento kód napsal major Benjamin Tallmadge. Pro další utajení také používali neviditelný inkoust.[6]

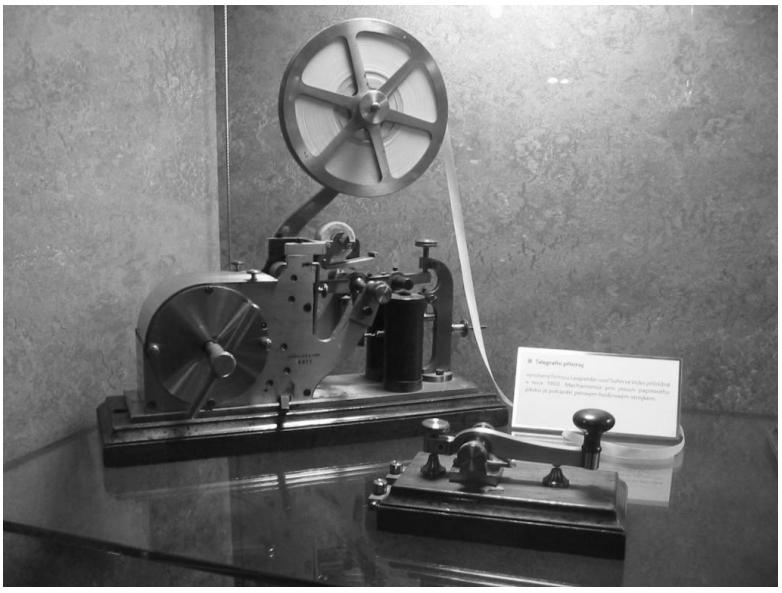
„Wheel cipher“ byla vynalezena Thomasem Jeffersonem kolem roku 1795 a ačkoli s ním nikdy moc nepracoval, velmi podobný systém byl používán námořnictvem U.S. ještě před několika lety. „Whell cipher“ byla tvořena sadou kol, každá s náhodným uspořádáním písmen abecedy. Klíčem k systému je uspořádání, ve kterém jsou kola umístěna na nápravě. Zpráva je kódována zarovnáním písmen podél osy otáčení nápravy tak, aby se vytvořila požadovaná zpráva. Jakýkoliv jiný řádek zarovnaných písmen lze pak použít jako šifrovací text pro přenos. Dešifrování vyžaduje, aby příjemce zarovnal písmena šifrového

textu podél osy otáčení a nalezl sadu zarovnaných písmen, které mají lingvistický význam jako prostý text. Existuje velmi malá pravděpodobnost, že z procesu dešifrování budou dvě čitelné zprávy, ale toto může jednoduše ověřit původce textu.[6]

```
        GJTXUVWCHYIZKLNMARBFDOESQP
W1
        IKMNQLPBYFCWEDXGZAJHURSTOV
W2
        HJLIKNXWCGBDSRVUEOFYPAMQZT
W3
                  ...
        BDFONGHJIKLSTVUWMYEPRQXZAC
Wn
```

**Obrázek 10 – Wheel cipher[6]**

V roce 1844 byl vývoj kryptografie dramaticky změněn vynálezem telegrafu. Komunikace telegrafem nebyla v žádném případě bezpečná, takže byly potřebné šifry pro přenos tajných informací. Zájem veřejnosti o kryptografii rozkvétal a mnozí jednotlivci se pokoušeli formulovat vlastní šifrovací systémy. Příchod telegrafu poskytl poprvé možnost komunikace v reálném čase na velkou vzdálenost, což bylo používáno hlavně ve vojenství pro komunikaci velení s frontou.[6]



**Obrázek 11 – Telegraf[9]**

Systém Playfair byl vynalezen v roce 1854 Charlesem Wheatstoneem a Lyon Playfair a byl prvním systémem, který pro šifrování používal pár symbolů. Abeceda je umístěna v náhodném čtverci 5 na 5 znaků a text je rozdělen na sousední dvojice. Použití je jednoduché, ale není těžké šifru zlomit. Skutečným průlomem v tomto systému bylo použití dvou znaků najednou. Účelem je, aby statistika jazyka byla méně výrazná, a proto

zvýšila množství práce a množství šifrového textu potřebného k dešifrování. Tento systém se omezeně používal za Druhé světové války a byl velmi účinný proti Japoncům.[6]

```
   IKMNQ
   LPBYF

PLAINTEXT =  PL AI NT EX TZ
   CWEDX       =              =
   GZAHU

LPMGMOXEAS = LP MG MO XE AS
   RSTOV
```

**Obrázek 12 – Playfail system[6]**

V roce 1863 vyvinul pan Kasiski metodu kryptoanalýzy, která dešifrovala téměř každou existující šifru té doby. Principem bylo najít opakování řetězců znaků v šifrovacím textu. Vzdálenost mezi těmito opakováními se pak používá k nalezení délky klíče. Protože opakování identicky šifrovaného identického textu se vyskytují na vzdálenostech, které jsou násobkem délky klíče, nalezení největších společných dělitelů opakovaných vzdáleností vede k délce klíče. Jakmile je délka klíče známá (N), vytvoří se statistiky na každém N-znaku a frekvence použití znamená, jaký znak představuje v této sadě šifrovaných symbolů.[6]

Během občanské války (1861-1865) nebyly šifry příliš složité. Mnoho technik spočívalo pouze v psaní slov v jiném pořadí a nahrazení kódových slov pro vlastní jména a umístění. Tam, kde měla Unie centralizovanou šifrovací kontrolu, měla konfederace tendenci nechat polní velitelé rozhodovat o svých vlastních formách šifer. Systém Vigenere byl široce využíván veliteli polní a někdy vedl k tomu, že Unie rozluštila zprávy rychleji než jejich příjemci. Konfederace používala tři klíčová slova ve většině svých zpráv během války, „Manchester Bluff“, „Complete Victory“ a „Come Retribution“, která byla rychle nalezena třemi kryptoanalyzátory Unie Tinkerem, Chandlerem a Batesem a zprávy byly pravidelně dešifrovány Unií. Použití běžných slov jako klíčů pro kryptosystémy způsobilo, že se objevily zprávy z čitelného textu. Použití běžných slov pro hesla je ve skutečnosti nejobvyklejším vstupním bodem v moderních počítačových systémových útocích.[6]

V roce 1883 Auguste Kerckhoffs napsal „La Cryptographie Militaire“, ve kterém definoval šest základních požadavků na kryptografii:[6]

- Šifrovaný text by měl být v praxi nerozluštitelný.
- Kryptosystém by měl být vhodný pro korespondenty.
- Klíč by měl být snadno zapamatovatelný a měnitelný.
- Šifrovací text by měl být přenositelný telegrafem.
- Šifrovací zařízení by mělo být snadno přenosné.
- Šifrovací stroj by měl být poměrně snadno použitelný.

Na počátku 20. století byla v Evropě na spadnutí válka, Anglie vynaložila značné úsilí na zlepšení svých kryptonalytických schopností, takže když válka začala, dokázali dešifrovat většinu nepřátelských šifer. Kryptoanalytická skupina byla nazývána „Room 40“ kvůli své počáteční poloze v konkrétní budově v Londýně. Jejich největší úspěchy byly při řešení německých námořních šifer. Tato řešení byla značně zjednodušena, protože Němci často používali politická nebo nacionalistická slova jako klíče.[6]

Stejně jako telegraf změnil kryptografii v roce 1844, rádio změnilo kryptografii v roce 1895. Nyní byly přenosy otevřené pro kohokoli, kdo chce odposlouchávat a fyzická bezpečnost už nebyla možná. Francouzi měli od 1. světové války mnoho rozhlasových stanic a zachytili většinu německých rozhlasových vysílání. Němci používali dvojitý sloupcový přechod, který nazývali "Ubchi", který byl snadno rozbit francouzskými kryptoanalyzátory.[6]

V roce 1917 vytvořili Američané kryptografickou organizaci MI-8. Analyzovali všechny typy tajných zpráv a pokračovali s velkým úspěchem během a po 1. světové válce, ale v 1929, Herbert Hoover se rozhodl skončit, protože si myslel, že je nesprávné číst cizí soukromé zprávy. Yardley byl během finanční krize v úzkých a proto napsal knihu popisující práci MI-8 s názvem „Americká černá komora“. Mnozí ho kritizovali za to, že prozradili tajemství a oslavovali své vlastní akce během války.[6]

Až do roku 1917 byly přenosy odesílány přes telegrafní kabely kódovány Baudotovým kódem pro teletypy. Americká telefonní a telegrafická společnost se velmi zajímala o to, jak snadno je lze číst, takže Gilbert S. Vernam vyvinul systém, který spojuje elektronické impulsy prostého textu s klíčem pro vytváření šifrovacích impulzů. Použití vylo občas složité, protože klíče byly těžkopádné. Vernam vyvinul stroj na šifrování zpráv, ale systém nebyl nikdy široce používán.[6]

Dalším významným pokrokem v elektromechanické kryptografii byl vynález rotoru. Rotor je tlustý disk se dvěma čelními plochami, každý s 26 mosaznými kontakty oddělenými izolačním materiálem. Každý kontakt na vstupní ploše je připojen pomocí drátu k náhodnému kontaktu na výstupní (šifrovací) masce. Každému kontaktu je přiřazen znak. Elektrický impuls aplikovaný na kontakt na vstupní ploše způsobí, že z čipového textu bude vyvedeno jiné písmeno. Jednoduchý rotor tedy implementuje monoalfabetickou substituční šifru. Tento rotor je nastaven v zařízení, které přijímá vstupní text z klávesnice psacího stroje a odešle odpovídající elektrický impuls do masky prostého textu. Šifrovací text je generován z rotoru a vytištěn nebo přenášen.[6]

Další krok odděluje rotor od předchozích systémů. Po každém písmu je rotor otočen tak, aby se celá abeceda posunula o jedno písmeno. Rotor je tedy progresivní klíčová polyalfabetická substituční šifra se smíšenou abecedou a s periodou 26. Pak je přidán druhý rotor, který posune svou pozici o jedno místo, když první rotor dokončí každou rotaci. Každý elektrický impuls je poháněn oběma rotory tak, aby byl dvakrát šifrován. Vzhledem k tomu, že se oba rotory pohybují, abeceda má nyní periodu 676. Když se přidá více rotorů, doba se dramaticky zvyšuje. U 3 rotorů je doba 17 576, 4 je 456 976 a 5 je 11

881 376. Aby šifra 5 rotorů byla rozluštěna frekvenční analýzou, musí být šifrovací text extrémně dlouhý.[6]

Systém rotoru může být přerušený, protože pokud se objeví opakování v prvních 26 písmenech, kryptoanalyzátor ví, že se pohyboval pouze první rotor a že spojení se měnila pouze tímto pohybem. Každá po sobě následující sada 26 písmen má tuto vlastnost a pomocí rovnic může kryptoanalyzátor zcela určit daný rotor, čímž eliminuje jeden rotor z celého problému. To se může opakovat u každého po sobě jdoucího rotoru, jakmile se stane známý předchozí rotor, s dodatečnou výhodou, že se doba prodlužuje, takže je zaručeno, že bude mít mnoho opakování. To je poměrně složité dělat ručně. První rotorový stroj vynalezl Edward Hugh Hebern v roce 1918.[6]

Během prohibice byl alkohol přepravován do země nelegálními pašeráky, kteří používali kódovanou radiovou komunikaci k ovládání nelegální dopravy a pomohli vyhnout se hlídkám pobřežní stráže. K udržení pobřežní stráže ve tmě používali pašeráci složitý systém kódů a šifer. Pobřežní strážník najal paní Elizebeth Smithovou Friedmanovou, aby rozluštil tyto kódy, a tak přinutil pašeráky používat složitější kódy a častěji měnit své klíče.[6]

### Kryptografie za Druhé světové války

Asi nejznámější z tohoto období z oblasti kryptografie je stroj Enigma, byl široce používán nacistickým Německem. Ve druhé světové válce se využívalo mechanických a elektromechanických šifrovacích strojů nebo manuálních, pokud to situace vyžadovala. Velký pokrok byl dosažen jak v šifrování, tak v kryptoanalýze, vše v utajení. Informace o tomto období začaly být odtajňovány, jelikož oficiální 50leté britské období utajení skončilo, a jak se archivy pomalu otevřely, objevily se různé záznamy a články.[8]

Němci používali elektromechanický rotorový stroj známý jako Enigma. Matematik Marian Rejewski na polském šifrovacím úřadu v prosinci 1932 vyvodil podrobnou strukturu německé armádní Enigmy pomocí matematiky a omezené dokumentace, což byl největší průlom v kryptoanalýze za tisíc let, podle historika Davida Kahna. Rejewski a jeho kolegové Bureau, Jerzy Różycki a Henryk Zygalski, dále dešifrovali Enigmu a drželi krok s vývojem komponent německého vojenského stroje a šifrovacích postupů. Jak se válka objevila, šifrovací úřad ve Varšavě zasvětil francouzské a britské zpravodajské zástupce do tajemství dešifrování Enigmy. Na konci války byli britští špičkoví vojenští důstojníci informováni, že nikdy nemohou odhalit, že německá šifra byla porušena, protože by dala poraženému nepříteli příležitost říci, že nebyli poraženi férově.[8]

**Obrázek 13 – Enigma[8]**

Kryptografové amerického námořnictva, ve spolupráci britských a nizozemských kryptografů, se po roce 1940 dostali do několika kryptografických systémů japonského námořnictva. Prolomení do jednoho z nich, JN-25, vedlo k vítězství USA v bitvě u Midway. Skupina amerických armád, SIS, dokázala předtím, než začala druhá světová válka, překonat nejvyšší bezpečnostní japonský diplomatický šifrovací systém (elektromechanický „krokový přepínač“).[8]

### Moderní kryptografie

Éru moderní kryptografie odstartoval Claude Shannon (někdy nazýván též jako otec matematické kryptografie), prací, kterou vypracoval během druhé světové války o bezpečnosti komunikace. V roce 1949 publikoval „Communication Theory of Secrecy Systems“ v technickém deníku a později knihu „Mathematical Theory of Communication“. Tyto, kromě jeho dalších prací na teorii informace a komunikace, vytvořili pevný teoretický základ pro kryptografii a také pro kryptoanalýzu a tím kryptografie víceméně zmizela pod křídla tajných vládních komunikačních organizací, jako je NSA, GCHQ a jejich ekvivalenty jinde. Do sedmdesátých let 20. století byla kryptografie z velké části pod kontrolou vlády, dvě hlavní události ji v té době přinesli do veřejné sféry, vytvoření veřejného šifrovacího standardu (DES) a Vynález kryptografie s veřejným klíčem.[8]

První zmíněná událost, šifra DES byla předložena výzkumnou skupinou v IBM na výzvu Národního úřadu pro normalizaci (nyní NIST) v úsilí rozvoje bezpečnosti elektronických komunikačních zařízení pro podniky, jako jsou banky a jiné velké finanční organizace. Po poradě a úpravách ze strany NSA, jenž byl v zákulisí, byl přijat a publikován v roce 1977 (nyní FIPS 46-3). DES byla první veřejně přístupná šifra, která byla „požehnána“ národní agenturou, jako je NSA. Vydání specifikace podnítilo vzestup veřejného a akademického zájmu o kryptografii.[8]

Druhá událost v roce 1976 byla možná ještě důležitější, neboť zásadně změnila způsob fungování kryptosystémů, a to bylo vydání publikace „New Directions in Cryptography“ od Whitfielda Diffieho a Martina Hellmana. Byla zavedena nová metoda distribuce šifrovacích klíčů, která popsala obrovský pokrok v řešení jednoho ze základních problémů kryptografie, distribuci klíčů a stala se známá jako výměna klíčů Diffie-Hellman. Článek také podnítil téměř okamžitý veřejný vývoj nové třídy, asymetrických šifrovacích algoritmů. Předtím byly všechny užitečné moderní šifrovací algoritmy symetrické, ve kterých se stejný kryptografický klíč používá v algoritmu jak odesílatele, tak příjemce, kteří ho musí udržovat v tajnosti. Všechny elektromechanické stroje používané ve WWII byly z této třídy, stejně jako šifry Caesar a Atbash a v podstatě všechny šifrovací systémy v historii.[8]

## 3.2 Kryptografie

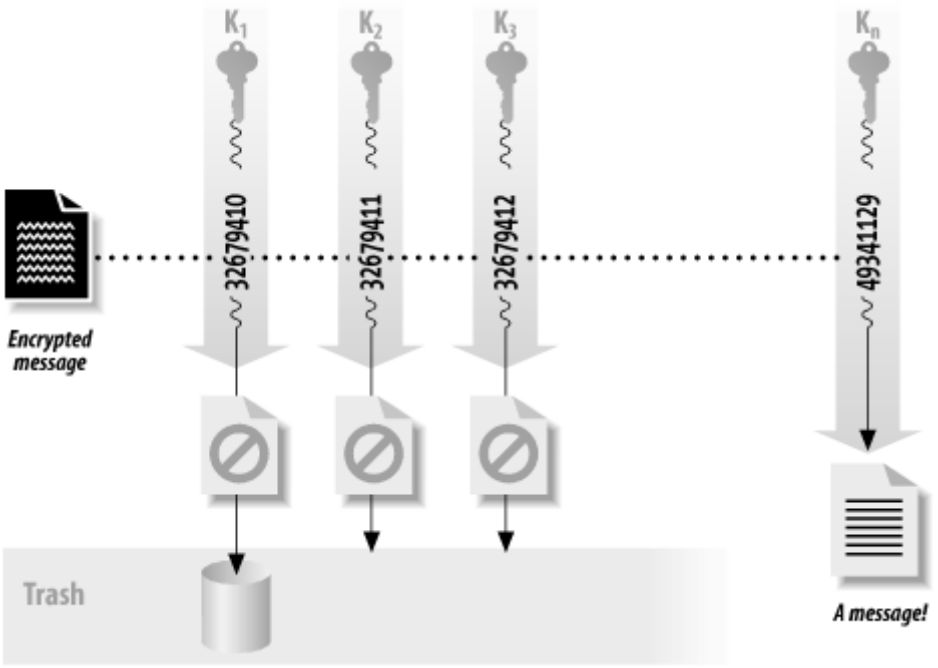
Vzhledem k tomu, že internet a další formy elektronické komunikace se stávají stále používanějšími, elektronická bezpečnost roste na důležitosti. Kryptografie slouží k ochraně e-mailových zpráv, informací o kreditní kartě a firemních dat. Jeden z nejpopulárnějších kryptografických systémů používaných na internetu je „Pretty Good Privacy“ (PGP).[7]

Systémy kryptografie lze obecně klasifikovat do systémů symetrických klíčů, které používají jediný klíč, který mají jak odesílatel, tak i příjemce, a systémy veřejných klíčů, které používají dva klíče, veřejný klíč známý všem a soukromý klíč, který používá pouze příjemce zpráv.[7]

Uvnitř počítače je kryptografický klíč reprezentován jako řetězec binárních číslic. Každá binární číslice může být 0 nebo 1. Pokud je klíč dlouhý 1 bit, existují dva možné stavy: 0 a 1. Obecně každý přidaný bit klíči zdvojnásobuje počet možných stavů:[11]

$$Počet\ stavů = 2 * (délka\ klíče)$$

Pokud se útočník pokouší dešifrovat zprávu a nemá kopii klíče, nejjednodušší způsob dešifrování zprávy je útok hrubou sílu (nalezení všech klíčů pro zjištění, zda jeden z nich dešifruje zprávu). Pokud je klíč vybrán náhodně, pak bude muset útočník v průměru vyzkoušet polovinu všech možných klíčů předtím, než najde hledaný dešifrovací klíč.[11]




**Obrázek 14 – Princip brute force[11]**

| **Znaky hesla (písmena a číslice)** | 0123456789AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz | | | |
|---|---|---|---|---|
| **Heslo** | | **Třída útoku** | | |
| **Délka** | **Kombinací** | A | B | C |
| 2 | 3,844 | Okamžitě | Okamžitě | Okamžitě |
| 3 | 238,328 | Okamžitě | Okamžitě | Okamžitě |
| 4 | 15 miliónů | < 2 sec | Okamžitě | Okamžitě |
| 5 | 916 miliónů | 1½ min | 9 sec | Okamžitě |
| 6 | 57 biliónů | 1½ hodin | 9½ min | 56 sec |
| 7 | 3.5 triliónů | 4 dnů | 10 hodin | 58 min |
| 8 | 218 triliónů | 253 dnů | 25¼ dnů | 60½ hodin |

Třídy:
A. 10,000,000 hesel /sec – výkonné PC
B. 100,000,000 hesel /sec – více spolupracujících PC
C. 1,000,000,000 hesel /sec – superpočítač

**Tabulka 1 – Prolomení hesla hrubou silou[12]**

Různé šifrovací algoritmy nejsou stejně kvalitní, některé systémy nejsou příliš dobré při ochraně dat, což umožňuje, aby šifrované informace byly dešifrovány bez znalosti potřebného klíče, zato jiné jsou poměrně odolné i nejodhodlanějším útokům. Schopnost

kryptografického systému chránit informace před útokem se nazývá „Strenght“ (síla). Síla systému závisí na mnoha faktorech:[11]

- Utajení klíče.
- Obtížnost hádání klíčů nebo vyzkoušení všech možných klíčů (Bruteforce). Dlouhé klíče je obecně obtížnější odhadnout nebo najít.
- Obtížnost dešifrování šifrovacího algoritmu bez znalosti šifrovacího klíče.
- Existence děr v algoritmu (back door) nebo další způsoby, jak lze šifrovaný soubor snadno dešifrovat bez znalosti klíče.
- Schopnost dešifrovat celou šifrovanou zprávu, pokud je známo, jak dešifrovat část - zprávy.
- Vlastnost textu a znalost těchto vlastností útočníkem (například kryptografický systém může být zranitelný vůči útoku, pokud všechny zprávy jím šifrované začínají nebo končí známou částí textu).

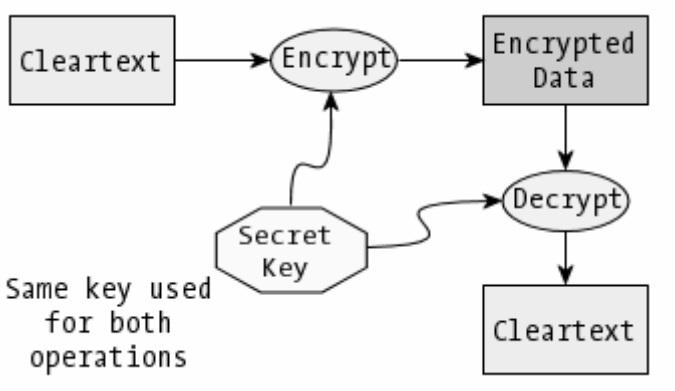
Obecně není síla kryptografického systému prokazována, může být jen vyvrácena. Když je navržen nový šifrovací algoritmus, autor algoritmu je téměř vždy přesvědčen, že algoritmus nabízí dokonalou bezpečnost, tedy že autor se domnívá, že neexistuje možnost dešifrování šifrované zprávy bez vlastnictví příslušného klíče.[11]

Pro prokázání síly algoritmu, matematik může ukázat odolnost algoritmu vůči specifickým druhům útoků, které byly předtím pro prolomení jiných algoritmů. Bohužel ani algoritmus, který je odolný vůči všem známým útokům, není nutně bezpečný, protože nové způsoby útoků se neustále vyvíjejí.[11]

### 3.2.1 Symetrická kryptografie

Na symetrické kryptografii jsou založeny všechny historické kryptosystémy, počínaje antikou, středověkem a jsou používané i v současnosti. Principem symetrické kryptografie je použití jednoho klíče jak pro zašifrování, tak i pro dešifrování (viz. Obrázek 15).

Algoritmy symetrického šifrování se používají především pro šifrování velkého počtu dat nebo datových toků a tyto algoritmy jsou navrženy tak, aby byly velmi rychlé. Nejlepší symetrické šifrovací algoritmy nabízejí vynikající zabezpečení, tedy jakmile jsou data šifrována daným klíčem, neexistuje (není dosud znám) žádný rychlý způsob, jak dešifrovat data bez znalosti šifrovacího klíče. Algoritmy symetrických šifer lze rozdělit do dvou kategorií: blokové a streamové (proudové). Blokové algoritmy šifrují data v bloku (mnoho bajtů) najednou, zatímco algoritmy pro streamování šifrují bajt bajtem (nebo bit po bitu).[11]



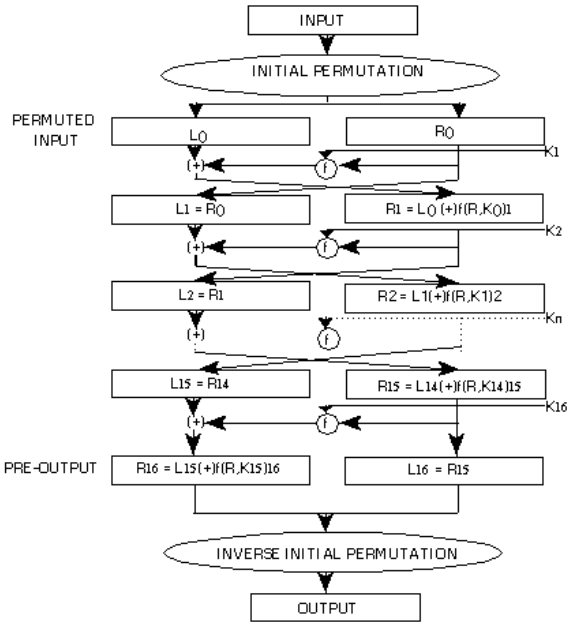

**Obrázek 15 – Symetrické šifrování[10]**

### DES

Data Encription Standart (DES) byl vyvinut v roce 1974 společností IBM a americkou vládou, kteří stanovili standard, kterým by každý mohl bezpečně komunikovat. DES operuje na bloku 64 bitů pomocí tajného klíče, který je dlouhý 56 bitů. Původní návrh používal tajný klíč o délce 64 bitů. Obecně se předpokládá, že odstranění těchto 8 bitů z klíče bylo provedeno, aby americké vládní agentury mohly tajně dešifrovat zprávy.[16]

Šifrování bloku zprávy se uskutečňuje v 16 etapách (cyklech). Z vstupního klíče se generuje šestnáct 48 bitových klíčů, jeden pro každou etapu. V každém cyklu se používá osm takzvaných S-boxů. Tyto S-boxy jsou pevně stanoveny ve specifikaci standardu. Pomocí S-boxů jsou skupiny šesti bitů mapovány do skupin čtyř bitů. Obsah těchto S-boxů byl určen Národní bezpečnostní agenturou USA (NSA). S-box se zdá být náhodně vyplněn, ale není tomu tak. Nedávno bylo zjištěno, že tyto S-boxy, definované v sedmdesátých letech, jsou odolné proti útoku diferenciální kryptanalýzy. Blok zprávy je rozdělen na dvě poloviny. Pravá polovina se rozšiřuje z 32 na 48 bitů pomocí pevné tabulky. Výsledkem je kombinace s podklíčem pro daný cyklus pomocí operace XOR. Pomocí S-boxů se 48 výsledných bitů znovu transformuje na 32 bitů, které se následně znovu promítají pomocí další tabulky. Tyto části se následně důkladně promíchají, pravá polovin je zkombinována s levou polovinou pomocí operace XOR.[16]

V době kdy byl standart uveden, se věřilo, že vyzkoušení všech 72,057,594,037,927,936 možných kombinací klíče ($7\times10^{16}$) by bylo nemožné, díky slabšímu výkonu počítačů té doby, avšak v roce 1998 vytvořila Electronic Frontier Foundation (EFF) speciální stroj, který by mohl dešifrovat zprávu vyzkoušením všech možných klíčů za méně než tři dny. Stroj stojí méně než 250 000 dolarů a hledá přes 88 miliard klíčů za sekundu.[16]




**Obrázek 16 – DES[16]**

### Triple-DES

Varianta Triple-DES byla vyvinuta poté, co se ukázalo, že DES již není tolik odolní proti prolomení hrubou silo. Používá tři 56bitové klíče DES, které dávají celkovou délku klíče 168 bitů. Šifrování pomocí systému Triple-DES je jednoduché, protože Triple-DES používá algoritmus DES třikrát za sebou a tedy i trvá třikrát delší než standardní DES.[16]

### AES (Rijndael)

Bloková šifra Rijndael je navržena tak, aby používala pouze jednoduché celo-bajtové operace. Poskytuje také větší flexibilitu, než požaduje kandidát AES, protože jak velikost klíče, tak velikost bloku mohou být vybrány jako 128, 192 nebo 256 bitové varianty.[17]

Rijndael má různý počet cyklů:[17]

- 9, jestliže blok i klíč mají délku 128 bitů.
- 11, jestliže buď blok, nebo klíč má délku 192 bitů a druhý není delší než 192 bitů.
- 13, pokud je blok nebo klíč dlouhý 256 bitů.

„Tato šifra je postavena na principu použití substituce a permutace k převedení bloků otevřeného textu do textu šifrového a naopak. U šifry Rijndael dochází v každém opakování k zašifrování všech 128 bitů oproti DES, ve které se promění pouze polovina bitů (tudíž 32), což ji činí mnohem efektivnější. AES se v zásadě skládá z takzvaných vrstev, přes které prochází bloky bitů z plaintextu a převádějí se do textu šifrovaného. Co se matematických principů týče, jedná se o aplikaci konečných těles, tzn. těles s konečným počtem prvků.

#### *Key Addition layer*

Zde dochází k aplikaci šifrovacího klíče a procesům, jako je XOR.

#### *Byte Substitution layer (S-Box)*

Každý prvek je transformován za pomoci tabulek zvaných S-Box. Je tím zaručeno, že se změny jednotlivých bitů projeví i na všech ostatních.

#### *Diffusion layer*

Tato vrstva se skládá z dvou částí, jež obě provádějí lineární operace. Dochází zde k permutaci dat.

#### *Dešifrování*

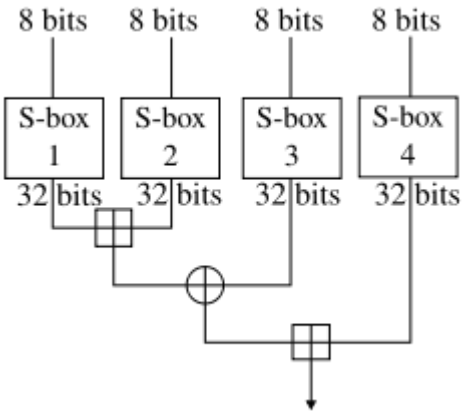
Dešifrování není sice shodné se šifrováním a probíhá trochu jinak, ale využívá ty samé myšlenky a principy.“[18]

### Blowfish

Blowfish je bloková šifra navržená Brucem Schneierem v roce 1993. Zabezpečení bylo značně otestováno a osvědčeno. Jako šifra pro veřejnou doménu byla Blowfish předmětem značného množství kryptoanalýzy a úplné šifrování Blowfish nebylo nikdy prolomeno. Blowfish je také jedním z nejrychlejších blokových šifer pro veřejné použití.[19]

Blowfish má 64bitovou velikost bloků a délku klíče od 32 bitů do 448 bitů. Jedná se o šifru s 16 cykly a používá velké klíče závislé na S-boxu. Je strukturován podobně jako CAST-128, který používá fixní S-boxy.[19]

Blowfish nepodléhá žádným patentům a je proto volně k dispozici pro kohokoli k použití. To přispělo k jeho popularitě v kryptografickém softwaru.[19]




**Obrázek 17 – Feistel struktura[19]**

### IDEA

IDEA (International Data Encryption Algorithm) je nejvíce známý jako blokový šifrovací algoritmus používaný v populárním šifrovacím programu PGP. Algoritmus IDEA je zajímavý sám o sobě. Zahrnuje některé kroky, ve kterých se zdá, že místo blokové šifry se jedná o jednosměrnou hashovací funkci, také nepoužívá vyhledávacích tabulek nebo S-boxů. IDEA používá 52 podklíčů, každý 16 bitů dlouhý. Dva se používají během každého cyklu a čtyři se používají před každým cyklem a po posledním cyklu.[20]

Blok prostého textu v IDEA je rozdělen na čtyři čtvrtiny, každý 16 bitů dlouhý. Tři operace jsou používány v IDEA pro zkombinování dvou 16bitových hodnot pro vytvoření 16bitového výsledku (doplněk, XOR a násobení).[20]

### MARS

Jeden z kandidátů pro Advanced Encryption Standard (AES) byl i MARS, algoritmus vyvinutý společností IBM. Jedná se o symetrickou blokovou šifru klíčů, která používá 128bitové bloky a podporuje variabilní velikosti klíče (od 128 do 1248 bitů). MARS je jedinečný v tom, že v jednom algoritmu kombinuje prakticky všechny konstrukční techniky známé kryptografům. Používá sčítání a odčítání, S-krabice, pevné a datové rotace a násobení.[21]

MARS je šifra, která pracuje s velikostí bloku 128 bitů a proměnnou velikostí klíče. Algoritmus je síť typu Feistel, která je orientována na slovo (32 bitů). Orientace slov by měla přinést výkonnost softwarových implementací na většině počítačových architektur, které jsou dnes k dispozici. Očekává se, že plně optimalizovaná implementace běží rychlostí 100 Mbit/s a hardware může dosáhnout dalšího 10 násobného zrychlení.[21]

### Serpent

Serpent je 128bitová bloková šifra, kterou navrhli Ross Anderson, Eli Biham a Lars Knudsen jako kandidáta pro Advanced Encryption Standard. Byl to finalista v soutěži AES. Serpent a Rijndael jsou si podobné, hlavní rozdíl je v tom, že Rijndael je rychlejší (má méně cyklů), ale Serpent je zase bezpečnější.[22]

Serpent byl navržen tak, aby poskytl nejvyšší praktickou úroveň bezpečí, aby toho mohlo být dosaženo, vývojáři se omezili na dobře pochopené mechanismy, které jsou již ověřené praxí. Také bylo použito dvakrát tolik cyklů než u AES, které jsou dostačující k zakódování všech aktuálně známých zkratek.[22]

### Twofish

Twofish je bloková šifra, má velikost bloku 128 bitů a přijímá klíč libovolné délky až 256 bitů. Twofish je rychlý na 32bitových i 8bitových CPU, lze jej používat v síťových aplikacích, kde se klíče často mění, a v aplikacích, kde je k dispozici malá nebo žádná paměť RAM a ROM. Twofish je síťí Feistel, to znamená, že v každém kole je polovina

textového bloku odeslána prostřednictvím funkce F a poté XORována s druhou polovinou textového bloku.[23]

#### RC2

Tato bloková šifra byla původně vyvinutá společností Ronald Rivest a vedena jako obchodní tajemství společnosti RSA Data Security. Algoritmus byl odhalen anonymním zasíláním Usenetu v roce 1996 a zdá se být přiměřeně silný (ačkoli existují určité klíče, které jsou slabé). RC2 umožňuje klíče od 1 do 2048 bitů. Délka klíče RC2 byla tradičně omezena na 40 bitů v softwaru, který byl exportován do zahraničí, aby umožnil dešifrování národní bezpečnostní agenturou USA.[11]

#### RC4

Tato streamová šifra byla původně vyvinutá společností Ronald Rivest a vedena jako obchodní tajemství společnosti RSA Data Security. Tento algoritmus byl také odhalen anonymním vysíláním Usenet v roce 1994 a zdá se být přiměřeně silný. RC4 umožňuje klíče od 1 do 2048 bitů. Délka klíče RC4 byla tradičně omezena na 40 bitů v softwaru, který byl exportován do zahraničí.[11]

#### RC5

Šifrovací algoritmus RC5 je rychlá, symetrická bloková šifra vhodná pro hardwarové nebo softwarové implementace. Novinkou RC5 je velké využití datově závislé rotace. RC5 má tajný klíč s proměnnou délkou a poskytuje flexibilitu ve své bezpečnostní úrovni. RC5 není určen pro zajištění všech možných hodnot parametrů. Na druhou stranu, volba maximálních hodnot parametrů by byla pro většinu aplikací přehnaná.[24]

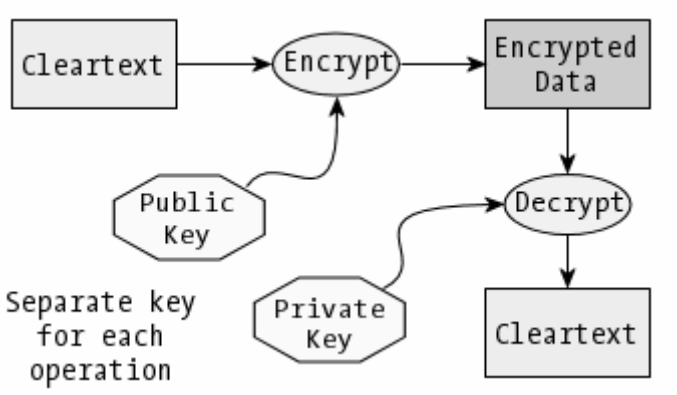
#### RC6

Bloková šifru RC6 navrhl Ron Rivest ve spolupráci s Mattem Robshawem, Raym Sidneyem a Yiqun Lisa Yin z RSA Laboratories. Tento algoritmus je evoluční vylepšení oproti blokové šifře RC5 a stejně jako RC5 využívá datových závislých rotací. V rámci úsilí nalézt nástupce DES, společnost RSA Laboratories předložila RC6 jako kandidáta na Advanced Encryption Standard (AES). RC6 zůstává ideální volbou pro mnoho vysokorychlostních a vysoce výkonných aplikací.[25]

### 3.2.2 Asymetrická kryptografie

Existence kryptografie s využitím veřejných klíčů (asymetrická kryptografie) byla poprvé představena na podzim roku 1975 Whitfieldem Diffiem a Martinem Hellmanem. Dva badatelé, napsali článek, v němž předpokládali existenci šifrovací techniky, ve které by mohla být informace šifrovaná jedním klíčem (veřejný klíč) a dešifrována druhým, zdánlivě nesouvisejícím klíčem (soukromý klíč). Robert Merkle, tehdy student v Berkeley, měl podobné myšlenky současně, ale kvůli rozmarům akademického publikačního procesu nebyly dokumenty publikovány, dokud nebyly všeobecně známy základní principy a matematika algoritmu Diffie-Hellman.[11]

Od té doby byly vyvinuty různé šifrovací systémy s využitím veřejného klíče. Bohužel se v algoritmech s veřejnými klíči vyskytl výrazně menší vývoj než u algoritmů symetrických klíčů, důvod se může označit způsob, jakým jsou tyto algoritmy tvořeny. Dobré algoritmy symetrických klíčů jednoduše zakódují svůj vstup v závislosti na vstupním klíči a vývoj nového algoritmu symetrického klíče vyžaduje nové způsoby, jak spolehlivě provádět tuto operaci. Algoritmy veřejných klíčů mají tendenci být založeny na teorii čísel. Vývoj nových algoritmů veřejného klíče vyžaduje identifikaci nových matematických rovnic se zvláštními vlastnostmi.[11]




**Obrázek 18 – Asymetrické šifrování[10]**

### Diffie-Hellman

Diffie-Hellman je systém (princip) pro výměnu kryptografických klíčů mezi komunikujícími stranami, takže vlastně není metoda šifrování a dešifrování, ale metoda pro vývoj a výměnu sdíleného soukromého klíče přes veřejný komunikační kanál. Ve skutečnosti obě strany souhlasí s některými společnými číselnými hodnotami a pak každá strana vytvoří klíč. Matematické transformace klíčů se vyměňují. Každá strana pak může vypočítat třetí klíč relace, který nelze snadno odvodit útočníkem, který zná obě vyměněné hodnoty.[11]

### *DSA/DSS*

Standard digitálního podpisu (DSS) byl vyvinut národní bezpečnostní agenturou USA a přijatý jako Federální standard pro zpracování informací (FIPS) od Národního institutu pro standardy a technologie. DSS je založen na algoritmu Digital Signature Algorithm (DSA). Ačkoli DSA umožňuje klíče libovolné délky, jsou povoleny pouze klíče od 512 do 1 024 bitů pod FIPS. Jak je uvedeno, DSS lze použít pouze pro digitální podpisy, ačkoli je možné použít i některé implementace DSA pro šifrování.[11]

### *RSA*

RSA je kryptografický systém veřejného klíče vyvinutý v roce 1977 třemi profesory na MIT (Ronald Rivest, Adi Shamir a Leonard Adleman). RSA lze použít jak pro šifrování informací, tak jako základ systému digitálního podpisu. Digitální podpisy mohou být

použity k prokázání autorství a autenticity digitálních informací. Klíč může mít libovolnou délku, v závislosti na použité implementaci.[11]

#### Eliptické křivky

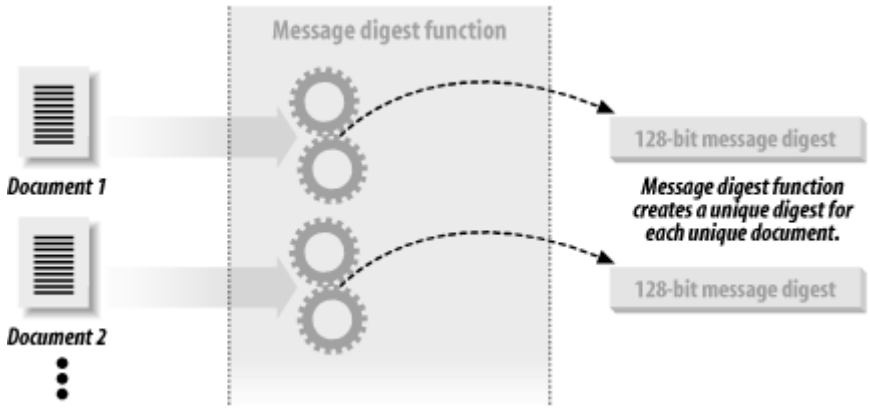
Systémy veřejných klíčů jsou tradičně založeny na faktoringu (RSA), diskrétních logaritmech (Diffie-Helman) a na problematice zvané „knapsack“. Kryptosystémy eliptických křivek jsou šifrovací systémy s veřejným klíčem, které jsou založeny spíše na eliptické křivce než na tradiční logaritmické funkci, tedy jsou založeny na řešeních rovnice $y^2=x^3+ax+b$. Výhoda použití eliptických křivkových systémů vyplývá ze skutečnosti, že neexistují žádné známé subexponenciální algoritmy pro výpočet diskrétních logaritmů eliptických křivek. Krátké klávesy v kryptosystému eliptických křivek tak mohou nabídnout vysoký stupeň ochrany soukromí a bezpečnosti, přičemž zůstávají snadno vypočitatelné. Eliptické křivky lze vypočítat velmi efektivně v hardwaru.[11]

### 3.2.3 Message Digest Functions

Message digest function se také nazývají jednosměrnými hashovacími funkcemi nebo kryptografickými hashovacími funkcemi, protože vytvářejí hodnoty obtížně invertovatelné, odolné vůči útoku, efektivně jedinečné a široce distribuované, čehož se využívá v kryptografii.[11]

Message digest algoritmy shlukují informace obsažené v souboru (malé nebo velké) do jediného velkého čísla, typicky mezi 128 a 256 bitů. Nejlepší algoritmy kombinují tyto matematické vlastnosti:[11]

- Každý bit výstupu funkce (algoritmu) je potenciálně ovlivněn každým bitovým vstupem funkce.
- Pokud se změní některý bit ve vstupu funkce, každý výstupní bit má 50% šanci na změnu.
- Při znalosti vstupního souboru a jeho příslušnému vygenerovanému hash číslu by mělo být výpočetně nemožné najít jiný soubor se stejnou hodnotou hash.




**Obrázek 19 – Message Digest Function[11]**

### *MD2*

Message Digest # 2 vyvinul Ronald Rivesth. Tato hashovaní funkce je pravděpodobně nejbezpečnější z celé rodiny funkcí služby Rivest, ale trvá nejdéle k výpočtu, proto je MD2 zřídka používán. MD2 vytváří 128bitový záznam.[11]

### *MD4*

Message Digest # 4, také vyvinutý Ronaldem Rivestem, byl vyvinut jako rychlá alternativa k MD2. Následně bylo prokázáno, že MD4 má možnou slabinu. Může být možné nalézt druhý soubor, který produkuje stejný soubor MD4 jako daný soubor, aniž by vyžadoval vyhledávání hrubou silou (což by bylo nemožné ze stejného důvodu, proč není možné prohledávat 128bitový klíčový prostor). MD4 produkuje 128bitový záznam.[11]

### *MD5*

Message Digest # 5, také vyvinutý Ronaldem Rivesthem, je modifikací modulu MD4, který zahrnuje techniky navržené tak, aby byly bezpečnější. Ačkoli MD5 je široce používán, v létě 1996 bylo v MD5 zjištěno několik nedostatků, které umožnily výpočet některých druhů kolizí v oslabené podobě algoritmu. Výsledkem je, že MD5 pomalu upadá. MD5 a SHA-1 se používají v SSL i v technologii Microsoft Authenticode. MD5 produkuje 128bitový hash.[11]

### *SHA-0 a SHA-1*

Secure Hash Algoritmus, který se vztahuje k MD4 a je určen pro použití se standardem digitálního podpisu (NIST's National Standard Institute) pro standardy a technologie (NIST's DSS). Krátce po zveřejnění SHA společnost NIST oznámila, že SHA není vhodné pro použití, pokud se neprovedou malé změny. SHA produkuje 160bitový hash. Revidovaný algoritmus (SHA-1) zahrnuje drobné změny, není veřejně známo, zda tyto změny činí SHA-1 bezpečnější než SHA, i když mnoho lidí se domnívá, že jo. SHA-1 produkuje 160bitový digest.[11]

### SHA-2

Jedná se o rodinu 224, 256, 384 a 512bitových hashovacích funkcí určených pro použití se 128, 192 a 256bitovými šifrovacími algoritmy. Tyto funkce byly navrženy NIST v roce 2001 pro použití s Advanced Encryption Standard.[11]

### SHA-3

Keccakův algoritmus je dílem Guida Bertoniho, Joana Daemena, Michaela Petersa a Gilesa Van Assche a byl předložen jako kandidát SHA-3 v říjnu 2008. Keccak využívá inovativní „sponge engine“ ke ztenčení textu zprávy. Je rychlý, s průměrnou rychlostí 12,5 cyklů na bajt na procesoru Intel Core 2. Jeho jednoduchý design je vhodný pro implementaci v hardwaru. Keccak odolává známým útokům s minimální složitostí 2n, kde

n je velikost hashe. Má značnou bezpečnostní rezervu. Dosavadní kryptoanalýza třetích stran neukázala v Keccaku žádné vážné slabiny.[26]

## 3.3 Kryptoanalýza

### Symetrické šifrování

Pokud by délka klíče byla jediným faktorem určujícím bezpečnost šifry, všichni zájemci o výměnu tajných zpráv by jednoduše používali kódy s 128bitovými klíči a všichni kryptoanalytici by si museli najít nová pracovní místa. Kryptografie by byla vyřešená větev matematiky.[11]

To, co kryptografie dělá zajímavou, je skutečnost, že většina šifrovacích algoritmů nesplňuje naše očekávání. Klíčové vyhledávací útoky jsou zřídka vyžadovány k odhalení obsahu šifrované zprávy. Místo toho může být většina šifrovacích algoritmů potlačena použitím kombinace sofistikované matematiky a výpočetní síly. Výsledkem je, že mnoho šifrovaných zpráv může být dešifrováno bez znalosti klíčů. Šikovný kryptoanalyzátor může někdy dešifrovat šifrovaný text bez znalosti šifrovacího algoritmu.[11]

Kryptoanalytický útok může mít dva možné cíle. Kryptoanalyzátor může mít šifrovací text a chtít získat původní text, nebo může mít šifrovací text a chce zjistit šifrovací klíč, který byl použit k šifrování (tyto cíle jsou podobné, ale ne úplně stejné). Následující útoky se běžně používají při znalosti šifrovacího algoritmu a mohou být použity pro šifrované soubory nebo internetový provoz:[11]

#### *Known plaintext útok*

V tomto typu útoku má kryptoanalyzátor blok nešifrovaného textu a odpovídající blok šifrového textu. I když se tato kombinace může zdát nepravděpodobná, je ve skutečnosti docela běžná, když se kryptografie používá k ochraně elektronické pošty (se standardními záhlavími na začátku každé zprávy), standardních formulářů nebo pevných disků (se známými strukturami na předem určených místech disku). Cílem útoku je určit kryptografický klíč (a případně algoritmus), který lze pak použít k dešifrování dalších zpráv.[11]

#### *Chosen plaintext útok*

V tomto typu útoku má kryptoanalyzátor (nevědomky) šifrovat zvolené bloky dat, což vytváří výsledek, který kryptoanalytik pak může analyzovat. Zvolené útoky jsou jednodušší, než by se mohly zdát (například předmětem útoku může být rádiový odkaz, který zašifruje a přenáší zprávy přijaté telefonicky). Cílem zvoleného útoku je zjistit kryptografický klíč, který lze pak použít k dešifrování dalších zpráv.[11]

#### *Diferenciální kryptoanalýza*

Tento útok, který je formou chosen plaintext útoku, zahrnuje šifrování mnoha textů, které se vzájemně mírně liší a porovnávají se výsledky.[11]

### *Analýza diferenciální poruchy*

Tento útok funguje proti kryptografickým systémům, které jsou postaveny závislé na hardware. Přístroj je vystaven vlivům okolního prostředí (teplo, stres, záření), které jsou navrženy tak, aby zařízení mohly způsobit chyby během šifrování nebo dešifrování. Tyto chyby mohou být analyzovány a od nich je možné naučit se interní stav zařízení, včetně šifrovacího klíče nebo algoritmu.[11]

### *Analýza diferenciálního výkonu*

Jedná se o další útok proti kryptografickému hardwaru, zejména chytrým kartám. Pozorováním síly, kterou inteligentní karta používá k šifrování zvoleného bloku dat, je možné se naučit trochu informací o struktuře tajného klíče. Tím, že je čipová karta podrobena řadě speciálně zvolených datových bloků a pečlivě monitoruje použitý výkon, je možné určit tajný klíč.[11]

### *Analýza diferenciálního časování*

Tento útok je podobný analýze diferenciálního výkonu, útočník pečlivě monitoruje dobu, kterou chytré kartě trvá, než provede požadované operace šifrování.[11]

### Asymetrické šifrování

Algoritmy veřejného klíče jsou teoreticky náchylnější k útoku než algoritmy symetrických klíčů, protože útočník (pravděpodobně) má kopii veřejného klíče, který byl použit k šifrování zprávy. Úloha útočníka je dále zjednodušena, protože zpráva pravděpodobně identifikuje, který šifrovací algoritmus veřejného klíče byl použit k šifrování zprávy.[11]

Klíčové vyhledávací útoky jsou nejpopulárnějším typem útoků, protože jsou nejlépe pochopitelné. Tyto útoky se pokusí odvodit soukromý klíč z odpovídajícího veřejného klíče.[11]

Druhým způsobem útoku na šifrovací systém veřejného klíče je nalezení chyby nebo slabosti matematického problému, na němž je šifrovací systém založen. První systém šifrování veřejného klíče, byl založen na matematickém problému nazývaném Superincreasing Knapsack Problem. Několik let poté, co byla tato technika navržena, byla nalezena cesta, jak matematicky odvodit tajný klíč z veřejného klíče ve velmi krátkém čase.[11]

Stojí také za zmínku, že je možné, je-li objeven velký matematický průlom ve faktoringu, nemusí být zveřejněn. Například, pokud je nová metoda vyvinuta vládní agenturou, může být uchovávána tajně, aby mohla být použita proti šifrovaným zprávám zaslaným úředníky z jiných zemí. Stejně tak, pokud je nová metoda vyvinuta někým s kriminálními tendencemi, může být uchovávána tajně, aby ji bylo možno použít v budoucích ekonomických zločinech zahrnujících stávající šifrovací metody.[11]

# 4. Zdroje entropie

Text této kapitoly byl převzat z mé vlastní bakalářské práce[1] a přepracován a rozšířen v těch částech, které byly implementovány do praktické části.

Jako zdroj entropie se dá prakticky použít jakýkoliv dostupný zdroj (zařízení či program), který generuje výstup, po otestování tohoto výstupu statistickými testy (např. zmíněnými v kapitole č. 2) se dá určit, jestli je vhodný či nevhodný, což bylo provedeno v předchozí práci. Ovšem je nutné se i zamyslet (a pokud je možno otestovat), zda zdroj fungující na jednom stoji bude fungovat stejně dobře na jiném stroji.

Běžně používanými zdroji entropie jsou počítačová myš, klávesnice, čas IDE (Integrated Development Environment), ale existují i jiné potenciální zdroje. Například by bylo možné vytvářet entropii z mikrofonu počítače nebo postavením snímače pro měření turbulence vzduchu uvnitř diskové jednotky, nicméně mikrofony nejsou obvykle přítomny na serverech. Pro odvozeniny z Unix/BSD existují USB řešení, která používají ARM Cortex CPU pro filtrování a zajištění bitového toku generovaného dvěma zdroji entropie v systému.

## 4.1 Vhodné zdroje

Jako vhodné zdroje jsou uvedeny příklady entropie, které byly zjištěny a otestovány v předchozí bakalářské práci.

### Souřadnice pozice kurzoru

Dnešní rozlišení obrazovky počítačů, které jsou již v jednotkách tisíců obrazových bodů, dovolují použít z každé x a y souřadnice jednotky a desítky, díky čemuž dostaneme dohromady 4 čísla v desítkové soustavě, tedy 13 bitů ($log_2(10 \times 10 \times 10 \times 10)$) entropie. Tento zdroj je použitelný jen na strojích, kde je zaručena aktivita uživatele, tedy tuto entropie generuje uživatel. Tato metoda je i poměrně snadno implementovatelná, jelikož snad každý programovací jazyk dokáže snímat pozici kurzoru a jak bylo v předchozí bakalářské práci změřeno a otestováno, je tato entropie i poměrně kvalitní.

### Využití CPU

Zatížení procesoru se může zdát ne moc kolísavé, tedy i ne moc náhodné, bereme-li v potaz celé jednotky procenta využití, avšak jednotky za desetinou čárkou již kolísají bez zjevné odhadnutelnosti, tedy každé číslo, které získáme za desetinou čárkou, se dá využít jako zdroj entropie. Co se týče implementace, většina programovacích jazyků nedovoluje měřit využití procesoru, tedy je nutné si měření naprogramovat buď sami, nebo použít již existující externí knihovny, které to umožní. Samotné využití procesoru je definováno jako podíl časů jednotlivých stavů procesoru, v daném intervalu, podle rovnice:

$$\frac{User + Kernel}{User + Kernel + Idle}$$

kde: User – čas strávený prováděním uživatelských procesů
Kernel – čas strávený voláním jádra, přerušením nebo chybami
Idle – čas strávený ve stavu nečinnosti

### Využití fyzické paměti PC

Využití fyzické paměti, stejně jako využití CPU, je využitelné pro získávání entropie pro jednotky procent za desetinou čárkou, avšak je také nutné zaručit, že se zatížení fyzické paměti bude měnit, jelikož jakmile si programy alokují, co potřebují, hodnota se nebude měnit. Co se týče implementace, stejně jako u využití CPU, programovací jazyky nedovolují měřit celkové využití fyzické paměti, tedy je nutné si měření naprogramovat, nebo použít externí knihovny, které to umožňují.

### Počet stisků kláves v daném intervalu

Pro využití těchto hodnot by bylo nutné shromažďovat data dlouhodobě za podmínky, že je klávesnice aktivně využívána, využívat by se daly desítky a jednotky, z hodnot podle aktivity uživatele, v intervalu například jedné minuty, dostáváme tedy 6 bitů ($log_2(10 \times 10)$) entropie. Implementace této metody získávání entropie je poměrně snadná, jelikož programovací jazyky umožňují skrz „handlery“ rozpoznávat stisk kláves.

### Teplota na čidlech CPU, GPU, HDD

Využití čidel teploty na procesoru, grafické kartě či pevném disku lze opět, stejně jako pro využití CPU, získat entropii, pokud dokážeme číst hodnoty alespoň za desetinou čárkou, jelikož celé stupně Celsia nebudou náhodné. Samotná implementace je poměrně složitá, programovací jazyky neumožňují číst hodnoty z těchto čidel přímo, proto je nutné si naprogramovat ovladač, který to umožní.

### Aktuální lokální čas

Velice často používané, avšak musí být použito správně. Dnešní systémy měří čas s přesností na stovky nanosekund, takže dostáváme spolehlivých 20 bitů entropie, pokud aplikujeme dostatečný odstup za desetinou čárkou od celých hodnot sekund. Implementace je velice snadná, většina programovacích jazyků umožňuje získání hodnoty lokálního času OS, avšak tato metoda je vhodná pouze pro jednorázové hodnoty, ne pro sbírání entropie průběžně.

### Počet aktivních procesů

Jednorázové použití celkového počtu procesů v jednotkách je na uživatelské úrovni poměrně různorodé, avšak nevýhoda je, že se dá použít jen jednorázově, opakované hodnoty budou pravděpodobně stejné v malém časovém intervalu. Na jednorázové použití

dostáváme tedy 3 bity entropie ($log_2(10)$). Na serverech ale běží pravděpodobně stejný počet procesů vždy, zde tedy nelze využít tuto metodu.

### Počet položek v adresáři

Stejně jako počet aktivních procesů, lze počet položek v adresáři použít jen jednorázově, ale zato lze použít několik adresářů. Pro každý adresář tedy dostáváme 3 bity entropie ($log_2(10)$).

### Počet zařízení v síti

Počet zařízení v síti se mnoho nemění, proto lze efektivně opět použít jen jednorázově hodnotu počtu, avšak na velkých veřejných sítí se počet v jednotkách bude pravděpodobně měnit více v pracovní době. Dostáváme tedy znova 3 bity entropie ($log_2(10)$) v ideálním případě.

### Jméno počítače

Použití také jednorázové, avšak výhodou je, že každá z těchto hodnot je na většině PC jiná a pro prolomení je potřeba znalosti PC (předpokládejme tedy, že útočník nezná jméno PC), z kterého bylo heslo generováno. Jméno uživatelského PC je většinou jméno uživatele s příponou „PC“ na Windows, avšak jméno serveru je vždy jiné díky FQDN, které je unikátní a jsme si jistí, že žádný PC nedostane stejnou hodnotu a počet použitelných bitů dostáváme podle jeho délky.

### Výstupy a vstupy zvukové karty

Při puštěném mikrofonu vzniká analogová aktivita doplněná šumem okolí, i když je ticho, jež lze převést do digitální podoby a čerpat z těchto hodnot entropii, při vypnutém mikrofonu vzniká zase teplotní šum na vstupech karty a i tu lze číst. Rychlost generování závisí na nastavení mixeru. Z těchto hodnot lze převést velkou část jako entropii, kterou lze generovat poměrně rychle v krátkých intervalech. Velikost entropie závisí na konkrétních parametrech zvukové karty, ale odhadujme přibližně 16 bitů minimálně.

### Síťová aktivita

V síti existuje mnoho nezopakovatelné nepředvídatelnosti, měřit by se mohl počet paketů, délka spojení, délka v B paketu nebo čas doručení paketu, dokonce router generuje provoz, o jehož náhodnost se postará sama síť, počet paketů se může vyskytovat v milionech kusů. Pro implementaci je často zapotřebí použití externích knihoven, které dokážou odchytávat pakety a následně je číst.

### Entropie obrázků či videa

Entropii z obrázků či videí lze získat z jejich barevné složky jednotlivých pixelů. Počet využitelných bitů je různý, závisí na velikosti obrázku a jeho kvality, kterou určuje software, to samé platí u videí. Za barevnou složku pozice kurzoru dostaneme 19 bitů

entropie v ideálním případě, ovšem je nutné vzít v potaz, že sousední body budou mít pravděpodobně stejné nebo skoro stejné hodnoty, tudíž je nutné určit hranici, kdy jsou hodnoty už rozdílné.

#### Počet čtení z disku a zápisů na disk

Na disku, kde je umístěn operační systém, je generována aktivita jednotlivých čtení a zápisů na disk poměrně častá, proto při vhodné implementaci a zajištění aktivity je možné tyto hodnoty použít, pro použití se vyplatí jednotky, tedy dostaneme dvě čísla a 6 bitů entropie.

## 4.2 Nevhodné zdroje

Nevhodné zdroje jsou ty, které postrádají nepředvídatelnost, jsou konstantní nebo snadno odhalitelné a tudíž nevhodné pro kryptografické účely.

#### Délka běhu programu

Využití je nevhodné, jelikož se k danému kódu dostaneme ve stejně podobný čas (periodu) na jednom stejně zatíženém stroji, dostaneme tedy velice nízkou nepředvídatelnost. Avšak potencionál při vhodném použití zde je, stejně jako u aktuálního lokálního času.

#### get_OS_MXBean v javě

Metoda getCommittedVirtualMemorySize generuje na jednotkách a desítkách rozdílné hodnoty, tedy by využitelný byl, stejně tak jako metoda getFreePhysicalMemorySize a metoda getFreeSwapSpaceSize, avšak nelze se na ně 100% spoléhat.

#### Uživatelské jméno v kódování ASCII

Snadné a jednoduché zjistit, tedy nevhodné pro entropii ač uživatelské stroje mývají jména svých uživatelů, ale tento počet jmen není nekonečný a ne moc náhodný.

#### Velikost HDD, RAM (velikost registrů CPU, GPU)

Dostupná kapacita je předvídatelná, disky a paměti mají své parametry veřejně dostupné v katalozích.

#### ID aktuálního procesu

ID procesu z principu není náhodné číslo, jelikož jeho ID určuje systém konkrétním a předvídatelným generátorem. Při troše snahy je možné určit ID budoucího procesu z ID existujícího procesu (např. Linux ID procesů nerozhazuje ani náhodně, ale příbuzné procesy shlukuje k sobě).

#### Počet CPU

Jednorázové číslo, smysluplné pro použití pouze na náhodných strojích, které disponuje špatnou entropií a jeho váhou.

### ID portu nesystémové aplikace

V drtivé většině předvídatelná hodnota, ač by být neměla, ale ID jsou generovány inkrementací čísla.

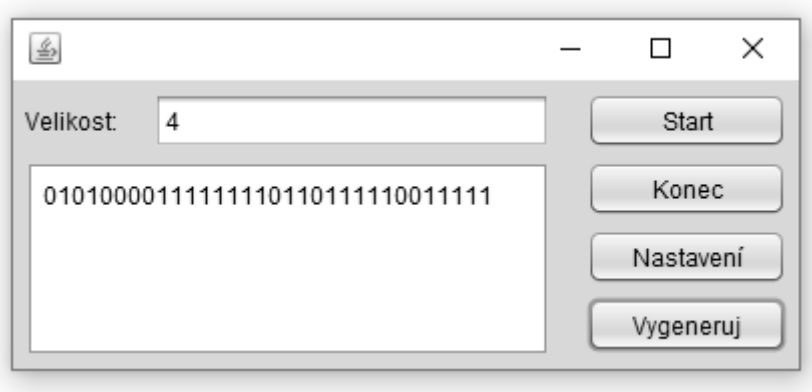
### IP a MAC adresa

Tyto hodnoty jsou veřejně známé a dohádatelné při troše snahy, tedy nevhodné pro použití.

# 5. Implementace RNG

Kód implementace generátoru náhodných čísel byl napsán v jazyce Java za použití vývojového prostředí Netbeans IDE 8.2 s JDK8, programovací jazyk Java byl použit hlavně proto, že je multiplatformní, tedy pro splnění zadání implementace pro operační systémy Linux a Windows stačí napsat jeden kód.

Implementace generátoru náhodných čísel je provedena ve formě knihovny, kterou lze implementovat pro jiné Java programy a používat její funkcionalitu. Pro ukázku byl vytvořen jednoduchý a intuitivní program s grafickým uživatelským rozhraním (GUI), který umožní spustit (a následně zastavit) generování dat a vracení (zobrazování) náhodných dat o požadované velikosti ve formě řetězce binárního čísla, které si následně každý může převést do formátu, které preferuje (číslo, text, atd.).




**Obrázek 20 – GUI aplikace**

### Souhrn

Principem implementace RNG je, podobně jak to funguje v systému Linux s jeho /dev/random, sběr vhodné entropie, která se odkládá do souboru, kde jí je možné vyzvednout pro pozdější použití. K splnění tohoto cíle je nejdříve nutné definovat vhodné zdroje entropie, které se mohou použít, toto bylo vypracováno a změřeno v mé vlastní bakalářské práci[1] a doplněno o zdroje, které bylo v průběhu implementace objeveny (viz kapitola č. 4 Zdroje entropie).

### Ošetření kritické sekce

Kritická sekce (kritický kód) je část kódu, ve kterém dochází k přístupu ke sdílené paměti dvěma a více procesy nebo vlákny, z toho vyplývá nebezpečí při zápisu do sdílené paměti, že se může porušit integrita dat. Z toho důvodu je potřeba zajistit exkluzivní přístup do kritické sekce.

V rámci Javy je možné použít (a v implementaci bylo použito) třídu „FileLock“ z balíčku „java.nio.channels“, jenž při zavolání metody lock() uzamkne soubor v rámci Java Virtual Machine pro dané vlákno a při zavolání metody relase() soubor odemkne.

### Použité knihovny

*Sigar.jar a log4j.jar*[13]

Knihovny, které dokážou zobrazit informace o využití hardware v počítači, v práci bylo použito pro získávání hodnot využití procesoru, fyzické paměti a využití disků.

*System-hook.jar*[14]

Knihovna, která byla použita pro sběr entropie ze stisků kláves, jelikož Java toto dokáže odchytávat pouze pro danou aplikaci a ne globálně, sběr této entropie by nebyl možný na pozadí bez této knihovny.

*Jnetpcap.jar*[15]

Tato knihovna umožňuje číst pakety, což umožňuje sběr entropie na síti. Pro funkčnost této knihovny je potřeba mít nainstalováno aplikační rozhraní „Pcap“ (WinPcap pro Windows, libpcap pro unixové systémy), návod lze nalézt na jejich stránkách[15] a mít spuštěnou aplikaci s oprávněním administrátora.

## 5.1 Třídy

### RandomNumberGenerator.java

RandomNumberGenerator je stěžejní třída, která obstarává jak sběr náhodných dat, tak i jejich vracení (generování), třída využívá funkcionalit třídy BinFile popsané níže. Konstruktor této třídy obsahuje dva důležité parametry:

*public RandomNumberGenerator(int mode, BinFile file)*

Parametr „mode“ umožňuje určení typu chování generátoru náhodných čísel, umožňuje zadání čísla „0“, jenž značí vhodné chování pro stroje používané jako server, tedy generátor nepoužívá zdroje entropie, které pro generování kvalitní entropie potřebují uživatelskou aktivitu a čísla „1“, které je určeno pro stroje, na nichž pracuje (vytváří aktivitu) uživatel. Parametr „file“ slouží pro určení souboru, do kterého se entropie odkládá.

Hlavními metodami této třídy jsou (podrobnější popis a zbylé, méně podstatné metody či ne nezbytně nutné pro fungování generátoru jsou k nalezení v dokumentaci, která je přiložena v přílohách):

*void start()*

Spuštění sběru náhodných dat, která jsou ukládána do souboru, tedy spuštění jednotlivých vláken, která obstarávají sběr náhodných dat z určených zdrojů entropie podle hodnoty parametru „mode“.

*void cancel()*

Ukončení sběru náhodných dat, tedy ukončení všech běžících vláken, která obstarávaly sběr dat ze zdrojů entropie.

*String getRandomNumberBin(int velikost)*

Tato metoda vrátí požadovanou (dle parametru) velikost náhodného čísla v binární hodnotě, měrná jednotka velikosti je v bajt, implementace jednotky bitu byla zbytečná, jelikož počítače dokážou adresovat jako nejmenší jednotku paměti právě bajt. Náhodné číslo je kombinací odložených náhodných dat v souboru a entropie, kterou jsme schopní získat z aktuálního času (tento zdroj lze používat jen jednorázově, jelikož pro dlouhodobý sběr v intervalech bude vracet velmi podobné hodnoty).

*setDisk(String disk)*

Metoda, která nastaví diskový oddíl („disk“) a spustí sběr náhodných dat založený na aktivitě disku. Tato metoda je speciální, jelikož jako jediná vyžaduje ruční nastavení ouživatele této implementace, pokud není nastavena, je sběr náhodných dat ze zdroje entropie pro třídu „HDD_Usage.java“ neaktivní.

### BinFile.java

BinFile je stěžejní třída pro fungování třídy „RandomNumberGenerator.java“, jelikož většina její implementace počítá s generováním náhodných dat ze sběru a odkládání náhodných dat do souboru, tudíž je zapotřebí této třídy, která obstarává soubor. Konstruktor této třídy je jednoduchý:

*public BinFile(String name, String path, int size)*

Paramentry „name“ a „path“ udávají umístění souboru na počítači a parametr „size“ omezuje velikost souboru, bez kterého by soubor mohl narůstat do velikosti, kdy zabere všechnu volnou paměť.

Hlavními metodami této třídy jsou (podrobnější popis a zbylé, méně podstatné metody či ne nezbytně nutné pro fungování generátoru jsou k nalezení v dokumentaci, která je přiložena v přílohách):

*void writeToFile(byte[] data)*

Tato metoda zapisuje pole bajtů do souboru, při zápisu je třeba ošetřit kritickou sekci, jelikož tuto metodu volá každá třída, která se zabývá sběrem entropie a může nastat případ, že by si přepisovali navzájem data a znehodnocovali soubor.

*byte[] readFromFile(int pocet)*

Metoda, která umožňuje přečtení uložených náhodných dat (bajtů) ze souboru o dané velikosti (parametr počet) a následnou reorganizaci souboru, tedy smazání přečtených dat z důvodu, aby nemohla být znovu použita a tím i znehodnocena kvalita generátoru. Princip čtení je založen na metodě FIFO.

### Rozhraní GetEntropy.java

Toto rozhraní objektu určuje, jaké metody jsou viditelné (veřejné) pro objekt zvenku, je použito ve všech metodách sběru entropie, která se může sbírat dlouhodobě:

void getRandomData() – metoda, která při zavolání zjistí použitelnou entropii a tu zapíše do souboru
public void startGenerating() – metoda, která vytvoří vlákno a v daných intervalech nebo při možnosti dalšího sběru entropie opakuje metodu getRandomData()
public void stopGenerating() – metoda která zastaví sběr entropie (zastaví vytvořená vlákna)

### CursorPosition.java

Třída, která obstarává sběr entropie z pozic kurzoru myši, tento zdroj je použitelný pouze pro stoje, kde generuje aktivitu uživatel, což je také ošetřeno při sběru entropie, kdy se nemůžou opakovat stejné souřadnice za sebou (značí neaktivitu uživatele). Tento zdroj se vyhodnocuje každou sekundu.

Principem sběru entropie je snímání pozic kurzoru myši, kdy tyto hodnoty jsou ořízntuty a použity jednotky a desítky z hodnot jednotlivých x a y souřadnic, tyto čísla vrací 13 b entropie, jelikož zapisovat do souboru je možné po celých bajtech, je 5 bitů odkládáno do dočasného pole, které je do souboru zapsáno po svém naplnění a následně vyprázdněno pro znovupoužití.

### CPU_Usage.java

Třída, která obstarává sběr entropie s využitím CPU, tedy jeho zatížení udávané v procentech, k čemuž byla použita knihovna Sigar[13], která umožňuje měřit využití až na 15-18 desetinných míst (dle pozorování). Pro sběr entropie jsou použity jednotky od 10 mili do jednotek femto ($10^{-15}$), čím lze získat 33 bitů entropie (použito 32 b pro snadnější implementaci). Tento zdroj se vyhodnocuje každou sekundu.

### RAM_Usage.java

Třída, která obstarává sběr entropie s využitím fyzické paměti, tedy jeho využití udávané v procentech, k čemuž byla použita knihovna Sigar[13], tedy implementace je prakticky stejná jako předchozí s drobnými úpravami, jako je zajištění aktivity, kdy entropie se sbírá, pokud se hodnoty změnili alespoň o jedno procento. Tento zdroj se vyhodnocuje každou sekundu.

### HDD_Usage.java

Tato třída obstarává sběr entropie s využitím statistik využití HDD, které poskytuje knihovna Sigar[13], konkrétně počet čtení a zápisů na disk, kde jsou použity jednotky každého atributu, tedy celkem 6 bitů entropie, která je ukládána do dočasného pole, dokud není možné zapsat celý bajt. Tento zdroj se vyhodnocuje každých 5 sekund.

### Network_Packets.java

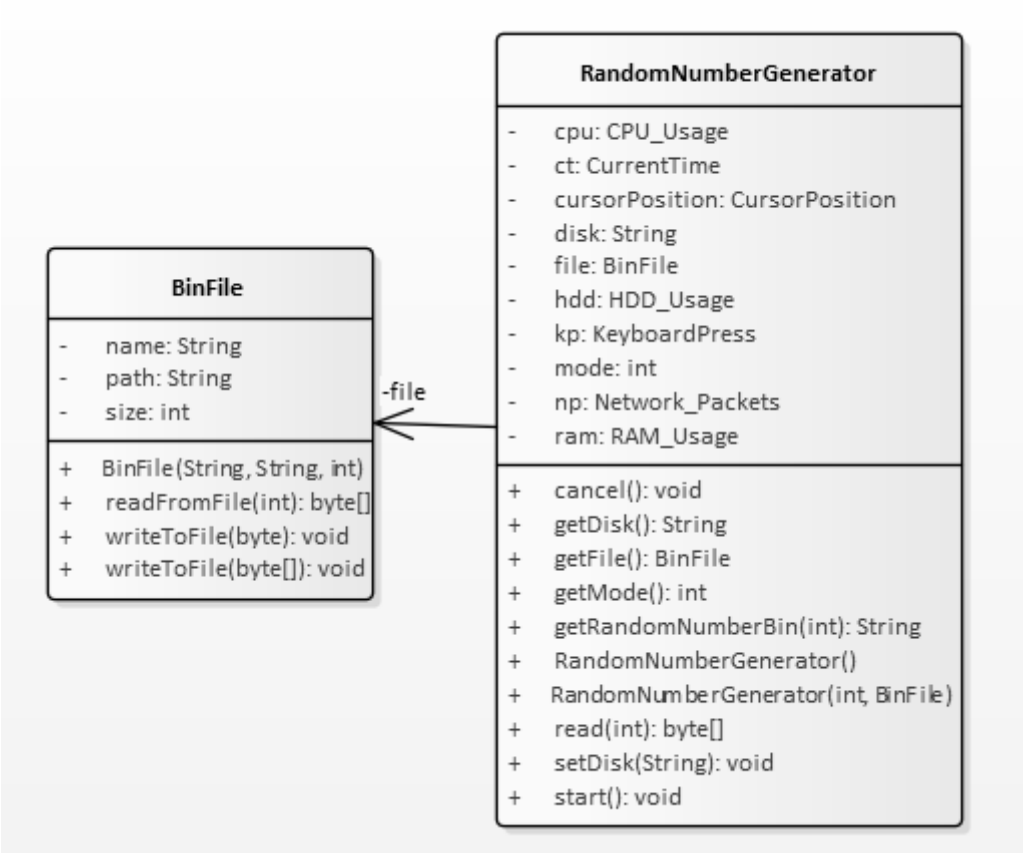
Třída, která obstarává sběr entropie za využití síťové aktivity, konkrétně času příchodu paketů, což umožňuje knihovna Jnetpcap.jar[15], kde jsou využívány pro sběr entropie hodnoty 10 ms až jednotky mikro sekund, tedy 5 číslic a 16 bitů entropie. Tato třída jako jediná ke svému sběru entropie potřebuje oprávnění administrátora. Tento zdroj se vyhodnocuje pokaždé, když je přijat paket.

### KeyboardPress.java

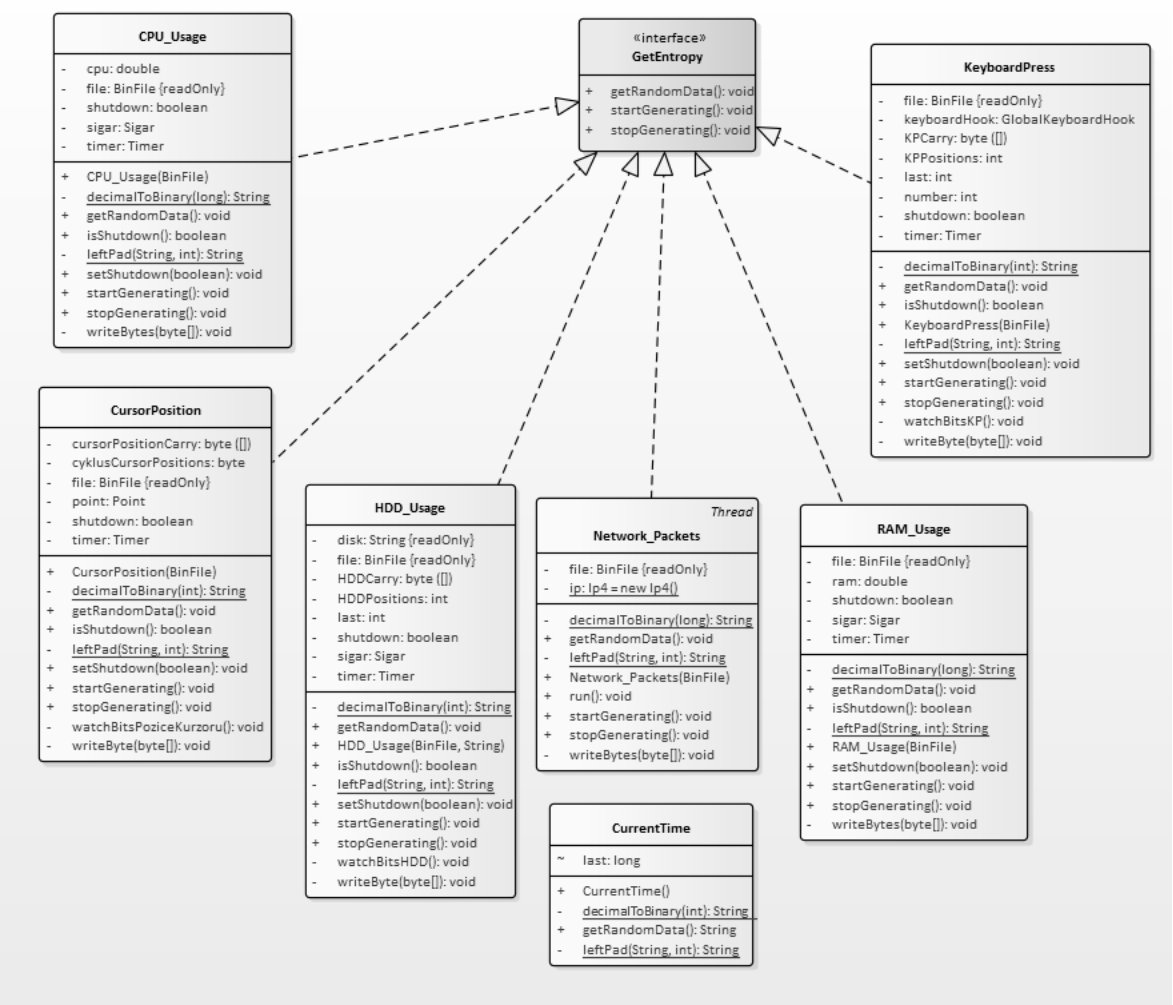
Třída, která monitoruje stisky kláves, k čemuž bylo zapotřebí použít knihovnu System-hook.jar[14], sbírá entropii ze zdroje po 60 sekundách a kontroluje pro aktivitu uživatele (nad 10 stisků), jako data jsou brány stisky v řádech jednotek a desítek, tedy 6 bitů entropie, jež jsou ukládána do dočasného pole, dokud není možné zapsat celý bajt. Použití výše zmíněné knihovny je z důvodu omezení Javy, která umožňuje zachytávat události stisků kláves pouze v rámci programu, což se vylučuje se sběrem entropie na pozadí z celého OS.

### CurrentTime.java

Třída, která svůj zdroj entropie nesbírá dlouhodobě, ale pouze na požádání, důvodem je, že kdyby se generovaly data po intervalech, hodnoty by byli málo variabilní. Protože se zdroj neukládá do souboru, je entropii možné brát pouze v bajtech, tedy konkrétně v této implementaci 8 bitů, protože Java v současné verzi neumožňuje měřit čas na více jak mikro sekundy.




**Obrázek 21 – UML diagram generátoru**




**Obrázek 22 – UML diagram datových tříd**

Podrobnější funkcionalitu je možné nalézt v dokumentaci, která je přiložena v přílohách, stejně jako zdrojové kódy všech tříd, ze kterých lze přečíst podrobný princip zpracování sbíraných dat.

## 5.2 Otestování generovaného souboru

Vytvořený generátor náhodných čísel byl spuštěn pro sběr náhodných dat na 17 min:

### První pokus

Generátor vygeneroval 6 704 bajtů náhodných dat.

```
Entropy = 0.999906 bits per bit.

Optimum compression would reduce the size
of this 53632 bit file by 0 percent.

Chi square distribution for 53632 samples is 6.98, and randomly
would exceed this value 0.82 percent of the times.

Arithmetic mean value of data bits is 0.4943 (0.5 = random).
Monte Carlo value for Pi is 3.244404655 (error 3.27 percent).
Serial correlation coefficient is 0.016653 (totally uncorrelated = 0.0).
```

**Obrázek 23 – První měření kvality**

K vygenerování tohoto obrázku byl použit program ENT[29], z jeho výsledků lze odvodit, že data jsou náhodná a to poměrně kvalitně.

### Druhý pokus

Generátor vygeneroval 4 730 bajtů náhodných dat, což je o něco méně než při prvním pokusu, ovšem na stroji byla zároveň mnohem menší uživatelská aktivita, tudíž zdroje entropie byli menší.

```
Entropy = 0.999976 bits per bit.

Optimum compression would reduce the size
of this 37840 bit file by 0 percent.

Chi square distribution for 37840 samples is 1.26, and randomly
would exceed this value 26.24 percent of the times.

Arithmetic mean value of data bits is 0.5029 (0.5 = random).
Monte Carlo value for Pi is 3.228426396 (error 2.76 percent).
Serial correlation coefficient is 0.011701 (totally uncorrelated = 0.0).
```

**Obrázek 24 – Druhé měření kvality**

Tento obrázek je také výstupem z programu ENT[29], jeho výsledky ukazují, že náhodná data jsou opět kvalitní.

### Třetí pokus

Po ověření funkčnosti v krátkých pokusech, byl proveden delší pokus, pro ověření stability a kvality v dlouhodobějším měřítku, čas, po který byla entropie generována, byla 1 hodina a snažil jsem se počítači poskytnout mnoho uživatelské aktivity. Generátor vygeneroval 30 591 bajtů.

```
Entropy = 0.999997 bits per bit.

Optimum compression would reduce the size
of this 244728 bit file by 0 percent.

Chi square distribution for 244728 samples is 1.18, and randomly
would exceed this value 27.68 percent of the times.

Arithmetic mean value of data bits is 0.4989 (0.5 = random).
Monte Carlo value for Pi is 3.141624166 (error 0.00 percent).
Serial correlation coefficient is 0.008903 (totally uncorrelated = 0.0).
```

**Obrázek 25 – Třetí měření kvality**

Z delšího měření se vytratily menší odchylky, které bylo možno pozorovat v předchozích pokusech, tedy lze předpokládat, že navržený generátor náhodných čísel je kvalitní.

# Závěr

V diplomové práci byly v teoretické části stručně vysvětleny pojmy týkající se entropie, jako je hustota informací, náhodnost a relativita entropie, které je nutné pochopit, aby bylo možné vůbec správně navrhnout implementaci, jež se opírá o tyto pojmy. Dále byla vysvětlena co možná nejstručněji historie kryptografie, její principy a vývoj v lidské historii, počínaje jako umění a zdobení hrobů pro vladaře, až po využívání v zabezpečení a její neustálé zdokonalování.

Dále byly uvedeny a popsány možnosti entropie a její způsob využití v moderní (dnešní) kryptografii, zmíněny nejznámější a nejkvalitnější kryptografické algoritmy jak symetrické, tak i asymetrické. Byly také uvedeny, dnes známe, metody útoků na kryptografické algoritmy.

V teoretické části byly také popsány a rozšířeny podklady z bakalářské práce[1], jež je třeba pro navržení implementace generátoru náhodných čísel. Prvním z nich jsou zdroje, z kterých lze čerpat entropii pro vlastní implementaci, a které lze považovat za kvalitní a které ne, popřípadě případy, v kterých se daný zdroj může použít. Druhým z podkladů, který je také velice důležitý, jsou způsoby a metodiky, kterým lze získaná data ze zdrojů entropie testovat na kvalitu náhodnosti, počínaje těmi nejsnadnějšími až po testy statisticky náročné.

V praktické části byly využity poznatky z teoretické části a byla navržena vlastní implementace generátoru náhodných čísel. Princip fungování generátoru byl z důvodu osvědčenosti použit princip, jenž se používá v unixových systémech, tedy průběžný sběr entropie a její ukládání do souboru, kde se následně v budoucnu může vyzvednout. V neposlední řadě byl tento navržený generátor náhodných čísel otestován testy, které bylo možné aplikovat na data, jež ukázali, že vytvořený generátor náhodných čísel je poměrně kvalitní.

Co se týče možností rozšíření, existují další zdroje entropie, které nebyly implementovány z důvodu jak časové náročnosti, tak i neuniverzálnosti a závislosti na hardware, avšak i přes to by bylo možné implementaci o tyto zdroje rozšířit. Další možnost úpravy by bylo udržování souboru ve fyzické paměti, pokud by se jednalo o malý soubor, pokud by se jednalo o velký soubor, tak by bylo možné optimalizovat ukládání souboru na disk.

# Literatura


1. FARKAS, Marek. *Dostupné zdroje entropie a jejich kvalita v systémech Linux a Windows* [online]. Pardubice, 2014 [cit. 2017-04-19]. Dostupné z: http://dspace.upce.cz/handle/10195/60838. Bakalářská práce. Univerzita Pardubice. Vedoucí práce Tomáš Hudec.
2. Statistics to use. KIRKMAN, T.W. *Kolmogorov-Smirnov Test* [online]. 1996 [cit. 2017-04-20]. Dostupné z: http://www.physics.csbsju.edu/stats/KS-test.html
3. Arithmetic Primitives for u.d mod 1. HELLEKALEK, Peter. *Spectral test* [online]. 1997 [cit. 2017-04-19]. Dostupné z: http://random.mat.sbg.ac.at/tests/theory/spectral
4. Testing Random Number Generators. DWYER, Jerry a K.B. WILLIAMS. *Testing Random Number Generators* [online]. 1996 [cit. 2017-04-20]. Dostupné z: http://www.drdobbs.com/testing-random-number-generators/184403185
5. A Brief History of Cryptography. *Redhat* [online]. 2013 [cit. 2017-04-28]. Dostupné z: https://access.redhat.com/blogs/766093/posts/1976023
6. A Short History of Cryptography. COHEN, Fred. *All.Net* [online]. 1995 [cit. 2017-04-28]. Dostupné z: http://all.net/edu/curr/ip/Chap2-1.html
7. Cryptography. BEAL, Vangie. *Webopedia* [online]. [cit. 2017-04-28]. Dostupné z: http://www.webopedia.com/TERM/C/cryptography.html
8. Cryptography. *ETHW* [online]. [cit. 2017-04-29]. Dostupné z: http://ethw.org/Cryptography
9. Od jantarové tyče k telegrafu. *Radiohistorie* [online]. [cit. 2017-05-01]. Dostupné z: http://radiohistorie.webnode.cz/telegrafie/
10. A Proposal for Secure Storage of Credit Card Data. FRIEDL, Steve. *Software Consulting Central* [online]. [cit. 2017-05-01]. Dostupné z: http://unixwiz.net/techtips/secure-cc.html
11. Cryptography Basics. *ETutorials* [online]. [cit. 2017-05-08]. Dostupné z: http://etutorials.org/Linux+systems/unix+internet+security/Part+II+Security+Building+Blocks/Chapter+7.+Cryptography+Basics
12. Password Recovery Speeds. *The Home Computer Security Centre* [online]. 2009 [cit. 2017-05-01]. Dostupné z: http://www.lockdown.co.uk/?pg=combi#classB
13. System Information Gatherer And Reporter. *GitHub* [online]. [cit. 2017-05-07]. Dostupné z: https://github.com/hyperic/sigar
14. Global Keyboard / Mouse Hook for Java applications. *GitHub* [online]. [cit. 2017-05-07]. Dostupné z: https://github.com/kristian/system-hook
15. JNetPcap. *Sly Technologies* [online]. [cit. 2017-05-07]. Dostupné z: http://jnetpcap.com
16. The DES encryption algorithm. *Ius mentis* [online]. 2005 [cit. 2017-05-08]. Dostupné z: http://www.iusmentis.com/technology/encryption/des
17. The Advanced Encryption Standard (Rijndael). *Cryptographic Compendium* [online]. [cit. 2017-05-08]. Dostupné z: http://www.quadibloc.com/crypto/co040401.htm
18. Advanced Encryption Standard. *Wikisofia* [online]. [cit. 2017-05-08]. Dostupné z: https://wikisofia.cz/wiki/Advanced_Encryption_Standard
19. Introduction to Blowfish. *Splashdata* [online]. [cit. 2017-05-08]. Dostupné z: http://www.splashdata.com/splashid/blowfish.htm

20. IDEA (International Data Encryption Algorithm). *A Cryptographic Compendium* [online]. [cit. 2017-05-08]. Dostupné z: http://www.quadibloc.com/crypto/co040302.htm
21. GALLI, Reto. MARS encryption algorithm. *Galli* [online]. [cit. 2017-05-08]. Dostupné z: http://reto.orgfree.com/us/projectlinks/MARSReport.html
22. A Candidate Block Cipher for the Advanced Encryption Standard. *SERPENT* [online]. [cit. 2017-05-08]. Dostupné z: http://www.cl.cam.ac.uk/~rja14/serpent.html
23. The Twofish Encryption Algorithm. *Drdobbs* [online]. [cit. 2017-05-08]. Dostupné z: http://www.drdobbs.com/security/the-twofish-encryption-algorithm/184410744
24. The RC5 Encryption Algorithm. *Drdobbs* [online]. [cit. 2017-05-08]. Dostupné z: http://www.drdobbs.com/security/the-rc5-encryption-algorithm/184409480
25. RC6® Block Cipher. *RSA Laboratories* [online]. [cit. 2017-05-08]. Dostupné z: https://www.emc.com/emc-plus/rsa-labs/historical/rc6-block-cipher.htm
26. Keccak: The New SHA-3 Encryption Standard. *Drdobbs* [online]. [cit. 2017-05-08]. Dostupné z: http://www.drdobbs.com/security/keccak-the-new-sha-3-encryption-standard/240154037
27. Entropy (information theory). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-04-29]. Dostupné z: http://en.wikipedia.org/wiki/Entropy_%28information_theory%29
28. *Tour of accounting*. 2001. Dostupné z: http://dilbert.com/strips/comic/2001-10-25/
29. WALKER, John. A Pseudorandom Number Sequence Test Program. In: *Fourmilab* [online]. 2008 [cit. 2014-04-29]. Dostupné z: https://www.fourmilab.ch/random
30. Statistical Analysis. *RANDOM.ORG* [online]. [cit. 2017-05-15]. Dostupné z: https://www.random.org/analysis/

# Ukázky kódů

### Kód č. 1 – Inicializace generátoru

```
import rng.BinFile;
import rng.RandomNumberGenerator;


public class RNG_test {


   public static void main(String[] args) throws InterruptedException {
      //inicializace generátoru
      BinFile file = new BinFile("data.rng", "data/", 80);
      RandomNumberGenerator rng = new RandomNumberGenerator(1, file);
      //nastavení oddílu
      rng.setDisk("C:");
      //spuštění sběru entropie
      rng.start();
      //čekání 10 sekund
      Thread.sleep(10000);
      //vypnutí sběru entropie
      rng.cancel();
      //výpis 16 bitů entropie do konzole
      System.out.println(rng.getRandomNumberBin(2));
      System.exit(0);
   }
}
```

**Kód č. 2 – Zápis do souboru (kritická sekce)**

```
public void writeToFile(byte[] data) {
    String soubor = path + name;
    try {
        RandomAccessFile file = new RandomAccessFile(soubor, "rw");
        if (file.length() >= size) {
            file.close();
            return;
        }
        FileChannel fc = file.getChannel();
        //uzamčení souboru
        FileLock lock = fc.lock();
        file.seek(file.length());
        file.write(data);
        //odemčení souboru
        if (lock != null) {
            lock.release();
        }
        fc.close();
        file.close();
    } catch (IOException e) {
        System.err.println("Chyba při zápisu do souboru");
    }
}
```

**Kód č. 3 – Spuštění a ukončení sběru entropie pro jednu metodu**

```
@Override
public void startGenerating() {
   this.timer.schedule(new TimerTask() {
      public void run() {
         if (isShutdown()) {
            CursorPosition.this.timer.cancel();
            CursorPosition.this.timer.purge();
            CursorPosition.this.timer = new Timer();
         }
         getRandomData();
      }
   }, 0L, 1000L);
}

@Override
public void stopGenerating() {
   setShutdown(true);
   CursorPosition.this.timer.cancel();
   CursorPosition.this.timer.purge();
   CursorPosition.this.timer = new Timer();
}
```