UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Šifrovací algoritmy

Michal Indra

Bakalářská práce

2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal Indra**
Osobní číslo:	**I09133**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Šifrovací algoritmy**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce se bude po teoretické i praktické stránce věnovat kryptografickým algoritmům a jejich názorné prezentaci.
Teoretická část se zabývá popisem a vysvětlením základních pojmů a principů z oblasti kryptologie, teoretickým popise vybraných šifrovacích algoritmů a zhodnocení používaných protokolů a dostupných knihoven pro podporu šifrování.
V rámci praktické části bude provedena implementace vybraných šifrovacích algoritmů a vytvoření aplikace pro demonstraci různých způsobů šifrování s využitím vybraných algoritmů. Aplikace bude vytvořena jako formulářová aplikace s důrazem na názornost principů fungování vybraných algoritmů.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

ZELENKA, Josef. Ochrana dat: kryptologie. Vyd. 1. Hradec Králové: Gaudeamus, 2003, 198 s. ISBN 80-704-1737-4.
SINGH, Simon. Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v čes. jaz. Překlad Petr Koubský, Dita Eckhardtová. Praha: Dokořán, 2009, 382 s. Aliter, sv. 9. ISBN 978-802-5701-447.
Bishop, D., Introduction to Cryptography with Java Applets. ISBN 0-7637-2207-3

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra softwarových technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2014

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 6 . 5. 2014

Michal Indra

Poděkování

Rád bych touto cestou poděkoval především vedoucímu této bakalářské práce, panu Ing. Petru Veselému, za jeho čas, odborné rady, pomoc s výběrem publikací a také především za jeho ochotu a nekonečnou trpělivost během příprav této práce. Velký dík patří také v neposlední řadě lidem v mém okolí, kteří mne podporovali a umožnili mi tak pokračovat ve studiu.

Anotace

Hlavním cílem této práce je seznámit uživatele s vybranými typy šifrovacích algoritmů, od nejjednodušších až po ty složitější. Účelem práce je tedy demonstrace způsobů šifrování, kdy si uživatel bude moci prostřednictvím jednotlivých šifer jak zabezpečit svá data, tak se pokusit dané algoritmy nejrůznějšími způsoby prolomit a ověřit tak jejich bezpečnost v praxi.

Klíčová slova

Šifra, šifrovací algoritmy, polyalfabetická šifra, vysoká prvočísla, prolomení šifer.

Title

Enciphering algorithms (presentation of making and breaking representative cyphers)

Annotation

The main goal of this work is introducing of user with randomly selected cypher algorithms from simpliest to complexed. Role of this work is exactly presentation of the various cypher ways, user is able through the different codes to secure own data or break those codes with diverse methods and verify their practice safety.

Keywords

Cypher, code algorhitms, polyalphabetical cypher, large prime numbers, breaking the secret codes.

Obsah

Seznam zkratek

3G – Third Generation

AES – Advanced Encryption Standard

ARC4 – Alleged Rivest Cipher 4

ARCFOUR – Alleged Rivest Cipher Four

ASCII – American Standard Code for Information Interchange

CCMP – Counter Cipher Mode with Block Chaining Message Authentication Code Protocol

DES – Data Encryption Standard

GSM – Groupe Spécial Mobile/Global System for Mobile Communications

HTTPS – Hypertext Transfer Protocol Secure

IBM – International Business Machines Corporation

MAC – Message Authentication Code

PDF – Portable Document Format

RDP – Remote Desktop Protocol

RSA – Rivest, Shamir, Adleman

RC4 – Rivest Cipher 4/Ron's Code 4

SSH – Secure Shell

SSL/TLS – Secure Socket Layer/Transport Layer Security

TKIP – Temporal Key Integrity Protocol

TripleDES – Triple Data Encryption Standard

UEA – Ultra Encryption Algorithm

UTF – Universal Character Set Transformation Format

WEP – Wired Equivalent Privacy

WLAN – Wireless Local Area Network

WPA – Wireless Fidelity Protected Access

WPA2 – PSK – Wireless Fidelity Protected Access 2 – Pre-Shared Key

XOR – Exclusive or

Seznam obrázků

Seznam tabulek

Použité konvence

Aby se odlišily jednotlivé prvky této dokumentace a dokumentace tak byla čitelnější, byla nastavena jednotná konvence zápisu.

`zdrojový kód`	Výňatek ze zdrojového kódu aplikace.
`balíčky, proměnné, metody, třídy, cesty, příkazy`	Cesty, příkazy, názvy balíčků, proměnných a tříd.
důležitý text	Podstatné části textu.
http://www.example.com/	Odkazy na internetové adresy.
$E_K(x) = (o + k) \bmod 65.536$	Vzorce, funkční hodnoty a konstanty.

Úvod

Bezpečnost a ochrana dat jsou v praxi často skloňovaná témata. Velké instituce ale i běžní uživatelé se snaží si své soukromí či „know-how“ hájit. K zamezení neautorizovaného přístupu k datům je třeba zajistit jejich ochranu, kterou se neoprávněné osoby či subjekty snaží prolomit. Je zde tedy nekonečný koloběh, kdy na jedné straně stojí kryptologové spolu s programátory a na straně druhé Ti, jež se jejich práci snaží zhatit. Vývoj musí jít neustále dopředu, šifry musí být složitější, doba zašifrování i dešifrování co možná nejkratší a strojový čas na prolomení šifry pokud možno nejdelší. V současnosti je kryptografie na tak vysoké úrovni, že prolomení šifrování, která používají běžní uživatelé, může zabrat desítky tisíc let strojového času na běžném počítači, kdežto proces zašifrování trvá v řádech milisekund.

V této práci se nahlédne jak na způsoby, kterými lze data zabezpečit, tak na možnosti prolomení těchto algoritmů. Uživatelské rozhraní bude demonstrovat šifrování a různé způsoby prolomení od nejjednodušších algoritmů až po ty obtížnější.

Dokumentace si klade za cíl seznámit čtenáře s vybranými šifrovacími algoritmy především po teoretické stránce. Dále zde budou uvedeny důležité části zdrojových kódů, které zajišťují bezchybný chod šifrovacích a dešifrovacích algoritmů. Teoretická část bakalářské práce bude především osvětlovat způsoby šifrování i dešifrování takovým způsobem, aby každý, kdo tuto práci bude číst, byl dostatečně seznámen se silnými a slabými stránkami jednodušších i složitějších šifer a mohl se případně i rozmyslet, jakým směrem se bude ubírat jeho práce, pokud by v budoucnu programoval a uváděl do praxe vlastní šifrovací algoritmus, který by mělo být obtížné prolomit, nebo by jeho prolomení mělo zabrat nereálně dlouhou dobu strojového času. Zároveň tato dokumentace může sloužit i těm, kteří chtějí svá data co nejlépe zabezpečit, a není v jejich zájmu studovat rozsáhlé kryptografické publikace, ale mají zájem nechat se alespoň trošku uvést do problematiky bezpečnosti dat.

Všichni, kteří mají zájem proniknout do tajů vybraných elementárních šifer, si v této práci mohou přijít na své. Každému, kdo touží poznat alespoň zčásti kryptologii a má zájem poznat šifry, by tato práce mohla do jisté míry pomoci.

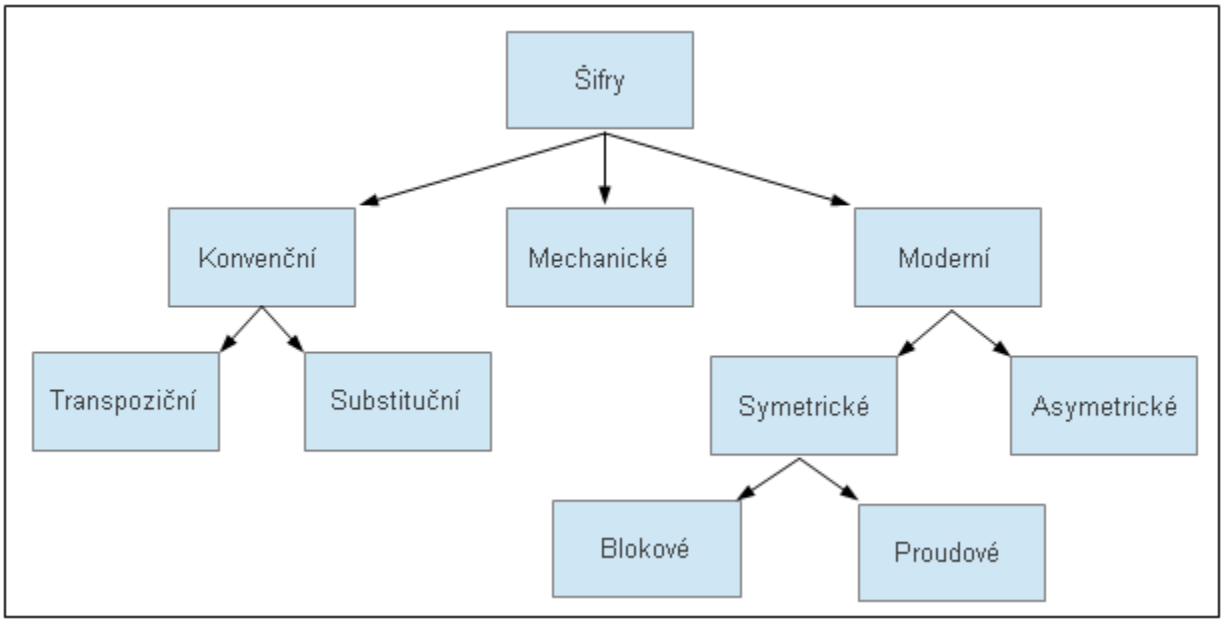
1 Teoretický úvod do kryptologie

Věda, která se zabývá mimo jiné utajováním informací, šifrováním nebo vznikem šifer se nazývá **kryptologie,** jež vznikla již ve starověku, aby byly citlivé informace uchovávány v tajnosti. Kryptologie je zároveň matematicko-vědním oborem, zabývajícím se kryptoanalýzou, kryptografii a steganografii**. Kryptoanalýzou** se rozumí věda, zabývající se metodami získávání obsahu šifrovaných informací, tedy ta část kryptologie zabývající se odhalením klíče k šifrovanému obsahu. **Kryptografie** je ta část kryptologie, která se zabývá převáděním zpráv ze srozumitelné do nesrozumitelné podoby a zpět, přesněji řečeno, zabývá se přímo vývojem šifer. Vědní disciplína, která se zabývá utajováním zpráv takovým způsobem, aby si toho útočník pokud možno nevšiml, se nazývá **steganografie.**

Otevřený text je zpráva, která vstupuje do procesu šifrování. **Šifrovaný text** je pozměněný otevřený text, který prošel procesem šifrování. Otevřený i šifrovaný text mají svou **abecedu,** což je množina symbolů, které jsou v daném textu použity.

Úprava otevřených dat na šifrovaná data se nazývá **kódování. Šifrování** znamená ukrytí dat před třetí stranou, která tato data nemá získat.

K zašifrování i dešifrování slouží **klíč**, který je společným tajemstvím pro komunikující strany a **klíčovým prostorem** rozumíme množinu všech klíčů, které se sdílí mezi uživateli. V případě komunikace uživatelů se serverem jde o klíčový prostor **virtuální**, v opačném případě jde o **fyzický** klíčový prostor.

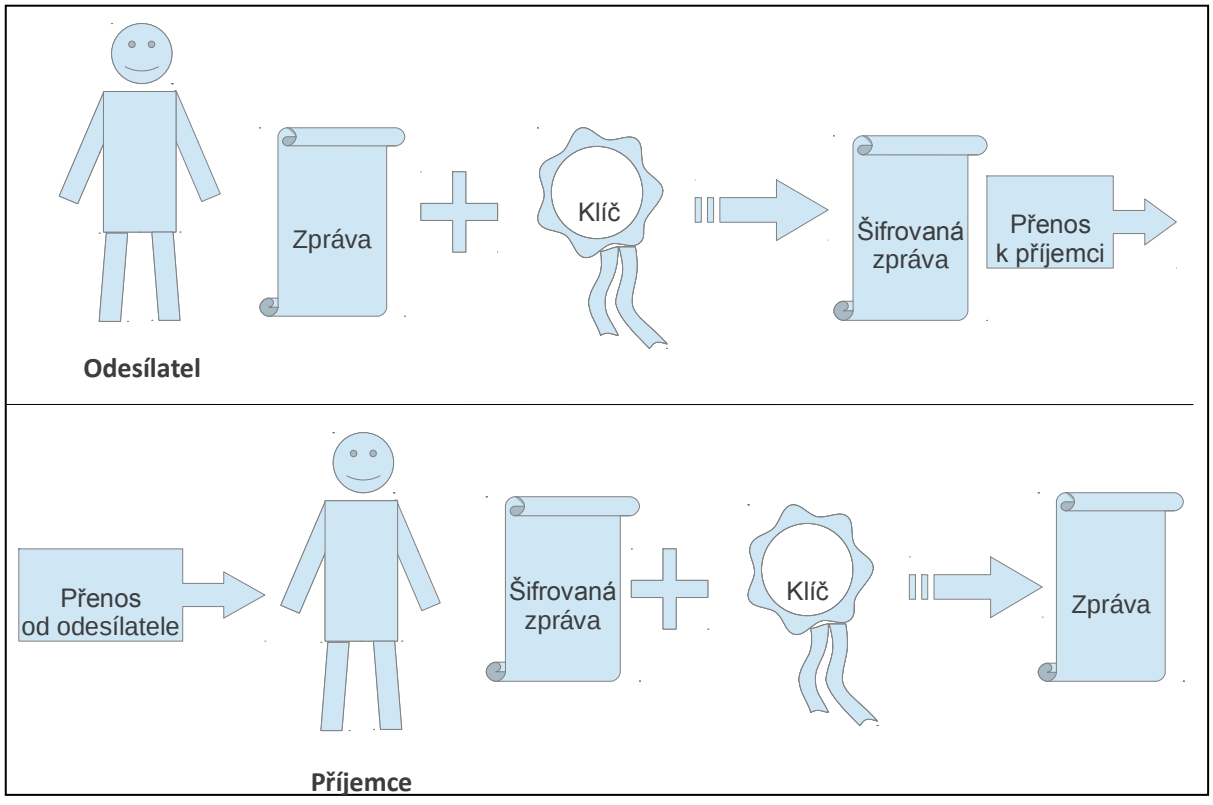


Obrázek č. 1 – Rozdělení šifer

Šifry se dělí na **konvenční**, **mechanické** a **moderní**. Kryptografie je dělena na základě přenosu klíče na **symetrickou** a **asymetrickou**. Symetrická kryptografie používá jeden klíč k šifrování i dešifrování, asymetrická používá dva klíče, z nichž jeden je **veřejný**, který je volně k dispozici, a druhý je **privátní**, který zůstává ponechán v tajnosti. Konvenční šifry používají pouze jeden klíč, jde tedy o symetrickou kryptografii, mezi ně patří mimo jiné monoalfabetická, polyalfabetická nebo Caesarova šifra. Příkladem mechanické šifry může být trezor, kufřík na kód nebo zámek na kolo s číselným kódem. U moderních šifer se používá symetrická i asymetrická kryptografie. Dále se šifry dělí na **proudové** a **blokové**. Proudové šifry zpracovávají data po bitech, kdežto blokové po jednotlivých blocích.

1.1 Symetrická kryptografie

Jak již bylo uvedeno, v symetrické kryptografii je použit pouze jeden klíč, který slouží k šifrování i dešifrování zprávy. Obrázek č. 2 popisuje obecný princip komunikace užitím symetrické kryptografie. Mezi hlavní výhody symetrického šifrování patří především rychlost. Algoritmy jsou jednoduché a rychlé, tudíž symetrické šifry nevyžadují velké množství strojového času. Avšak symetrické šifrování má i své negativní vlastnosti.



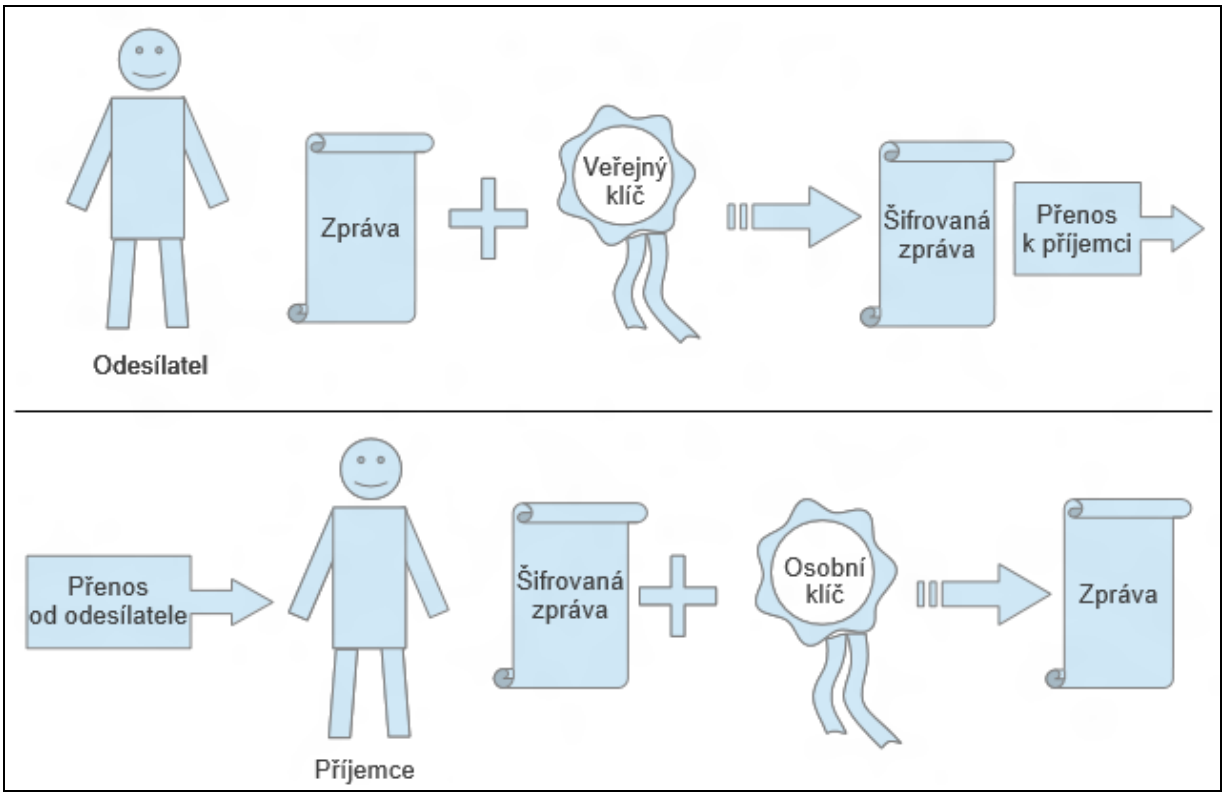
Obrázek č. 2 – Princip symetrického šifrování [zdroj: vlastní tvorba]

Ještě před zahájením komunikace obou stran je potřeba vyřešit přenos klíče ke straně druhé. Při komunikaci je potřeba více klíčů v případě, že se komunikuje s více stranami. V moderní kryptografii jsou symetrickými šiframi mimo jiné algoritmy AES (Advanced Encryption System), RC4, Blowfish nebo TripleDES.

1.2 Asymetrická kryptografie

V případě asymetrického šifrování, jak již bylo zmíněno, se používají dva klíče neboli **klíčový pár**. **Veřejný klíč** je volně přístupný všem, kteří chtějí zahájit komunikaci, ale zároveň musí být chráněný tak, aby jej nešlo změnit. Osobní neboli **privátní klíč** mají k dispozici pouze komunikující, nesmí být nikde zveřejněn a je třeba jej uchovat v tajnosti. Obecný princip asymetrického šifrování je znázorněn na obrázku č. 3. Strana odesílající tajnou informaci svou informaci zašifruje prostřednictvím veřejného klíče tak, aby jí bylo možné dešifrovat pouze privátním klíčem. Šifrovaná zpráva pak může být přenesena i nezabezpečenou komunikací a strana přijímající zprávu jí dešifruje svým privátním klíčem.

Klíčový pár pro komunikaci musí obsahovat veřejný a privátní klíč, které nesmí být shodné a ani odvoditelné. Zároveň by měli být dostatečně dlouhé, aby byl výrazně obtížnější útok hrubou silou nebo rozkladem čísla na dělitele.

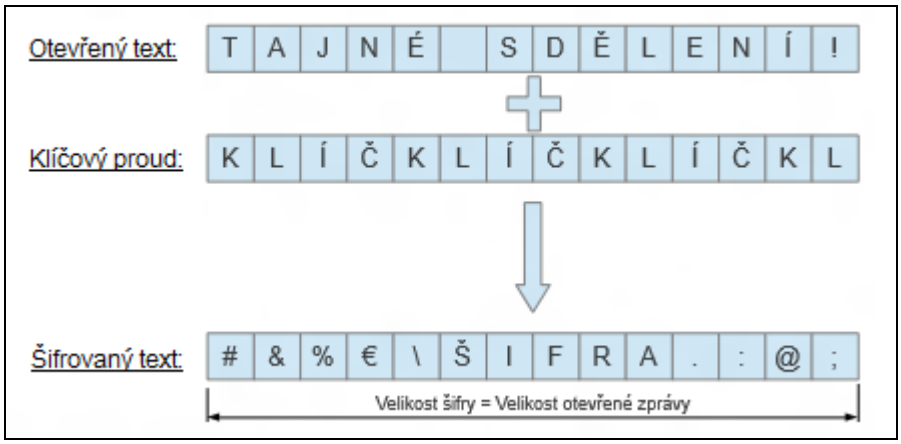


Obrázek č. 3 – Princip asymetrického šifrování [zdroj: vlastní tvorba]

Mezi klady asymetrických šifer lze zařadit fakt, že není třeba přenášet klíč k dešifrování a zároveň není třeba uchovávat velké množství klíčů. Naopak mezi zápory nepochybně patří mnohem větší náročnost na strojový čas, přičemž využití asymetrických šifer bývá i 1.000x pomalejší oproti symetrickým.

1.3 Proudové šifry

Proudové šifry jsou takové šifry, které zpracovávají data bit po bitu. Klíč je využit do takové míry, jak dlouhá je otevřená zpráva, která vstupuje do šifrovacího či dešifrovacího procesu (obecný princip je znázorněn na obrázku č. 4). Existuje několik variant implementace. Jednou z možných variant je situace, kdy se klíč generuje pomocí náhodných čísel, dokud nebude stejně dlouhý jako zpráva, přičemž se stejný klíč již z bezpečnostních důvodů znovu nepoužije. Taková situace nastává u Vernamovy šifry. Další možnou variantou je situace, kdy se klíč použije pouze za účelem generování tajné substituce bajtů, tedy nahrazení bajtu za bajt. Výsledkem je pak vygenerovaný konečně dlouhý klíč, který se použije na šifrovací a dešifrovací procesy, což je typické například pro šifru RC4.



Obrázek č. 4 – Princip proudové šifry [zdroj: vlastní tvorba]

K dalším významným zástupcům proudových šifer patří nepochybně šifry A5/1[1] a A5/2, které se používají k šifrování komunikace mobilními telefony běžící přes protokol GSM (global system for mobile communication), dále šifra E0[2], sloužící k šifrování protokolu Bluetooth, nebo například šifra SNOW[3], využívající šifrovací algoritmy UEA[4] 3 a UEA 2, pro zabezpečení mobilních sítí třetí generace (3G).

1 Více o šifře A5/1 pro GSM: http://home.anadolu.edu.tr/~nat/A5_1.ppt

2 Více o šifře E0 pro Bluetooth: http://eprint.iacr.org/2006/072.pdf

3 Více o šifře SNOW: http://www.m-hikari.com/ces/ces2010/ces1-4-2010/orhanouCES1-4-2010.pdf

4 Více o symetrických šifrách UEA (Ultra Encryption Algorithm) s náhodnými bity a odezvovým mechanizmem: http://research.ijcaonline.org/volume49/number5/pxc3880687.pdf

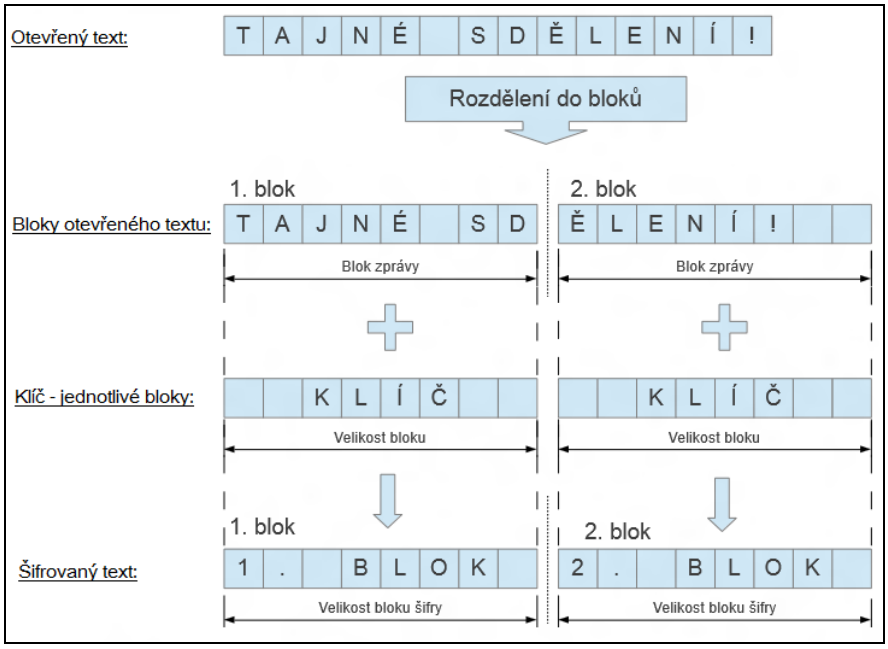
1.4 Blokové šifry

Na rozdíl od proudových šifer jsou blokové šifry rozděleny do jednotlivých bloků. Nejprve je potřeba otevřenou zprávu rozdělit do bloků, kde velikost bloku otevřené zprávy je shodná s klíčem i s blokem šifrované zprávy.

Blokové šifry pracují s fixní transformací bloků otevřené zprávy. Tento fakt lze brát jako výhodu, „neboť mohou předejít spotřebě strojového času během bitových operací“ a „pracují s daty v blocích počítačové velikosti“.

Obecně vzato, bloková šifra je takovou šifrou, kde všechny bloky otevřeného textu jsou šifrovány do bloků šifrovaného textu stejnou transformací a všechny šifrované bloky jsou dešifrovány na otevřený text opět stejnou transformací.

První blokovou šifrou byla šifra Lucifer[5], vytvořena Horstem Feistelem[6] v roce 1971, pracujícím u společnosti IBM. Tato šifra měla později komerční využití v elektronickém bankovnictví a byla přímým předchůdcem šifry DES[7] (Data Encryption Standard). Mezi další známé blokové šifry patří AES[8] (Advanced Encryption Standard), Triple DES[9] nebo Blowfish[10].



Obrázek č. 5 – Princip blokové šifry [zdroj: vlastní tvorba]

[5] Více o šifře Lucifer: http://fuseki.com/lucifer.pdf
[6] Stručně o Horstu Feistelovi: http://en.wikipedia.org/wiki/Horst_Feistel
[7] Více o šifře DES: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
[8] Více o šifře AES: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[9] Více o algoritmu Triple DES (3DES), modifikaci algoritmu DES: http://www.cryptosys.net/3des.html
[10] Více o šifře Blowfish: http://www.splashdata.com/splashid/blowfish.htm

2 Monoalfabetické substituční šifry

2.1 Stručný úvod do monoalfabetické šifry

Monoalfabetická šifra patří mezi nejjednodušší šifrovací algoritmy. Pro tuto práci byla zvolena jako zástupce šifer, které nepotřebují nějaké zvláštní znalosti matematiky. Dále se již práce o podobně jednoduchých šifrách nebude zmiňovat. Tuto šifru není tak obtížné prolomit neboť na základě frekvenční analýzy lze celkem rychle tuto šifru rozkrýt.

Název šifry vychází ze slov „mono“, tedy jedno a slovo „alfabet“ znamená abeceda. Jde tedy o substituci, výměnu, znaků jedné abecedy za znaky jiné abecedy. Každý znak otevřeného textu je v šifrovaném textu reprezentován jiným znakem. Jak je znázorněno na obrázku, každému znaku otevřeného textu odpovídá právě jiný znak šifrovaného textu.

Nezáleží na tom, zda je znak latinky nahrazen číslem, interpunkčním znakem nebo znakem čínské abecedy, na šifru tato situace nebude mít vliv.

2.1.1 Historie

Již ve starověku byla monoalfabetická šifra zmíněna v Caesarově díle „Zápisky o válce Galské“, kde je uvedeno, že v této verzi monoalfabetické šifry šlo o záměnu římských znaků za řecké.

K prolomení monoalfabetických šifer došlo v průběhu 9. století na blízkém východě, kdy si při studiích Koránu islámští teologové začínali všímat, že v Koránu se opakují některá slova častěji a některé znaky jsou četnější než jiné. Díky tomu nakonec arabský polyhistor Al-Kindí (neboli Alkindus) popsal techniku známou jako **frekvenční kryptoanalýza**. Ve středověké Evropě k tomuto objevu došlo až v 16. století, kdy italský učenec a kryptograf Giovanni Battista Porta ve svém díle „De Furtivis Literarum Notis“ popsal návod na luštění monoalfabetických substitučních šifer. Mimo jiné v tomto díle uvedl několik metod pro luštění polyalfabetických šifer, rozdělil šifry na substituční a transpoziční a zároveň představil první digrafickou šifru. Jeho dílo se stalo pro následující 3 století nejstěžejnějším dílem kryptologie v Evropě.

2.2 Šifrovací algoritmus monoalfabetické šifry

V této šifře jde o jednoduchou monoalfabetickou substituci znaků, tedy o záměnu každého znaku za znak jiný. V případě užití abecedy latinky bez diakritiky se jedná o 26! permutací (při užití náhodné transformace), tedy přibližně $4{,}03 \times 10^{26}$ možných kombinací, což by v případě útoku hrubou silou, kdy by byla zkoušena každou vteřinu jiná substituce znaků, znamenalo, že kryptoanalýza by trvala miliardkrát déle, než je dnešní odhad stáří vesmíru. V případě útoku frekvenční analýzou nemusí být toto číslo bráno vůbec v potaz, neboť správně napsaný algoritmus frekvenční analýzy, spuštěný na běžném stolním počítači, dokáže prolomit tuto šifru v řádu milisekund.

Existuje více možností tvorby substituce abecedy otevřeného textu. První možností je zaměnit mezi sebou jednotlivé znaky abecedy otevřeného textu. Druhou možností je tvořit substituci znaků abecedy otevřeného textu tak, že do substituce budou zahrnuty ještě jiné znaky. Třetí možností je náhrada znaků otevřeného textu například znaky sanskrtu (pozn. Ve starověku obdobně šifroval Caesar - používal místo římských znaků znaky řecké). Lze také jednotlivé znaky otevřeného textu nahradit shluky znaků, ale to již není monoalfabetická substituce, nýbrž o polygrafickou substituci (např.: v případě dvojic jde o bigrafickou substituci).

Existuje více druhů monoalfabetických šifer:

- jednoduchý posun – např. Caesarova šifra
- afinní šifra
- monoalfabetická substituce s klíčem
- náhodná transformace

2.3 Caesarova šifra

2.3.1 Teoretický úvod do Caesarovy šifry

Caesarova šifra patří stejně jako polyalfabetická šifra mezi monoalfabetické substituční šifry. Ve stručnosti to znamená, že každému znaku je přiřazen jiný znak, tedy dochází k substituci množiny znaků za jinou množinu znaků. U Caesarovy šifry se jedná o konstatní posun o „n“ znaků v abecedě. Jak již název samotné šifry napovídá, jako první tuto šifru popsal již ve starověkém Římě Gaius Julius Caesar ve svém díle “Zápisky o válce Galské”. Již tehdy měla tato šifra své využití a to ve vojenské komunikaci. Tehdy se však používal posun o 3 znaky v abecedě, ale obecně lze mluvit o jakémkoliv posunu znaků o určitou konstantu, která je klíčem pro zašifrování i dešifrování.

Znak otevřeného textu	A	B	C	D	E	F
Znak šifrovaného textu	D	E	F	G	H	I

Tabulka č. 1 – Původní Caesarova šifra – posun o 3 znaky v abecedě

Modulární aritmetikou lze popsat Caesarovu šifru pro běžnou abecedu bez diakritiky vzorečkem č. 1:

(1)

$$Posun(o,k) = (o+k) \bmod 26$$

kde o – je pozice znaku k zašifrování v abecedě,

k – je klíč šifry a modulo dělení odstraní přesah znaku mimo rozsah používané abecedy.

Příklad:

Bude-li znak k zašifrování „z“ (*o = 25*) a klíč k zašifrování bude konstanta o hodnotě *1*, pak dostaneme dosazením do vzorečku (1):

$$Posun(25,1) = (25+1) \bmod 26 = 26 \bmod 26 = 0,$$

čemuž odpovídá v abecedě znak “a”. Tento vzoreček využijeme k vytvoření funkce

(2)

$$E_K(x) = Posun(x[i], k),$$

kde *x* – je zašifrovaný text,

k – je klíčem,

x[i] – je znakem v originálním textu na pozici *i*, pro všechna *i*, která jsou menší nebo rovna délce originálního textu a zároveň větší nebo rovna *1*.

Dešifrování šifrovaného textu je analogicky

(3)

$$D_K(x) = ZpPosun(x[i], k) = (c - k + 26) \bmod 26,$$

kde *c* – je zašifrovaným znakem,

k – je klíčem

26 – je délka standardní abecedy bez diakritiky, pro všechna *i* větší než 1 a zároveň menší nebo rovna délce šifrovaného řetězce.

2.3.2 Caesarova šifra a Unicode

Nyní se již dostáváme k samotné implementaci Caesarovy šifry, kde již nebudeme pracovat pouze se standardní abecedou bez diakritiky o 25 znacích, nýbrž s 16bitovým Unicodem. Z toho vyplývá, že nyní budeme pracovat s abecedou, která má 2^{16} = 65.536 znaků. Analogicky se nám i mění vzoreček pro šifrování i dešifrování, kde se již nebude dělit modulem 25, ale 65.536, pro rozsah znaků 0 až 65.535.

Vzhledem k tomu, že budeme tuto šifru implementovat v programovacím jazyce Java, budeme využívat právě již zmíněné kódování UTF-16 (Universal character set transformation format dle

mezinárodního standardu ISO 10646). Vzoreček funkce pro šifrování textu bude vypadat následovně:

(4)

$$E_K(x) = (o + k) \bmod 65.536$$

Funkce bude provedena pro každý znak zprávy *o* s konstantním klíčem *k*. Dešifrování standardní cestou se provede analogicky podle vzorce (5):

(5)

$$D_K(x) = (c - k + 65.536) \bmod 65.536,$$

kde *c* – je každý šifrovaný znak šifrované zprávy,

k – je klíčem zprávy a

65.536 – je počet znaků UTF-16.

2.3.3 Útok na Caesarovu šifru

2.3.3.1 Útok hrubou silou

Jak již vyplývá z demonstrovaných vzorečků, prolomit Caesarovu šifru nebude složité. V případě běžné abecedy latinky bez diakritiky je počet klíčů 25 – 1 (možnost bez posuvu), tedy 24, z čehož vyplývá, že při útoku hrubou silou se v nejhorším případě na 24. pokus trefíme. V případě implementace Caesarovy šifry v programovacím jazyce Java s UTF-16 je k dispozici 65.534 možností nalezení konstantního klíče. Je to sice mnohem více možností než u běžné abecedy, ale budoucnost, kdy útočník najde klíč ke zprávě buď náhodnými pokusy, nebo systematickým postupem, není nějak výrazně vzdálená.

2.3.3.2 Útok pomocí frekvenční analýzy

Stejně jako u monoalfabetické šifry lze tuto šifru napadnout metodou frekvenční analýzy. Na rozdíl od monoalfabetické šifry však nemusíme rozkrývat všechny znaky, nýbrž postačí, když nalezneme nejčetnější znaky a podle nich již není žádným problémem klíč odkrýt. Používáme-li větší znakovou sadu, jako je v našem případě právě UTF-16, vyplatí se útok frekvenční analýzou více než útok hrubou silou.

Některé jazyky mají velmi podobnou četnost znaků v běžném textu, jiné mají zase zcela odlišnou četnost. Jako příklad lze uvést, že anglický a německý jazyk mají nejčetnější znak „E“, zatímco například český jazyk má obecně nejčetnější znak „O“. Anglický a německý jazyk mají mnohem podobnější tabulku četnosti znaků ve srovnání s českým jazykem. Jde především o fakt, že anglický a německý jazyk pochází z rodiny germánských jazyků a tyto dva jazyky prošly

podobným vývojem, naopak čeština se řadí mezi jazyky slovanské. V případě, kdy by útočník nevěděl, na jaký jazyk útočí, měl by alespoň odhadnout, z jaké skupiny jazyků jazyk otevřeného textu pochází. Věděl-li by, že útočí na germánský jazyk, mohl by považovat za nejčetnější znak otevřeného textu znak „E“ (např. holandský jazyk má nejčetnější znaky „E“, „N“, „T“ a „A“), zatímco u slovanských jazyků půjde spíše o znak „O“ (pořadí ve slovenském jazyce „A“, „O“, „E“, „S“). V každém případě by si útočník měl být vědom, že nejčetnějším znakem nemusí být právě hláska, nýbrž naopak tímto znakem může být mezera mezi jednotlivými slovy.

Pokud standardní frekvenční analýza pro hledání nejčetnějšího znaku selže a nejčetnější znak otevřeného textu nebude ani nejčetnější znak jazyka, ani mezera, může útočník hledat dle četnosti jednotlivé bigramy (dvojice znaků) a trigramy (trojice znaků), které jsou nejčetnější v daném jazyce. Samozřejmě si opět musí útočník uvědomit, že pokud otevřený text obsahuje mezery, bude muset postupovat podle tabulek nejčetnějších bigramů a trigramů zachycujících nejčetnější dvojice a trojice znaků včetně mezer, které mohou být výrazně odlišné od tabulek četností bigramů a trigramů pro otevřené texty psané bez mezer.

Český jazyk		Anglický jazyk		Německý jazyk	
O	8.67%	E	12.70%	E	16.93%
E	7.70%	T	9.06%	N	10.53%
N	6.54%	A	8.17%	I	8.02%
A	6.22%	O	7.51%	R	6.89%
T	5.73%	I	6.97%	S	6.42%
V	4.66%	N	6.75%	T	5.79%
S	4.52%	S	6.33%	A	5.58%
I	4.35%	H	6.09%	D	4.98%
L	3.84%	R	5.99%	H	4.98%
K	3.74%	D	4.25%	U	3.83%
R	3.70%	L	4.03%	L	3.60%
D	3.60%	C	2.78%	C	3.16%
P	3.41%	U	2.76%	G	3.02%
Í	3.27%	M	2.41%	M	2.55%
M	3.23%	W	2.36%	O	2.24%
U	3.14%	F	2.23%	B	1.96%
Á	2.24%	G	2.02%	W	1.78%
Z	2.20%	Y	1.97%	F	1.49%
J	2.12%	P	1.93%	K	1.32%
Y	1.91%	B	1.49%	Z	1.21%
Ě	1.65%	V	0.98%	V	0.84%
C	1.61%	K	0.77%	P	0.67%
B	1.56%	J	0.15%	Ü	0.65%
É	1.33%	X	0.15%	Ä	0.54%
H	1.27%	Q	0.10%	ß	0.37%
Ř	1.22%	Z	0.07%	Ö	0.30%
Ý	1.07%			J	0.24%
Ž	1.00%			X	0.05%
Č	0.95%			Y	0.05%
Š	0.81%			Q	0.02%
Ů	0.69%				
F	0.27%				
G	0.27%				
Ú	0.10%				
Ň	0.08%				
X	0.08%				
Ť	0.04%				
Ó	0.03%				
Ď	0.02%				
W	0.01%				
Q	0.00%				

Tabulka č. 2 – Četnost znaků v Češtině, Angličtině a Němčině (pro frekvenční analýzu) [zdroj: vlastní tvorba]

Nejprve je třeba zjistit, který znak je v šifrované zprávě nejčetnější. Stejně jako u útoku frekvenční analýzou na monoalfabetickou šifru bychom měli vědět, jakým jazykem je původní zpráva napsána. Máme-li zjištěný nejčetnější znak šifrované zprávy a víme-li, jakým jazykem je zpráva napsána, máme prakticky vyhráno. Uveďme si pár příkladů:

Ukázkový příklad č. 1:

Byla zachycena šifrovaná zpráva „єѡѪѮӽաѩѦѲѯѶѡѮӽѴѥѸѴѡѴѡѪѮөѳѤԛЮ“ a je známo, že je psána v českém jazyce.

Krok 1: Zjistili jsme, že v této zprávě je nejčetnějším znakem „ѡ“.

Krok 2: V českém jazyce je nejčetnějším znakem „a“.

є**ѡ**ѪѮӽաѩѦѲѯѶ**ѡ**ѮӽѴѥѸѴ**ѡ**Ѵ**ѡ**ѪѮөѳѤԛѬѥѮӭЮ

Ve zprávě je nejčetnějším znakem „ѡ“ a „Ѯ“. Zkusme nejprve pracovat se znakem „ѡ“. Posun mezi „a“ a „ѡ“ je 1.024 znaků ve znakové sadě UTF-16.

Krok 3: Máme tedy podezření, že znak „ѡ“ by mohl být znakem „a“ a znaky ve zprávy by potencionálně mohly být posunuty o 1.024 znaků. Dosadíme tedy do dešifrovací funkce *k = 1.024* a zkusíme, jestli zpráva bude dávat smysl. Každý znak tedy posuneme podle funkce (5): $D_K(x) = (c - 1.024 + 65.536) \bmod 65.536$.

Krok 4: Posunem o 1.024 pozic každého znaku, tedy použitím dešifrovacího klíče *k = 1.024* jsme získali zprávu: „Tajnýšifrovanýtextatajnésdělení.“

Autor zprávy zprávu napsal bez mezer a po oddělení slov zpráva dává smysl: „**Tajný šifrovaný text a tajné sdělení.**“ Útok frekvenční analýzou byl rychlý a úspěšný.

Ukázkový příklad č. 2:

Během komunikace byl získán řetězec šifrovaného textu:

„ήϕϓΌϑϚϏϥϜϔϑϞϑϐΌϙϑϟϟύϓϑΌϣϕϠϔΌϟϙύϘϘΌϟϑϏϞϑϠΚ“ a je známo, že je psán anglicky.

Krok 1: V této zprávě je nejčetnějším znakem „ϑ“.

Krok 2: V anglickém jazyce je nejčetnějším znakem „e“ a ten je od znaku „ϑ“ v UTF-16 vzdálen o 876 pozic.

Krok 3: Pokusíme se dešifrovat zprávu s klíčem *k = 867*.

Krok 4: Výsledkem je zpráva „**Big encyphered message with small secret.**“ a tato zpráva nám v anglickém jazyce dává smysl.

Autor zprávy v textu použil i mezery, přesto byl znak „e“ ve zprávě nejčetnější. Ukažme si ještě jeden příklad, kdy tomu tak nemusí být.

Ukázkový příklad č. 3:

Byla objevena zpráva „ƅησʠśƱƤƧƧśβƠƲƵƲśƵưśɓƜưƮƠśƝƧƠƤƝƠΣũ“ a je známo, že je psána německým jazykem.

Krok 1: Nejčetnějším znakem zprávy je znak „ś“.

Krok 2: Nejčetnějším znakem německého jazyka je stejně jako u anglického jazyka „e“. Vzdálenost mezi „ś“ a „e“ je 246 pozic v UTF-16.

Krok 3: Pokoušíme se dešifrovat klíčem k = 246.

Krok 4: Výsledkem je zpráva „Ž ¨e¼®±±e¯ª¹¿¹e¿ºe□¦º¸ªe§±ª®§ª³s“ a ta v německém a s největší pravděpodobností ani v jakémkoliv jiném jazyce nedává smysl.

Krok 5: Mohla by být tím nejčetnějším znakem šifrovaného textu mezera mezi slovy? Vzdálenost mezi „ś“ a mezerou je 315 pozic v UTF-16.

Krok 6: Užitím dešifrovacího klíče *k = 315* získáváme zprávu „**Ich will jetzt zu Hause bleiben.**“ a tato zpráva již dává smysl.

Při užití frekvenční analýzy je možné, že útočník narazí na problém, kdy nejčetnější znak šifrované zprávy není nejčetnějším znakem jazyka, ve kterém je zpráva psána a ani mezerou mezi slovy. V případě tohoto neúspěchu frekvenční analýzy nemusí útočník „házet flintu do žita“ a může zkusit, zda nejčetnější znak šifrované zprávy není druhým, třetím, čtvrtým nebo ještě méně četným znakem co se týče využití v jazyce zprávy. Zpravidla by útočníkovi řešení této problematiky nemělo zabrat mnoho času. V úplně nejhorším případě se za předpokladu, že šifrovaná zpráva je textová a psaná latinkou bez diakritiky, útočník zprávu rozkryje nejhůře při 52. pokusu (26 malých písmen a 26 velkých písmen). V případě užití znaků včetně české diakritiky by se jednalo v nejhořším případě o 82. pokus a to ještě v případě, že by se ve zprávě vyskytoval pouze jediný druh znaku.

2.4 Afinní šifra

Přestože speciálním případem afinní šifry může být i jednoduchý posun, afinní šifra využívá modulární aritmetiku, kdy jednotlivé znaky šifrovaného textu získáme vztahem (6):

(6)

$$C(x) = A \times P(x) + B,$$

kde $C(x)$ – je šifrovaný znak

$P(x)$ – je původní znak

B – je číslem ležícím na intervalu $\langle 0; 25 \rangle$, stupeň posunu šifry a

A – je takové číslo ležící na intervalu $\langle 1; 26 \rangle$ takové, aby platilo, že nejvyšší společný dělitel čísla A a čísla 26 je 1.

Dešifrování znaků šifrovaného textu na otevřený text se provádí podle vzorce (7):

(7)

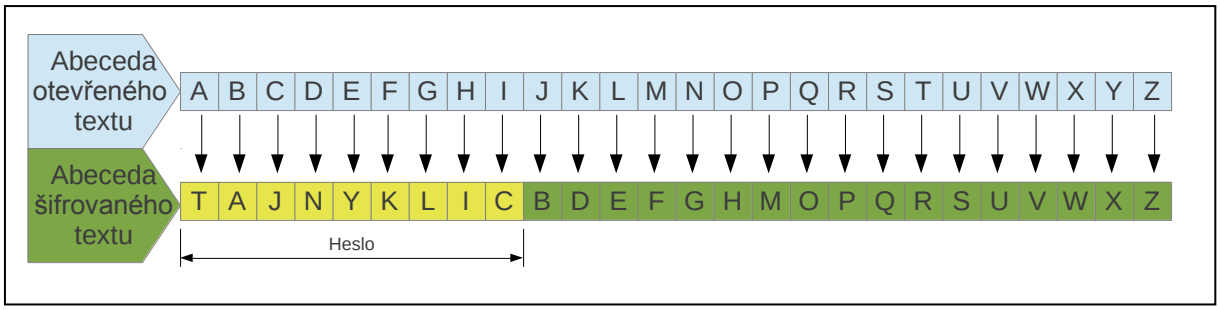
$$P(x) = A^{-1}[C(x) - B] \bmod 26$$

kde A^{-1} – je multiplikativní inverzí čísla A

Pro šifrování afinní šifrou existuje $\varphi(26) \times 26 = 312$ možných kombinací. Útočit na afinní šifru lze jako na kteroukoliv jinou monoalfabetickou šifru frekvenční analýzou. Dále je možný útok hrubou silou.

2.5 Monoalfabetická substituce s klíčem

Heslo, klíč, monoalfabetické substituce musí obsahovat znaky abecedy otevřeného textu. Abeceda šifrovaného textu se vytvoří z abecedy otevřeného textu tím způsobem, že nejprve se na začátek abecedy šifrovaného textu vloží znaky klíče a následně se vloží zbytek abecedy otevřeného textu. Na obrázku je uveden případ, jak by vypadala abeceda šifrovaného textu s klíčem „tajnyklic“. V případě, že by se znaky hesla opakovaly, užité znaky by byly vynechány.



Obrázek č. 6 – Monoalfabetická substituce s klíčem [zdroj: vlastní tvorba]

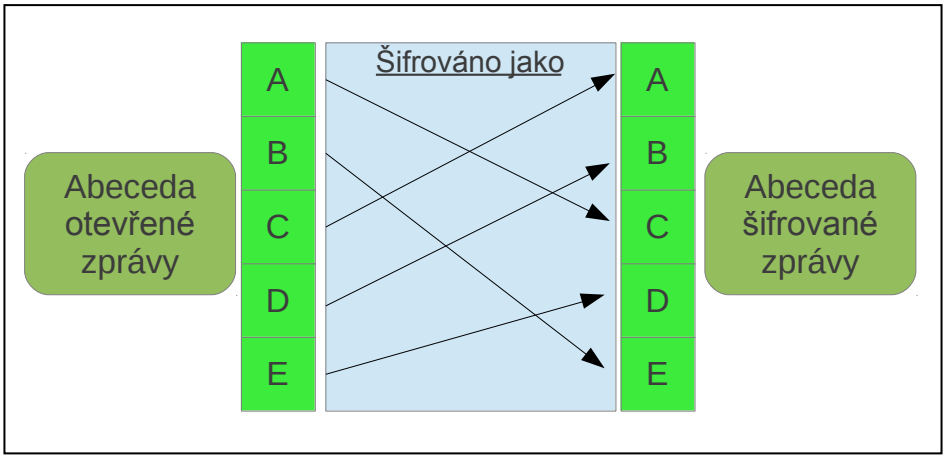
Například bylo-li by heslo monoalfabetické substituce „velmitajneheslo“, klíč k této substituci by byl „velmitajnhslo“.

Monoalfabetickou substituci s klíčem lze při špatně zvoleném klíči prolomit frekvenční analýzou.

2.6 Náhodná transformace

Jak bylo řečeno v úvodu, monoalfabetická substituční šifra může v případě užití latinky nabýt *26!* permutací, což je případ náhodné transformace abecedy otevřeného textu.

Jde o náhodná přiřazení každého znaku abecedy otevřeného textu znaku šifrovaného textu. V případě užití české abecedy s diakritikou, bez rozlišování velkých a malých znaků, jde o *41!* permutací, což je přibližně $3{,}34 * 10^{49}$ možných kombinací. Abeceda otevřeného textu však může být stále ještě rozsáhlejší, protože se do těchto případů nezapočítala čísla, bílé znaky, interpunkční znaménka a jiné symboly.



Obrázek č. 7 – Princip monoalfabetické substituční šifry s náhodnou transformací

[zdroj: vlastní tvorba]

Za zmínku stojí také některé případy, kdy by otevřený text byl psán jinou abecedou než latinkou. Sanskrt, který tvoří 52 znaků, je možné transformovat do 52! možných substitucí. Hebrejskou abecedu o 22 znacích bez znaků „ך ם ן ף ץ”, které se píší jinak na konci slov, a znaků s tečkou jde o 22! možných kombinací. Základní arabská abeceda má 28 znaků, tedy 28! transformací. Japonština abecedu nemá, je zde 204 znaků hiraghana a katakana, které reprezentují celé slabiky, to odpovídá 204! možným substitucím. Dále jsou v japonštině užívány logografické znaky kanji, mezi nimiž jsou i některé čínské znaky a počet znaků kanji činí dle odhadů 70 000 znaků, tedy

přibližně 70 000! permutací. V případě čínské abecedy[11] je ve slovníku Yitizi Zidian z roku 2004 zaznamenáno 106 230 znaků, což znamená 106 230! možných kombinací.

2.7 Vernamova šifra

2.7.1 Krátká historie Vernamovy šifry

Tuto šifru lze za určitých podmínek nazvat „one-time pad cypher“, čili „jednorázově podložená šifra“ či přesněji „šifra s heslem na jedno použití“. Roku 1917 Gilbert Vernam, tehdy pracující u společnosti AT&T Bell Labs[12], objevil tuto šifru, kterou za dodržení všech podmínek nelze prolomit a posléze v roce 1919 patentoval jako americký patent číslo 1310719. Avšak Gilbert Vernam nebyl jejím vynálezcem! Později se ukázalo, že Gilbert Vernam tuto šifru znovuvynalezl, protože již v roce 1882 tuto šifru popsal Frank Miller[13]. Nejstarší Millerova verze z roku 1882 byl popis, jak šifrovat při přenosu zprávy přenášené telegrafem. Vernamova verze z roku 1917 byla vytvořena pro využití šifrování zpráv přenášených dálnopisem, ale tato verze stále ještě nebyla neprolomitelná. **K šifrování byly tehdy používány děrované pásky, které se po vyčerpání záznamu znovu opakovaly ve smyčce.** Joseph Oswald Mauborgne přišel o několik let později se závěrem, že bude-li páska, kterou se otevřený text šifruje, obsahovat zcela náhodný záznam a může být použita pouze jednou. Tím se stal spoluvynálezcem šifry „one-time pad“, která je derivátem Vernamovy šifry.

Claude Shannon[14] v říjnu roku 1949 dokázal, že opravdu neprolomitelná šifra musí mít náhodný klíč stejně dlouhý jako zprávu, tento klíč ani žádná jeho část nesmí být znovu užita a celý klíč je třeba uchovat v tajnosti. Tím vlastně dokázal tvrzení Josepha O. Mauborgne, že za dodržení výše uvedených podmínek bude šifra „one-time pad“ opravdu neprolomitelná.

Upozornění

Často se „Vernamova šifra“ a „one-time pad“ označují jako synonyma, což však není dle názoru autora úplně pravda. První verze Vernamovy šifry počítala s opakováním klíče ve smyčce, až později J. O. Mauborgne, důstojník americké armády, upřesnil, že se záznam opakovat nesmí. Proto autor této práce shledává jako nevhodné uvádět, že Vernamova šifra a one-time pad, která přišla až po nějakém čase, je totéž. Mimo jiné se tímto tématem zabývají studijní materiály na univerzitě Princeton (one-time pad byla vynalezena později, než

---

[11] Více informací o čínské abecedě a čínském jazyce (práce zabývající se efektivnějším a názornějším vyučováním čínské abecedy): http://academiccommons.columbia.edu/download/fedora_content/download/ac:132066/CONTENT/Lu_columbia_0054D_10170.pdf

[12] Oficiální stránky AT&T Labs: http://www.corp.att.com/attlabs/

[13] Více o Franku Millerovi: https://mice.cs.columbia.edu/getTechreport.php?techreportID=1460

[14] Více o Claudu Shannonovi: http://en.wikipedia.org/wiki/Claude_Shannon

Vernamova šifra)[15], rakouská společnost Mils Electronic Ges.m.b.H. & Co. KG[16], zabývající se přenosem šifrovaných zpráv, též uvádí, že šifra one-time pad je pozdější verzí Vernamovy šifry nebo na francouzské Université Paris-Sud se uvádí, že jde o společné dílo Gilberta Vernama a J. O. Mauborgna[17]. Na univerzitě Columbia v New Yorku se dokonce spekuluje o možnosti, že může jít o dílo Franka Millera[18].

Materiálů, zabývajících se touto problematikou, je samozřejmě k dispozici více, zde jsou uvedeny pouze některé pohledy na situaci. Autor se však domnívá, že v každém případě je mezi Vernamovou šifrou a šifrou one-time pad neboli „šifrou s jednorázovým heslem“ rozdíl, nikoliv, jak tvrdí česky psaná Wikipedia[19], že jde o synonyma. Anglicky psaná Wikipedia však uvádí, že zde jsou určité nejasnosti v terminologii a brát Vernamovu šifru a šifru one-time pad jako synonyma není velmi vhodné[20]. **S tímto tvrzením autor práce souhlasí a touto kapitolou zároveň upozorňuje na rozdíl mezi těmito dvěma názvy.**

V praktické části této práce se však bude pracovat s Vernamovou šifrou, neboť vzhledem k šetření strojového času (zde jde především pouze o názornou ukázku) se nebude ověřovat, zda mezi vygenerovaným klíčovým proudem nedochází k opakování hodnot a posloupností těchto hodnot generovaných generátorem pseudonáhodných čísel. O problematice generování náhodných čísel se bude hovořit v následující kapitole.

2.7.2 Popis Vernamovy šifry

Vernamova šifra je šifrou proudovou, kombinující datový tok zprávy a datový tok náhodně či pseudonáhodně generovaného klíče. Klíč je stejně dlouhý jako zpráva a šifra je výsledkem logické operace XOR mezi původní zprávou a klíčem.

Znamená to tedy, že šifrování i dešifrování zprávy probíhá prostřednictvím logické operace XOR, přičemž realizace této operace není pro výpočetní techniku vůbec žádným problémem, z čehož vyplývá, že šifrování i dešifrování zabere minimum strojového času.

---

[15] Rozdíl mezi Vernamovou šifrou a šifrou one-time pad (Princeton University): http://www.princeton.edu/~achaney/tmve/wiki100k/docs/One-time_pad.html

[16] Mils Electronic Ges.m.b.H. & Co. KG – šifra one-time pad jako společné dílo G. Vernama a J. O. Mauborgna, který na G. Vernama navazoval: http://www.mils.com/en/system-data/tabs/one-time-pad-history/

[17] One-time pad vynalezli G. Vernam a J. O. Mauborgne společně: https://www.lri.fr/~fmartignon/documenti/systemesecurite/3-OneTimePad.pdf

[18] Spekulace, zda one-time pad je či není dílo Franka Millera, zatímco se obecně považuje jako společné dílo Gilberta Vernama, který tuto šifru „znovuvynalezl“, a Josepha O. Mauborgna (Columbia University, New York): https://mice.cs.columbia.edu/getTechreport.php?techreportID=1460

[19] Česká verze internetové encyklopedie Wikipedia hovoří o vernamově šifře a o šifře one-time pad, že jde o synonyma (ke dni 9. 8. 2013): http://cs.wikipedia.org/wiki/Vernamova_%C5%A1ifra

[20] Anglicky psaná verze internetové encyklopedie Wikipedia upozorňuje na nepřesnosti v terminologii (ke dni 9. 8. 2013): http://en.wikipedia.org/wiki/One-time_pad

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Tabulka č. 3 – Pravdivostní tabulka logické operace XOR (nonekvivalence)

Platí tedy, že:

$$Otevřený\ text \oplus klíč = šifrovaný\ text$$

a zároveň platí, že:

$$Šifrovaný\ text \oplus klíč = otevřený\ text$$

Jedinou operací, náročnou na strojový čas, je generování náhodného klíče. Generování pseudonáhodných čísel není pro výpočetní techniku problémem, problém nastává ve chvíli, kdy se tato posloupnost pseudonáhodně generovaných čísel může opakovat nebo útočník může zjistit, jaká je posloupnost těchto pseudonáhodně generovaných čísel a postupně tak odhalit klíč. Existují však i algoritmy, které jsou schopny generovat náhodná čísla tak, aby nedocházelo k žádným vztahům, posloupnostem ani k opakováním a tím znemožnily odhalení klíče. V Javě jsou tyto metody volně k dispozici od společnosti Sun Microsystems[21] ve třídě `SecureRandom`, ale v programové části této bakalářské práce nebudou kvůli náročnosti na strojový čas implementovány.

[21] Více o společnosti Sun Microsystems: http://www.oracle.com/us/sun/index.html

3 Polyalfabetické šifry

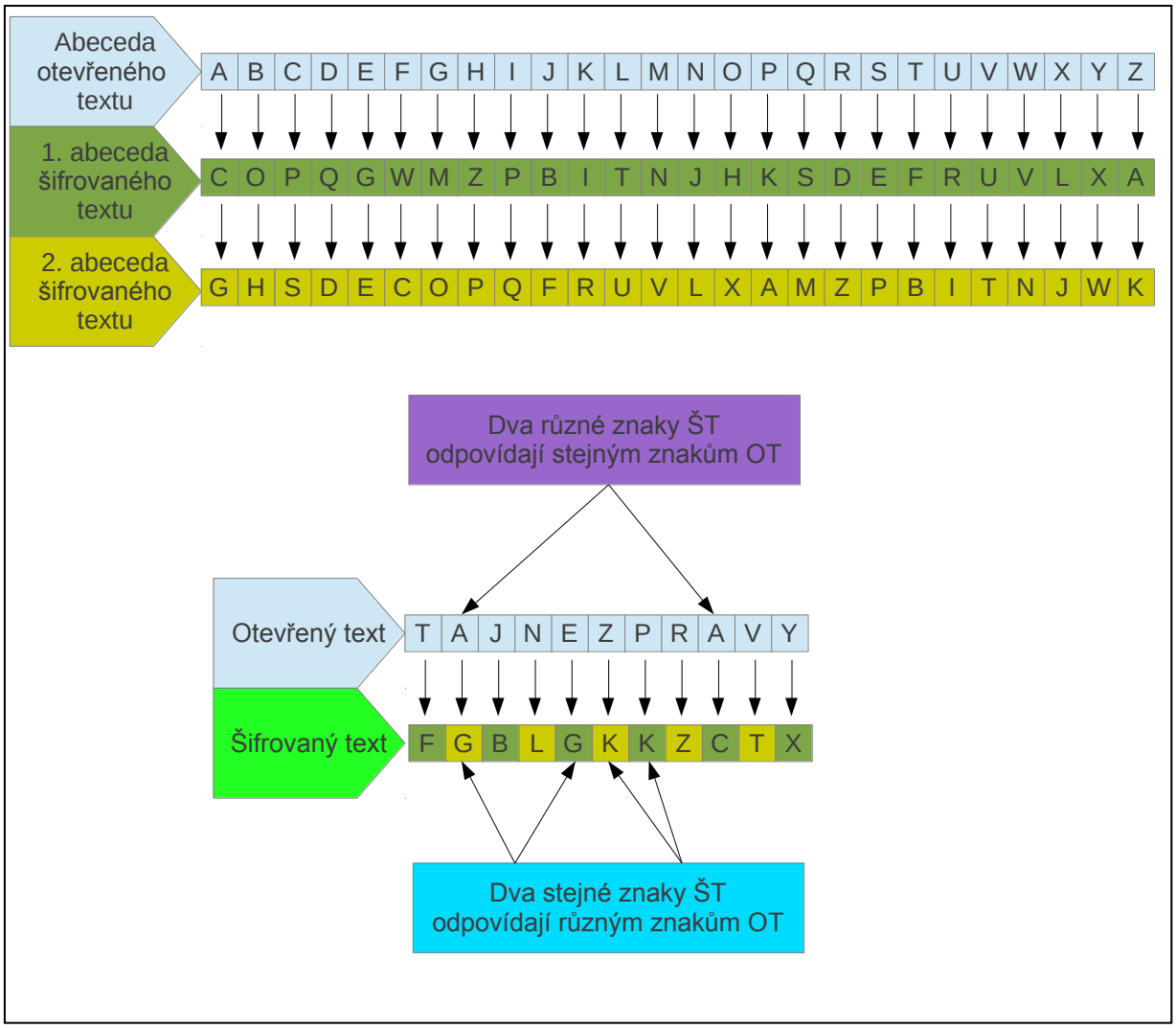
3.1 *Stručný úvod do teorie polyalfabetických šifer*

Polyalfabetické šifry patří mezi jednoduché **symetrické** šifry. Stejně jako monoalfabetické šifry jsou šiframi **substitučními.** Zásadním rozdílem mezi monoalfabetickými a polyalfabetickými šiframi je fakt, že stejné znaky šifrovaného textu mohou odpovídat rozdílným znakům otevřeného textu. Díky tomu jsou polyalfabetické šifry na rozdíl od monoalfabetických více imunní vůči útoku frekvenční analýzou. Rizikem je však útok polyalfabetickou statistikou, kdy útočník zkoumá četnost jednotlivých dvojic (bigramů) a trojic (trigramů) šifrovaného textu. Tento algoritmus útoku je sice náročnější na strojový čas než frekvenční analýza, ale prolomení šifry polyalfabetickou statistikou není nemožné.

3.2 **Albertiho šifra**

S objevem frekvenční analýzy v 9. století na blízkém východě a později v Evropě bylo třeba vymyslet lepší šifry, které budou vůči frekvenční analýze odolnější. Jako první popsal v 15. století polyalfabetickou šifru Jean Battista Alberti, který začal používat dvě a více abeced šifrovaného textu, aby zmátl tehdejší kryptoanalytiky.

Na obrázku č. 8 je znázorněn princip albertiho šifry, budou-li k šifrování použity dvě abecedy šifrovaného textu. K šifrování otevřeného textu prostřednictvím dvou abeced se každý lichý znak otevřeného textu šifruje podle první abecedy šifrovaného textu a každý sudý znak podle druhé abecedy. Jde o střídání abeced šifrovaného textu. V případě, že by byly použity tři abecedy, střídaly by se postupně pro každý znak otevřeného textu. Počet abeced k šifrování není limitován, byl-li by počet použitých abeced šifrovaného textu *n*, vystřídalo by se všech *n* abeced.



Obrázek č. 8 – Albertiho šifra se dvěma abecedami ŠT [zdroj: vlastní tvorba]

3.3 Vigenèrova šifra

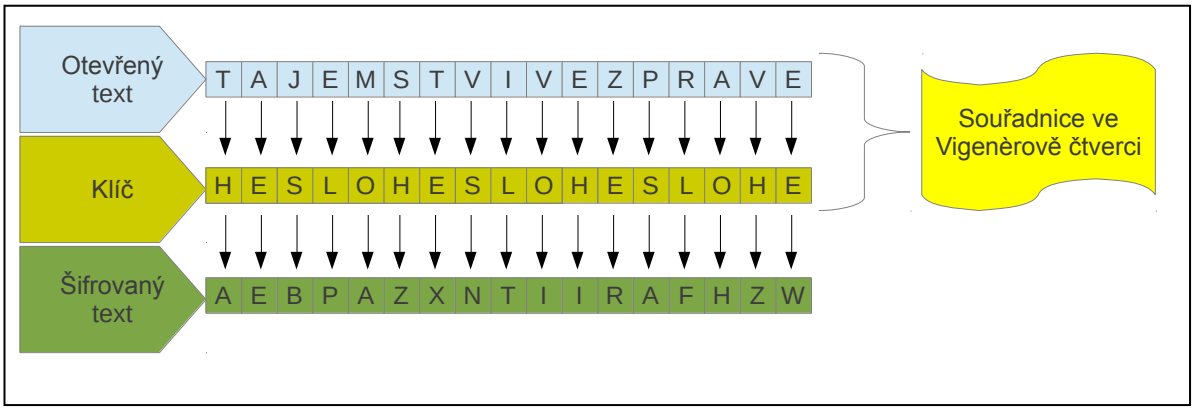
Tato šifra je pojmenována podle příjmení francouzského diplomata Blaise de Vigenère, který ji v 16. století uvedl do této podoby. Ve své době nastudoval díla Leona Battisty Albertiho, Giovanniho Porty a Johanna Trithemia. Ze svých poznatků stvořil tuto, v oné době silnou, šifru. Jeho nápad se bohužel nerozšířil, a přestože tato šifra v sobě uchovávala to nejlepší z prací Albertiho, Porty i Trithemia, nepoužívala se až do 18. století především kvůli své obtížnosti.

Vygenèrova šifra využívá 26 abeced šifrovaného textu. K šifrování otevřeného textu slouží tzv. Vigenèrův čtverec, znázorněný na *obrázku č. 9*. Každý řádek i sloupec je abecedou šifrovaného textu. První řádek i sloupec je abecedou šifrovaného textu odpovídající abecedě otevřeného textu. Každý další řádek i sloupec je abeceda šifrovaného textu z předchozího řádku či sloupce posunuta o jeden znak. Analogicky je pak první řádek (a zároveň sloupec) abecedou otevřeného textu s Caesarovým posunem 1, druhý řádek je abecedou s Caesarovým posunem 2,

až nakonec poslední řádek (poslední šifrovací abeceda) je abeceda otevřeného textu s Caesarovým posunem 25.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Obrázek č. 9 – Vigenèrův čtverec [zdroj: vlastní tvorba]



Obrázek č. 10 – Šifrování Vigenèrovou šifrou dle Vigenèrova čtverce [zdroj: vlastní tvorba]

Dále je třeba zvolit heslo, prostřednictvím kterého se zpráva zašifruje. Klíč se vytvoří tím způsobem, že se heslo bude opakovat dokud nepokryje každý znak otevřeného textu.

Šifrovaný text se vytvoří na základě souřadnic dle znaku otevřeného textu a znaku klíče ve Vigenèrově čtverci.

Vigenèrova šifra je především vysoce odolná vůči frekvenční analýze znaků. Zároveň také tato šifra disponuje velkým množstvím klíčů, libovolných hesel, což výrazně znemožňuje kryptoanalytikům útok hrubou silou.

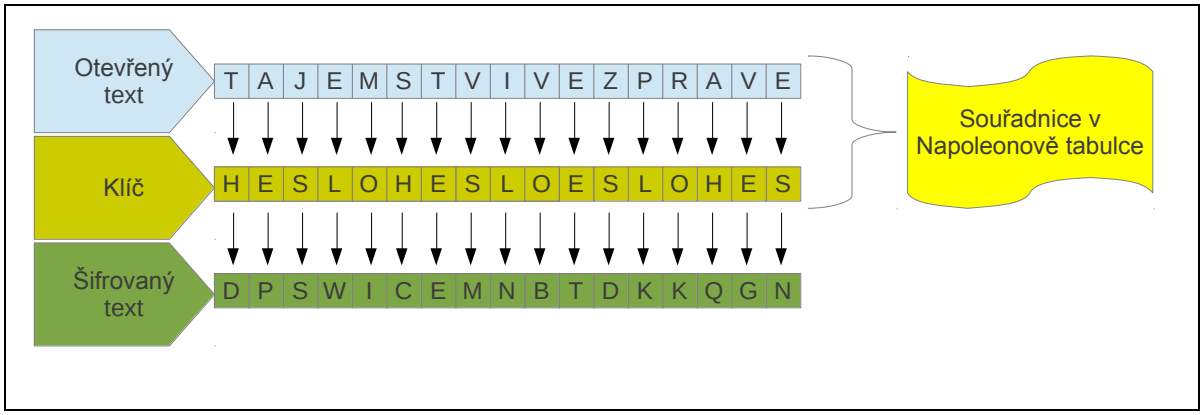
3.4 Napoleonova šifra

Stejně jako římský vojevůdce Gaius Julius Caesar měl i Napoleon Bonaparte svou vlastní techniku šifrování zpráv. Napoleonova šifra je však mnohem složitější na prolomení. Zatímco v Caesarově době nebyla frekvenční analýza známa, Napoleon by si ve své době s prostým posunem znaků v abecedě velmi neuspěl, neboť kryptoanalýza byla již na mnohem vyšší úrovni.

A	A	B	C	D	E	F	G	H	I	J	K	L	M
B	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	A	B	C	D	E	F	G	H	I	J	K	L	M
D	O	P	Q	R	S	T	U	V	W	X	Y	Z	N
E	A	B	C	D	E	F	G	H	I	J	K	L	M
F	P	Q	R	S	T	U	V	W	X	Y	Z	N	O
G	A	B	C	D	E	F	G	H	I	J	K	L	M
H	Q	R	S	T	U	V	W	X	Y	Z	N	O	P
I	A	B	C	D	E	F	G	H	I	J	K	L	M
J	R	S	T	U	V	W	X	Y	Z	N	O	P	Q
K	A	B	C	D	E	F	G	H	I	J	K	L	M
L	S	T	U	V	W	X	Y	Z	N	O	P	Q	R
M	A	B	C	D	E	F	G	H	I	J	K	L	M
N	T	U	V	W	X	Y	Z	N	O	P	Q	R	S
O	A	B	C	D	E	F	G	H	I	J	K	L	M
P	U	V	W	X	Y	Z	N	O	P	Q	R	S	T
Q	A	B	C	D	E	F	G	H	I	J	K	L	M
R	V	W	X	Y	Z	N	O	P	Q	R	S	T	U
S	A	B	C	D	E	F	G	H	I	J	K	L	M
T	W	X	Y	Z	N	O	P	Q	R	S	T	U	V
U	A	B	C	D	E	F	G	H	I	J	K	L	M
V	X	Y	Z	N	O	P	Q	R	S	T	U	V	W
W	A	B	C	D	E	F	G	H	I	J	K	L	M
X	Y	Z	N	O	P	Q	R	S	T	U	V	W	X
Y	A	B	C	D	E	F	G	H	I	J	K	L	M
Z	Z	N	O	P	Q	R	S	T	U	V	W	X	Y

Obrázek č. 11 – Napoleonova tabulka [zdroj: vlastní tvorba]

K Napoleonově šifře je třeba nejprve vytvořit Napoleonovu tabulku, tím způsobem, že pro každý pár znaků abecedy se vypíše šifrovací tabulka. Každý první řádek těchto třinácti šifrovacích tabulek bude posloupností znaků od „A“ do „M“. Druhý řádek první tabulky bude posloupnost znaků od „N“ do „Z“. Každý další druhý řádek bude posunut o jeden znak vlevo. Znamená to tedy, že ve druhé tabulce bude řádek začínat posloupností znaků od „O“ do „Z“ a následovat bude znak „N“. Druhý řádek třetí tabulky bude začínat posloupností „P“ až „Z“ a následovat budou znaky „N“ a „O“. Analogicky se bude postupovat až ke třinácté tabulce, která bude začínat znakem „Z“ a následovat bude posloupnost znaků „N“ až „Y“. Tím je Napoleonova tabulka připravena k šifrování otevřeného textu za pomoci klíče.



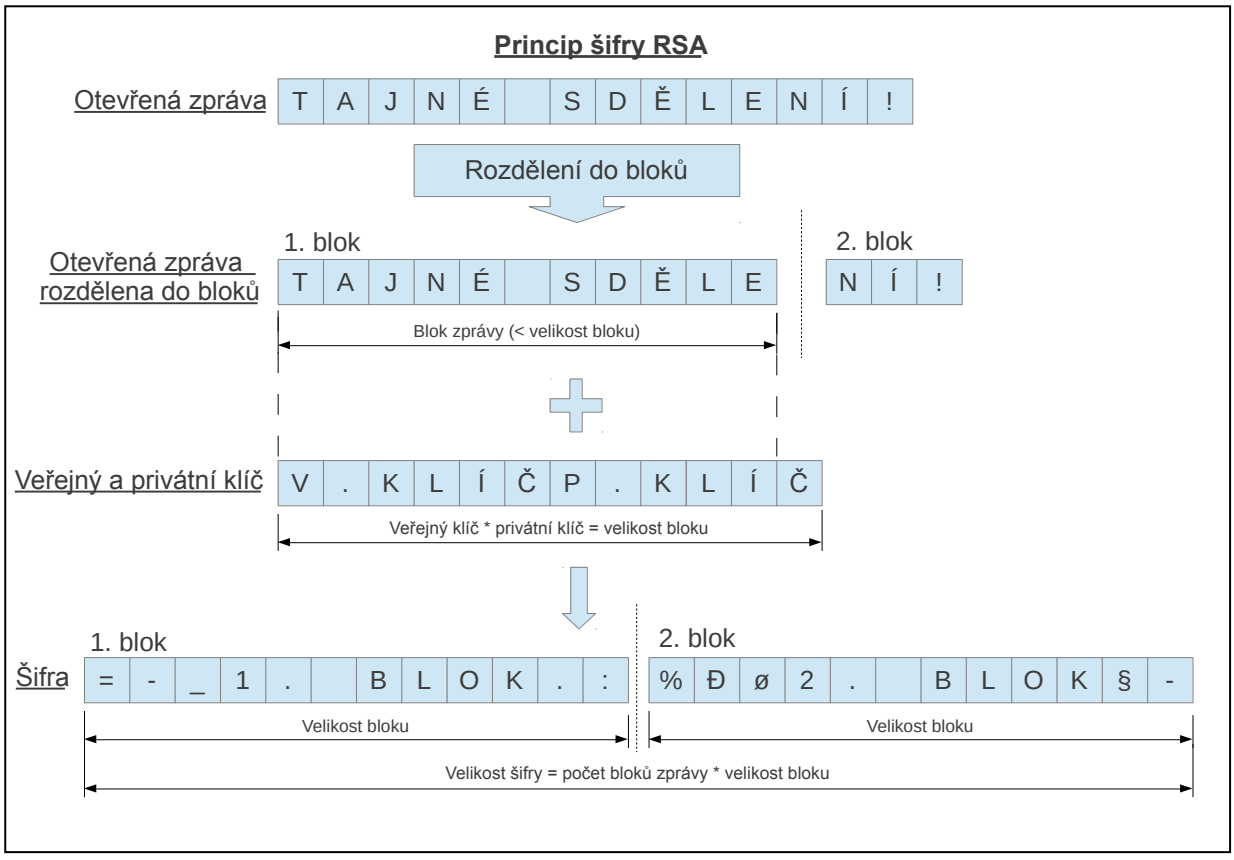
Obrázek č. 12 – Šifrování Napoleonovou šifrou [zdroj: vlastní tvorba]

4 Moderní šifrovací algoritmy

4.1 Asymetrická šifra RSA

4.1.1 Úvod do šifry RSA

Zkratka RSA jsou počáteční písmena jmen pánů Rona Rivesta, Adi Shamira a Leonarda Adlemana, kteří tuto šifru vytvořili již v roce 1977. Šifra RSA je asymetrická. Znamená to, že RSA má dva klíče, jeden k šifrování a druhý k dešifrování. Přestože byl tento algoritmus vytvořen již na konci 70. let, používá se dodnes pro šifrování i elektronické podepisování, protože při dostatečně velkém klíči jej lze považovat i dnes za bezpečný, neboť i dnes je rozložení velkého čísla na součin prvočísel úlohou náročnou na strojový čas, protože zatím není znám algoritmus, který by dokázal za krátký čas rozložit velké číslo. Naopak vynásobení dvou velkých čísel takovým problémem není.



Obrázek č. 13 – Princip šifry RSA obecně [zdroj: vlastní tvorba]

Aby byla data zabezpečená opravdu kvalitně, je třeba mít dostatečně velké klíče, které zajistí obtížnější prolomení této šifry. Některé zdroje dokonce uvádí, že „volba délky klíče RSA šifry je výměnný obchod mezi bezpečností a výkonem“[22]. RSA pracuje se dvěma klíči – s veřejným (public key) a osobním (private key). Veřejný klíč je k dispozici všem, osobní klíč se nikde nesdílí a uchovávají si jej pouze ti, kterým je dovoleno zprávu dešifrovat [31] [32].

Upozornění na šíření mylných informací

Některé zdroje uvádí, že Ron Rivest, Adi Shamir a Leonard Adleman jsou vynálezci asymetrické kryptografie. K objevu došlo již v roce 1973 v britském GCHQ (Government Communications Headquarters)[23], tento objev však podléhal přísnému utajení a byl zveřejněn až v roce 1997, kdy pan Clifford Cocks na konferenci oznámil pravdu o skutečném vývoji[24]. Vzhledem k tomu, že byla tato myšlenka asymetrické kryptografie utajována, šlo tehdy v roce 1977 o znovu-vynalezení. V tomto případě se opět zdroje často neshodují a dochází k šíření mylných informací. Jako příklad lze uvést česky[25] a anglicky[26] psanou internetovou encyklopedii Wikipedia, kde se v české verzi uvádí, že jde o první algoritmus, který je vhodný jak pro šifrování, tak pro podepisování, zatímco v anglické verzi se uvádí, že **pánové Ronald Rivest, Adi Shamir a Leonard Adleman jsou první, kteří tento princip zveřejnili**, zatímco **pan Clifford Cocks z Velké Británie navrhl ekvivalentní systém šifrování již o 5 let dříve**, ale vše podléhalo utajení. I v tomto případě dává autor za pravdu anglicky psané verzi encyklopedie Wikipedia.

4.1.2 RSA matematicky

Nejprve je třeba zvolit dvě prvočísla p a q. Pro lepší zabezpečení by měla být náhodně generována. Dále by měla být rozdílná a podobné bitové délky. Spočítáme jejich modulus jednoduchým vztahem $n = p \times q$, kde součin n je modulus a p a q jsou zmíněná prvočísla. Modulus n je společný pro osobní i veřejný klíč a vyjadřuje délku klíče v bitech.

[22] Zdroj a další informace o RSA v Javě: http://www.javamex.com/tutorials/cryptography/rsa_encryption.shtml

[23] Objev asymetrické kryptografie v britské GCHQ: http://www.gchq-careers.co.uk/about-gchq/history/asymmetric-key-algorithm/

[24] Zveřejnění objevu asymetrické kryptografie: http://www.bristol.ac.uk/pace/graduation/honorary-degrees/hondeg08/cocks.html

[25] Česká verze encyklopedie Wikipedia obsahuje nepravdivé informace, že RSA je první šifrou s veřejným klíčem (ke dni 9. 8. 2013): http://cs.wikipedia.org/wiki/RSA

[26] Anglicky psaná verze encyklopedie Wikipedia správně informuje, že RSA byla první zveřejněnou asymetrickou šifrou, ale k objevu došlo již dříve (ke dni 9. 8. 2013): http://en.wikipedia.org/wiki/RSA_(algorithm)

Nyní se spočítá hodnota Eulerovy funkce (8):

(8)

$$\varphi(n) = (p-1)\times(q-1),$$

kde p, q – zvolená prvočísla, která jsou nesoudělná.

Dále se zvolí takové číslo *e*, pro které platí, že $1 < e < \varphi(n)$ a zároveň nejvyšším společným dělitelem tohoto čísla *e* a čísla $\varphi(n)$ je číslo 1. Číslo *e* se nyní stává exponentem veřejného klíče, mělo by mít krátkou bitovou délku a nízkou Hammingovu zátěž. Hammingova zátěž je počet znaků použité "abecedy" rozdílných od znaku „0" dané „abecedy", např.: Číslo 0101 1110b má Hammingovu zátěž 5 nebo číslo 1000 0000b má Hammingovu zátěž 1. V praxi se nejčastěji volí číslo $2^{16} + 1 = 65.537$, neboť menší čísla by byla méně bezpečná.

Dále bude určeno takové číslo *d*, aby platila rovnice (9):

(9)

$$d \equiv e^{-1}[\mathrm{mod}\ \varphi(n)],$$

kde d – je exponentem soukromého klíče.

Přesněji řečeno $d \times e \equiv 1[\mathrm{mod}\ \varphi(n)]$. Často se číslo *d* počítá rozšířeným Euklidovým algoritmem podle vzorce (10):

(10)

$$a \times x + b \times y = nsd(a,b),$$

kde x, y - splňují požadavky Bézoutovy rovnosti a

$nsd(a, b)$ – je funkce pro hledání nejmenšího společného dělitele čísel, totéž co $gcd(x, y)$.

Bézoutova nerovnost znamená, že největší společný dělitel nenulových různých čísel *a* a *b* lze zapsat lineární kombinací $\alpha \times a + \beta \times b$.

Číslo *x* je převrácenou hodnotou modula *b* a *y* je převrácenou hodnotou modula *a*. Zjištěné číslo *d* bude zachováno jako exponent soukromého klíče.

Prvočísla *p* a *n* spolu s $\varphi(n)$, jsou uloženy a uchovány v tajnosti pro následné výpočty. Nyní jsou připraveny oba klíče. Veřejný klíč, skládající se z modula *n* a veřejného exponentu *e*, soukromý klíč, skládající se z modula *n* a soukromého exponentu *d*.

4.1.3 **Přenos informací prostřednictvím RSA**

Pro navázání spojení a odesílání zprávy se nejprve odešle veřejný klíč z cílové stanice, tj. velké číslo *n* (které vzniklo součinem *p* a *q*) a veřejný exponent *e*. Výchozí stanice chce odeslat zprávu M, ze které nejdříve vytvoří celočíselnou zprávu *m* takovou, aby byla splněna podmínka $0 \leq m < n$. Odesílána bude šifovaná zpráva *c*, kterou výchozí stanice vypočítá vztahem (11):

(11)

$$c \equiv m^e \equiv m^e \pmod{n},$$

kde *c* – je šifrovanou zprávou,
m – je celočíselnou otevřenou zprávou,
n – je součin prvočísel *p* a *q*,
e – je veřejný exponent.

Pro výpočet šifrované zprávy *c* lze využít různé techniky umocnění, které šetří strojový čas (metoda 2^k-tou, metoda sunutí okna, Montgomeryho žebříčková technika, apod.). Vypočítanou zprávu *c* nyní může výchozí stanice odeslat i skrze nezabezpečenou komunikaci.

Cílová stanice obdrží šifrovanou zprávu *c*. Tu dešifruje na původní zprávu *m* výpočtem vztahu (12):

(12)

$$m \equiv c^d \pmod{n},$$

kde *d* – je inverzní hodnotou veřejného exponentu *e*,
m – je celočíselnou otevřenou zprávou,
n – je součin prvočísel *p* a *q*,
c – je celočíselnou otevřenou zprávou.

Původní zprávu *m* z uvedeného vztahu získáme, protože zároveň platí[27] také vzoreček (13):

(13)

$$c^d \equiv \left(m^e\right)d \equiv m^{ed} \pmod{n},$$

kde pro *e*, d – již platí, že $ed \equiv 1 \pmod{p-1}$ a $ed \equiv 1 \pmod{q-1}$.

Malá Fermatova věta tvrdí, že platí $a^{p-1}1 \equiv \pmod{p}$ a zároveň platí $a^p \equiv a \pmod{p}$, tedy číslo, které vznikne výpočtem rozdílu $a^p - a$, je dělitelné číslem *p*. Bude-li se z této věty vycházet, pak platí také vztah (14):

(14)

$$m^{ed} \equiv m^{1+c(p-1)} \equiv m^1\left(m^{p-1}\right)^c \equiv m \times 1^c \equiv m \pmod{p}.$$

[27] Ukázkový příklad algoritmu RSA: http://www.cs.utexas.edu/~mitra/honors/soln.html

Zároveň také platí vztah (15):

(15)

$$m^{ed} \equiv m(\mathrm{mod}\ q).$$

Čínská věta o zbytcích tvrdí, že pokud jsou čísla p a q prvočísly, platí vztah (16):

(16)

$$m^{ed} \equiv m(\mathrm{mod}\ pq).$$

Z toho plyne opět původní vztah (12), tedy po úpravě získáme následující vztah (17):

(17)

$$c^d \equiv m \bmod n.$$

Z Čínské věty o zbytcích vychází kvůli efektivitě mnoho šifrovacích knihoven, využívá je například Java, OpenSuse nebo dotNET (.NET).

4.2 Symetrická šifra RC4

4.2.1 Stručný úvod do šifry RC4

„Rivest Cipher 4“ respektive „Ron's Code 4“ jsou významy zkratky RC4. Jak již zkratka napovídá, jedná se o šifru vytvořenou Ronem Rivestem, který byl mimo jiné spolutvůrcem šifry RSA. Šifra RC4 byla navržena roku 1987 v tehdejší společnosti RSA Data Security, která dnes nese název RSA Security.

Přestože návrh šifry pochází již z roku 1987, šifra RC4 nebyla nikdy publikována oficiálně. Roku 1994 zveřejnil její princip neznámý hacker, který jej získal „rozebráním“ programu BSAFE společnosti RSA Security a následně zveřejnil prostřednictvím automatických e-mailů skupiny Cypherpunks, odkud ji kdosi vystavil na nemoderovaném diskuzním fóru Sci.crypt, zabývajícím se technickými aspekty šifrování, ze kterého byla přejata do mnoha webových stránek a začala se volně šířit po internetu. Problém nastal ve chvíli, kdy bylo třeba tuto šifru pojmenovat. Název „RC4“ byl registrovanou obchodní známkou společnosti RSA Security, proto byla nazvána také zkratkami „ARC4“ či „ARCFOUR“, které znamenají „Alleged RC4“, tedy „Domnělá Rivestova šifra 4“, aby se předešlo problémům s obchodní značkou RC4.

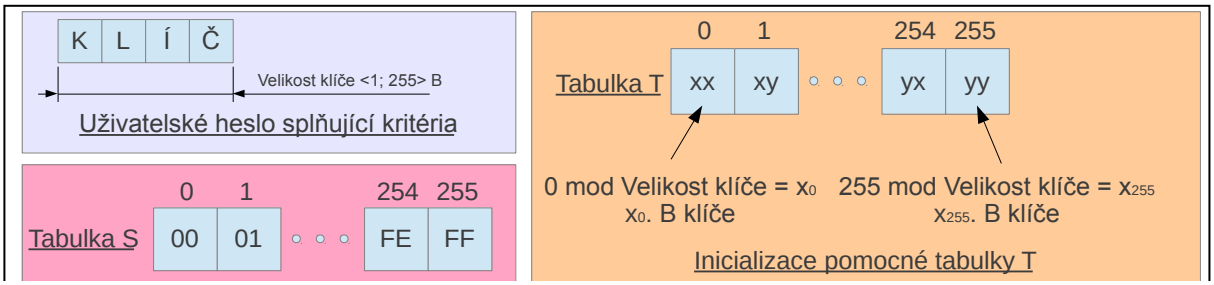
4.2.2 Princip šifry RC4

Šifrovací klíč musí mít délku alespoň 1 bajt a zároveň může být dlouhý maximálně 256 bajtů. Samotný šifrovací klíč zprávu nešifruje, nýbrž slouží k vytvoření tajné substituce, tedy šifrovací tabulky S, ze které se následně generují jednotlivé Byty hesla [38] [39].

4.2.3 Tvorba substituční tabulky

Nechť je délka klíče, hesla zadaného uživatelem, jednou z hodnot otevřeného intervalu $\langle 1;256 \rangle$ bajtů. Inicializují se dvě jednorozměrná pole (resp. jedno dvourozměrné pole o dvou řádkách) o velikosti 256 B, ve kterých se bude uchovávat 256 hodnot o velikosti 1 bajt, z nichž jedno je tabulkou S, která bude sloužit jako klíčový proud, a druhé je pomocnou tabulkou T, která dočasně poslouží k výpočtu tabulky S[28].

Každá buňka tabulky S se naplní vzestupně hodnotami šestnáctkové soustavy od 00h do FFh tak, aby 0. bajt měl hodnotu 00h a každý bajt byl inkrementovanou hodnotou bajtu předchozího.



Obrázek č. 14 – Inicializace tabulky S a pomocné tabulky T [zdroj: vlastní tvorba]

Tabulka T bude naplněna opakujícími se hodnotami bajtů klíče, pro upřesnění dle vzorce (18):

$$n \bmod v = x_n, \tag{18}$$

kde n – je pořadí bajtu tabulky T,

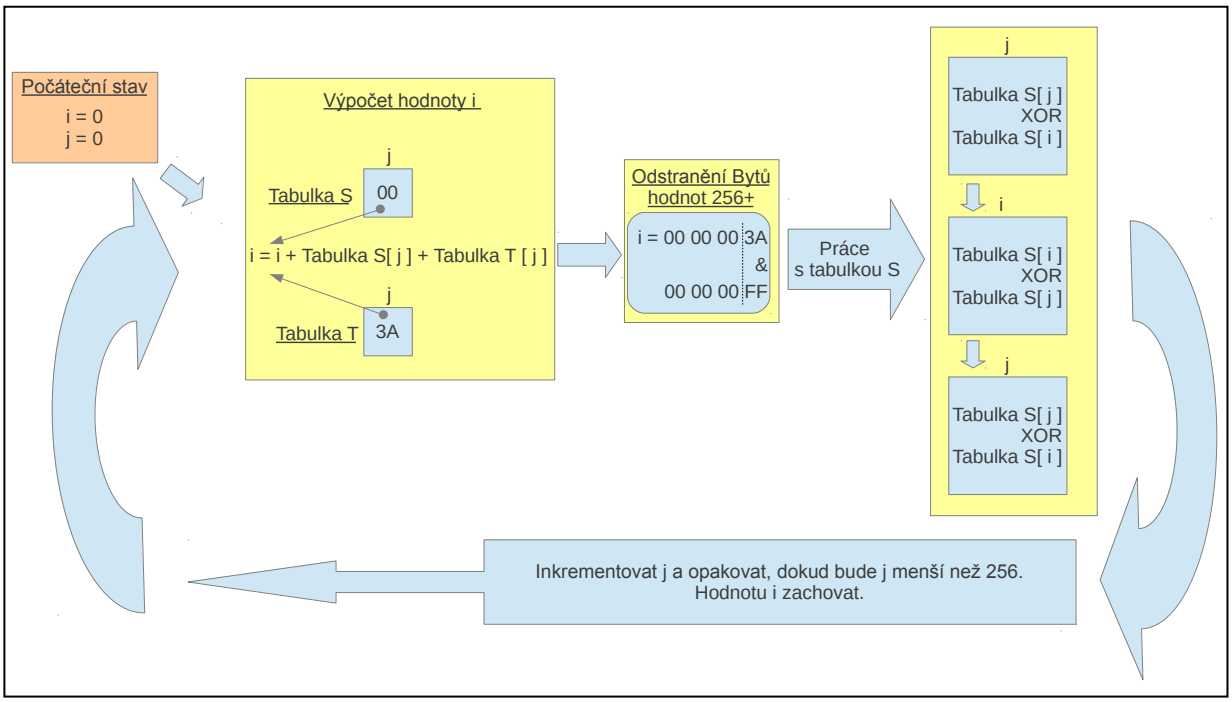
v – je délka klíče (uživatelského hesla) v bajtech a

x_n – je výsledná hodnota, která se přiřadí bajtu na pozici n.

Následuje cyklus, který dokončí tvorbu substituční tabulky S. Inicializují se proměnné *i* a *j* na hodnotu 0, přičemž hodnota proměnné *i* se do každé další smyčky cyklu zachová a hodnota proměnné *j* se před každým dalším cyklem inkrementuje.

[28] Detailnější popis tvorby pomocné tabulky T a substituční tabulky S: http://cse.spsu.edu/afaruque/it6833/RC4.pdf

Nejprve se na začátku smyčky cyklu k proměnné *i* přičte hodnota *j*. bajtu substituční tabulky S a zároveň hodnota *j*. bajtu pomocné tabulky T. Hodnota proměnné *i* nesmí nabývat hodnoty 256 a vyšší. Toho se dosáhne jednoduchou operací AND, logickým součinem, kdy hodnota proměnné *i* bude logicky vynásobena hexadecimální hodnotou, jejíž nejnižší bajt bude mít hodnotu FFh a zbylé bajty budou nulové. Tím se zachová pouze hodnota nejnižšího bajtu hodnoty proměnné *i* a ta se přiřadí proměnné *i*, z čehož plyne, že hodnota proměnné *i* bude nabývat pouze hodnot od 0 do 255.



Obrázek č. 15 – Princip dokončení substituční tabulky S [zdroj: vlastní tvorba]

Za pomocí hodnot proměnných *i* a *j* se postupně přepočítají hodnoty jednotlivých bajtů tak, že *j*. bajtu bude přiřazena výsledná hodnota operace XOR, exkluzivní disjunkce či vylučovací OR, mezi hodnotami bajtů na *i*. a *j*. pozici tabulky S. Dále se *i*. bajtu přiřadí výsledná hodnota exkluzivní disjunkce *i*. a *j*. bajtu tabulky S a nakonec se opět bajtu na pozici *i* tabulky S přiřadí hodnota exkluzivní disjunkce mezi *j*. a *i*. bajtem.

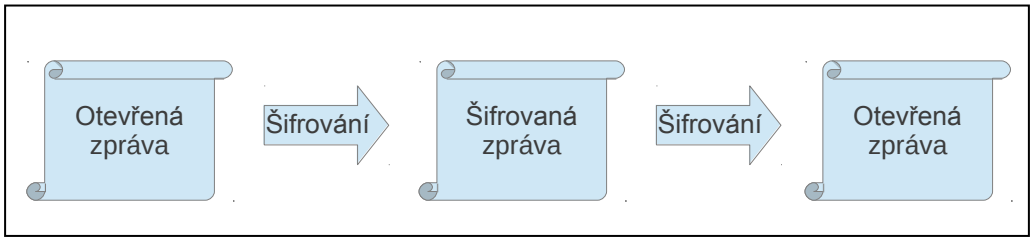
Následuje další smyčka cyklu. Hodnota proměnné *i* zůstává zachována a hodnota proměnné *j* se inkrementuje. Tento cyklus se opakuje, dokud hodnota proměnné *i* je menší než 256, což je počet bajtů tabulky S. Ve chvíli, kdy bude cyklus ukončen, bude připravena substituční tabulka S, která bude sloužit k šifrování i dešifrování zpráv. Tabulku T ani heslo již nebude třeba uchovávat, neboť jejich účel již byl splněn a dále se bude pracovat pouze s tabulkou S [38].

4.2.4 Šifrování a dešifrování prostřednictvím substituční tabulky

Do šifrovacího procesu vstupuje otevřená zpráva o velikosti *n* bajtů. Inicializuje se nové pole o velikosti *n* bajtů, ve kterém bude uložena šifrovaná zpráva. Než bude započat cyklus, který připraví pole s šifrovanou zprávou, je třeba inicializovat 5 pomocných proměnných, které

pomohou s výpočty hodnot jednotlivých bajtů šifrované zprávy. Nechť se inicializují proměnné *i*, *a* a *b* na hodnotu 0 a proměnné *c* a *d*, u kterých není třeba specifická počáteční hodnota. Hodnoty dočasných proměnných *a*, *b*, *c*, *d* a *i* musí zůstat po dokončení každé smyčky cyklu zachovány, neboť slouží k postupným výpočtům šifrované zprávy.

Na počátku každé smyčky bude inkrementována hodnota dočasné proměnné *a* a následně se do ní vloží hodnota jejího logického součinu s hexadecimálním číslem, jehož nejnižší bajt bude nabývat hodnoty FFh a jehož zbylé Byty budou nulové. Tato ošetření lze vypustit v případě, že proměnná *a* bude jednobytová a neznaménková. Stejně tak jde vypustit tato ošetření u kterýchkoliv jednobytových proměnných, jež jsou neznaménkové a mají nabývat pouze hodnot 0 až 255, jde tedy opět pouze o ošetření situace, kdy by hodnota proměnné *a* mohla nabývat hodnot 256 a vyšších. Dále se do proměnné *b* uloží hodnota jejího součtu s *a*. bajtem tabulky S. Též proměnná *b* se ošetří proti hodnotám vyšším 255 buď logickým součinem, nebo modulem 256.



Obrázek č. 16 – Šifrování a dešifrování zpráv prostřednictvím šifry RC4

[zdroj: vlastní tvorba]

Nyní se *a*. bajtu tabulky S přiřadí výsledná hodnota bitového exkluzivního součtu, tedy operace XOR, mezi hodnotami *a*. a *b*. bajtu tabulky S. Dále se *b*. bajtu tabulky S přiřadí výsledná hodnota operace XOR mezi *a*. a *b*. m tabulky S a následně se opět *a*. bajtu tabulky S přiřadí hodnota logického součtu *a*. a *b*. bajtu tabulky S. Pomocné proměnné *c* se přiřadí hodnota součinu (či modula 256) součtu *a*. a *b*. bajtu tabulky S a do proměnné *d* se uloží hodnota *c*. bajtu tabulky S a konečně se do *i*. bajtu šifrované zprávy uloží exkluzivní bitový součet mezi *i*. bajtem otevřené zprávy a proměnnou *d*. Smyčka končí, inkrementuje se hodnota proměnné *i* a cyklus pokračuje, dokud je *i* menší nebo rovno počtu bajtů otevřené zprávy.

Dešifrovací proces šifrované zprávy se realizuje stejným způsobem jako šifrování otevřené zprávy s rozdílem, že vstupují Byty šifrované zprávy a po dokončení algoritmu, při kterém byl užit správný klíč respektive správná substituční tabulka S, je výsledkem otevřená zpráva.

4.2.5 Šifra RC4 v praxi

Přestože název šifry „RC4“ většinou laikovi nic neřekne, jistě bude znát jiné varianty implementace šifry RC4. Pokud dotyčný používá zabezpečenou bezdrátovou Wi-Fi síť, připojuje se ke svému internetovému uživatelskému účtu, odemykal či zamykal heslem PDF dokument

nebo komunikoval přes Skype, již se s modifikací šifry RC4 setkal. V případě, že dotyčný již někdy použil Vzdálenou plochu, setkal se se šifrou RC4 i v RDP protokolu.

4.2.6 Bezdrátové místní sítě

V případě zabezpečení bezdrátové sítě standardu IEEE 802.11 šifrováním WEP se jedná o šifru RC4, kde je klíčem heslo a 24bitový inicializační vektor. Například 256bitového šifrování WEP se jedná o 58 hexadecimálních znaků a 24bitový inicializační vektor, což dohromady dá oněch 256 bitů. Tento klíč se prostřednictvím algoritmu šifry RC4 přepracuje na klíčový proud, zmíněnou substituční tabulku S a exkluzivním bitovým součtem (operací XOR) mezi otevřenou zprávou a klíčovým proudem připraví šifrovanou zprávu, kterou již může odeslat.

Je-li bezdrátová síť zabezpečena šifrováním WPA s šifrovacím protokolem TKIP, využívá šifru RC4 tím způsobem, že se pro každý další paket dynamicky generuje nový 128bitový klíč. Původní verze šifrování WPA s protokolem TKIP, tedy „protokolem se zásadou jednoho klíče“, byla pouze nástavbou šifry WEP.

4.2.7 RC4 a bezpečnost na internetu

Kryptografický protokol TLS, zabezpečení transportní vrstvy, a jeho předchůdce SSL, „vrstva bezpečných socketů“, používají šifru RC4 pro utajení přenášených informací. Šifra RC4 však není jediným algoritmem, který zde vystupuje. K tajné výměně klíčů slouží asymetrická šifra RSA a autentizační kód zpráv algoritmus MAC se stará o celistvost přenášených zpráv. V dubnu roku 2006 došlo k definování standardu TLS 1.1, ve kterém byla zavedena ochrana proti útoku CBC, tedy „zřetězení bloků šifry“ (cipher block chaining), implicitní inicializační vektor byl nahrazen explicitním, tudíž protokol TLS 1.1 se v současnosti zdá býti celkem bezpečným. Přesto bohužel dle průzkumu TIM dodnes až 80 % webových stránek není správně zabezpečených a je zde stále podporována komunikace přes protokoly TLS 1.0, SSL 3.0 a nižší.

5 Implementační část

5.1 Implementace Caesarovy šifry

Caesarova šifra je nejjednodušší šifrou této práce. Třída `CaesarovaSifra` se stará o přepis otevřeného textu na šifrovaný text. Má dvě privátní proměnné `posun` a `dobaPosledníhoSifrovani`. Proměnná posun uchovává vzdálenost každého znaku šifrovaného textu od každého původního znaku v otevřeném textu. Proměnná `dobaPosledníhoSifrování` uchovává strojový čas, který bylo zapotřebí při posledním šifrování. Metoda `sifrovat` se vstupním parametrem typu `String` a výstupním parametrem typu `String` se stará o převod otevřeného textu na šifrovaný text.

```
public String sifrovat(String otevrenyText) {
    long start = System.currentTimeMillis();
    String sifrovanyText = "";

    char c;

    for(int i = 0; i < otevrenyText.length(); i++) {
        c = otevrenyText.charAt(i);
        c += posun;
        sifrovanyText += c;
    }

    dobaPosledniho Sifrovani = System.currentTimeMillis() - start;

    return sifrovanyText;
}
```

Nejprve se nainicializuje proměnná `start` na aktuální systémový čas, která uchovává počáteční čas kódování. Dále se nainicializuje proměnná `sifrovanyText` typu `String` jako prázdný řetězec, která bude uchovávát šifrovaný text a znak `c` typu `char`, který bude sloužit k převodu každého znaku otevřeného textu na znak šifrovaného textu. Ve for-cyklu se pak každý znak otevřeného textu vloží do proměnné `c` a tento znak se následně připojí k řetězci šifrovaného textu `sifrovanyText`. Ve chvíli, kdy je cyklus dokončen, se prostřednictvím metody `currentTimeMillis()` třídy `System` získá aktuální systémový čas a rozdíl hodnoty aktuálního systémového času a hodnoty proměnné start se vloží do privátní proměnné `dobaPosledniho Sifrovani`, aby bylo možné následně zjistit prostřednictvím metody `getDobaPosledniho Sifrovani()`, jak dlouho trval šifrovací proces daného otevřeného textu.

5.2 Implementace monoalfabetické substituční šifry s náhodnou transformací

V případě této šifry je třeba uchovávat, který znak otevřeného textu odpovídá kterému znaku šifrovaného textu. Tyto znaky uchovává třída `MS_MapaZnaku` jako dva řetězce `otevreny`

a `sifrovany`. Každému znaku řetězce `otevreny` na indexu „i“ odpovídá znak v řetězci sifrovany na indexu „i“. Dále tato třída uchovává dvě proměnné typu `boolean proslaZmenou` a `sifrovatBileZnaky`. Proměnná `proslaZmenou` indikuje, zda se během procesu kódování vyskytl jeden či více nenamapovaných znaků, které byly přidán do mapy a mapa tak byla následně znovu zamíchána, aby tyto znaky byly součástí substituce. Tato indikace `proslaZmenou` slouží především kvůli uživatelskému rozhrani, aby mapa znaků nemusela být vykreslována či přepisována po každém procesu šifrování, ale pouze pokud došlo ke změnám. Po každém dotazu na změny v mapě se proměnné `proslaZmenou` přiřadí hodnota `false`, aby tato proměnná indikovala pouze, zda došlo ke změnám od posledního dotazu na stav mapy. Proměnná `sifrovatBileZnaky` odpovídá svému názvu a uchovává informaci, zda do substituce vstupují i bílé znaky nebo zda se bílé znaky otevřeného textu nebudou šifrovat a ponechají se tak v původní podobě do šifrovaného textu. Dále se zde uchovávají proměnné typu `long`, které uchovávají využitý strojový čas na náhodnou transformaci (`dobaMichani`) a dobu šifrování (`dobaSifrovani`).

Bezparametrický konstruktor třídy `MS_MapaZnaku()` vytvoří náhodnou transformaci základních znaků počínaje znakem „!“ (33d) a konče znakem „~“ (126d), což jsou všechny interpunkční znaky, malá i velká písmena, závorky, čísla, operátory a další často využívané znaky. V případě, že se v otevřeném textu vyskytne nenamapovaný znak, vytvoří se s tímto znakem nová substituce.

```
public MS_MapaZnaku() {
    for(char c = 33; c < 127; c++) {
        otevreny+=c;
    }
    zamichat();
}
```

K namapování náhodné transformace slouží veřejná metoda `zamichat()`. Veřejnou metodou je z důvodu, aby v případě potřeby mohl uživatel zamíchat znaky z uživatelského rozhraní.

```
public void zamichat() {
    proslaZmenou = true;
    long start = System.currentTimeMillis();
    List<Character> znaky = new ArrayList<Character>();
    for(char znak: otevreny.toCharArray()) {
        znaky.add(znak);
    }
    StringBuilder pripravaSifry = new StringBuilder(otevreny.length());
    while(!znaky.isEmpty()) {
        int vzitZnak = (int)(Math.random()*znaky.size());
        pripravaSifry.append(znaky.remove(vzitZnak));
    }
    sifrovany = pripravaSifry.toString();
    dobaMichani = System.currentTimeMillis() - start;
}
```

Nejprve se uloží počáteční systémový čas substituce. Vytvoří se instance seznamu znaků typu `char List znaky`. Do seznamu znaky se prostřednictvím cyklu for-each vloží všechny znaky užité v otevřeném textu. Následně se nainicializuje `StringBuilder pripravaSifry` tak, aby byla schopna uchovat stejný počet znaků, jaký je v otevřeném textu. Do této proměnné se

následně vkládají znaky náhodně odebrané ze seznamu znaky. Ve chvíli, kdy je `List znaky` prázdný, se `pripravaSifry` převede na řetězec a přiřadí se znakům šifrovaného textu. Samozřejmě se vypočítá a uloží doba, po jakou tato náhodná transformace probíhala. Vše je připraveno k procesu kódování otevřeného textu.

O zašifrování otevřeného textu se stará metoda `sifrovat`, jejímž vstupním parametrem je řetězec s otevřeným textem a výstupním parametrem je řetězec šifrovaného textu.

```
public String sifrovat(String otevrenyText) {
    String sifrovanyText = "";
    boolean uspesnePrepsano = true;
    long start;
    do{
       start = System.currentTimeMillis();
       uspesnePrepsano = true;

       for(int i = 0; i < otevrenyText.length(); i++) {
           if(!((!sifrovatBileZnaky) &&
                   ((otevrenyText.charAt(i) == '\n') || (otevrenyText.charAt(i) == ' '))) {
            if(otevreny.indexOf(otevrenyText.charAt(i)) == -1) {
                uspesnePrepsano = false;
                novyZnak(otevrenyText.charAt(i));
            }
                sifrovanyText+=sifrovany.charAt(otevreny.indexOf( otevrenyText.charAt(i)));
           } else {
               sifrovanyText+=otevrenyText.charAt(i);
           }
        }
       if(!uspesnePrepsano) {
            sifrovanyText = "";
            zamichat();
        }
    } while(!uspesnePrepsano);
    dobaSifrovani = System.currentTimeMillis() - start;
    return sifrovanyText;
}
```

Nadeklaruje se proměnná `sifrovanyText` typu `String` a nainicializuje se jako prázdný řetězec. Dále se nadeklaruje proměnná typu `boolean uspesnePrepsano` a přiřadí se jí hodnota `true`, která slouží jako indikace stavu, kdy se v otevřeném textu objevil nenamapovaný znak, který způsobí novou náhodnou transformaci abecedy šifrovaného textu. Pokud se takový znak vyskytne, je třeba zašifrovat otevřený text dle nové substituce znaků šifrovaného textu. Celý proces kódování je obalen cyklem do-while, který končí ve chvíli, kdy je otevřený text přepsán dle substituce, která zahrnuje již všechny jeho znaky. Na počátku tohoto do-while cyklu se přiřadí proměnné start hodnota aktuálního systémového času, aby doba procesu kódování nebyla ovlivněna dobou tvorby náhodné transformace, a zároveň se proměnné `uspesnePrepsano` přiřadí hodnota `true`, aby bylo ověřeno, zda právě v aktuální smyčce cyklu do-while došlo k nové substituci.

Následuje for-cyklus, který zajistí přepis každého znaku otevřeného textu na znak šifrovaného textu. Nejprve se musí prověřit, zda se šifrují bílé znaky a zároveň zda aktuální znak není bílým znakem. Pokud se nešifrují bílé znaky a aktuální znak je bílým znakem, ve větvi else se tento znak zkopíruje z otevřeného textu do šifrovaného textu. V opačném následuje ve větvi if další rozhodovací blok if, který se provede v případě, že aktuální znak není namapován. V případě,

že namapován není, je proměnné `uspesnePrepsano` přiřazena hodnota `false`, znak je přidán do substituce, provede se nová náhodná transformace a otevřený text se bude šifrovat dle nové substituce. Pokud znak namapován je, do šifrovaného textu se přiřadí takový znak abecedy šifrovaného textu, který odpovídá aktuálnímu znaku abecedy otevřeného textu v otevřeném textu.

Ve chvíli, kdy je otevřený text úspěšně přepsán na šifrovaný, oba cykly končí. Zaznamená se doba, po jakou šifrování trvalo a metoda vrátí proměnnou `sifrovanyText`.

5.3 Implementace Vernamovy šifry

Vernamova šifra využívá klíče, který je stejně dlouhý, jako otevřený text. Tento klíč je třeba uchovávat, aby bylo možné šifrovanou zprávu znovu dešifrovat. Třída `VernamovaSifra` uchovává klíč (`byte[] klic`) a šifrovaný text (`byte[] sifra`) jako pole `byte`. Dále jsou uchovávány údaje o využitém strojovém čase při poslední tvorbě klíče (`long tvorbaKlice`) a při posledním samotném šifrování (`long dobaSifrování`) jako privátní proměnné, které lze získat metodami `get`.

Nejdůležitější veřejnou metodou třídy `VernamovaSifra` je metoda `zasifrovat`, jejímž vstupním parametrem je otevřený text. Tato metoda nevrací žádný datový typ, pouze připraví pole `byte` s klíčem a šifrovaným textem, které si následně může vyzvednout uživatelské rozhraní pro následné zobrazení. Existuje i možnost, že by tato metoda vracela jednorozměrné pole typu `String`, obsahující dva řetězce s klíčovým proudem a otevřeným textem, nebo s dvourozměrným polem `byte`, které by vracelo byty klíčového proudu a byty šifrované zprávy, ale byla zvolena tato možnost, že si uživatelské rozhraní tato pole vyzvedne až po dokončení procesu kódování.

```
public final void zasifrovat(String otevrenyText) {
    if(otevrenyText != null) {
        byte[] textB = otevrenyText.getBytes();
        generovatKlic(textB);
        sifrovat(textB);
    }
}
```

Nejdříve je třeba zkontrolovat, zda příchozí řetězec s otevřeným textem není `null` (zda je vstupní parametr referencí na existující objekt), pokud je, šifrování neproběhne. V případě, že vstupní řetězec existuje v paměti, je tento řetězec rozdělen do pole `byte` a přiřazen poli `textB`. Toto pole bude následně využito jako vstupní parametr pro privátní metody `generovatKlic` a `sifrovat`.

```
private void generovatKlic(byte[] text) {
    if(klic.length < text.length) {
        this.tvorbaKlice = System.currentTimeMillis();
        klic = new byte[text.length];
        for(int i = 0; i < klic.length; i++) {
            klic[i] = (byte)(Math.random()*256);
        }
        this.tvorbaKlice = (System.currentTimeMillis() - this.tvorbaKlice);
    }
}
```

Vstupním parametrem metody `generovatKlic` je pole `byte text`, ve kterém je uložen řetězec otevřeného textu jako pole `byte`. Toto pole poslouží v této metodě pouze jako informace, jak dlouhý klíč je třeba vygenerovat. Pokud je délka pole s otevřeným textem menší než délka klíče, klíč se generovat nebude a nebude se ani dealokovat, aby zůstal připraven v případě, že se otevřený text opět rozroste. Je-li klíč kratší než otevřený text, vygeneruje se nový klíč pro tento delší otevřený text. Nainicializuje se pole klíče dlouhé jako otevřený text a jednotlivé byty se postupně naplní pseudonáhodnými hodnotami od „0“ do „255“ (resp. „00h“ až „FFh“).

Pozor! Tento algoritmus tvorby klíče nesplňuje podmínky neprolomitelnosti Vernamovy šifry! Byty klíče jsou generovány pseudonáhodně, nikoliv náhodně, z důvodu šetření strojového času. Pro generování náhodných čísel v Javě slouží třída `SecureRandom`, od jejíž implementace bylo upuštěno vzhledem k tomu, že tato práce slouží k účelům poznání šifrovacích algoritmů a uživatelské rozhraní by pak během generování náhodného klíče bylo často zaneprázdněno. V případě využití náhodného, nikoliv pseudonáhodného, klíče by bylo třeba si uchovávat privátní proměnnou typu `SecureRandom` a pro generování náhodného klíče by bylo třeba volat metodu `nextBytes()` instance této třídy, která by do `byte` pole `klic` vložila nový náhodný klíčový proud.

Druhou metodou procesu kódování je jednoduchá privátní metoda `sifrovat`, která převede byty otevřeného textu na byty šifrovaného textu a zaznamená strojový čas během procesu kódování.

```
private void sifrovat(byte[] text) {
    this.dobaSifrovani = System.currentTimeMillis();
    sifra = new byte[text.length];
    for(int i = 0; i < sifra.length; i++) {
        sifra[i] = (byte) (text[i] ^ klic[i]);
    }
    this.dobaSifrovani = (System.currentTimeMillis() - this.dobaSifrovani);
}
```

Do metody `sifrovat` vstupují byty otevřeného textu. Dostatečně dlouhý klíčový proud byl připraven metodou `generovatKlic()` a nyní již zbývá v cyklu provést exkluzivní disjunkci (operace XOR) mezi jednotlivými bity klíče a otevřeného textu. Nejprve se alokuje pole šifrovaného textu `sifra`, do kterého se vloží jednotlivé byty, které jsou výsledkem nonekvivalence mezi všemi bity otevřeného textu a klíčového proudu. Tím je šifrovaný text připraven.

5.4 Implementace šifry RC4

Třída RC4 uchovává jako privátní proměnnou pouze `byte` pole se substituční tabulkou „S“. Tato tabulka obsahuje 256 bytů klíčového proudu a vytvoří se ihned po zavolání konstruktoru, jehož vstupním parametrem klíč (uživatelské heslo) jako pole bajtů.

```
public RC4(byte[] klic) {
    if((klic.length >= 1) && (klic.length <= 256)) {
        int velikostKlice = klic.length;

        byte[] t = new byte[256];

        for(int i = 0; i < 256; i++) {
            s[i] = (byte) i;
            t[i] = klic[i % velikostKlice];
        }

        int i = 0;

        for(int j = 0; j < 256; j++) {
            i += s[j] + t[j];
            i &= 0xFF;

            s[j] ^= s[i];
            s[i] ^= s[j];
            s[j] ^= s[i];
        }
    }
```

Před generováním tabulky „S“ je třeba ověřit, zda klíč splňuje kritéria pro šifrování šifrou RC4, tj. musí mít velikost alespoň jeden byte, ale může být dlouhý maximálně 256 byte. V případě, že vstupující klíč tato kritéria splňuje, připraví se pomocná tabulka „T“ a tabulka „S“. Tabulka „T“ bude mít stejně jako tabulka „S“ velikost 256 B.

Nejprve se v cyklu naplní jednotlivé byty tabulky „T“ opakujícími se byty vstupního klíče, tabulka „S“ se naplní posloupností hodnot „00h“ až „FFh“. Dále se nainicializuje pomocná

celočíselná proměnná „i“ na hodnotu „0“ a proměnná „j“, která bude ve for-cyklu nabývat hodnot „0“ až „255“. Na začátku každé smyčky cyklu se k proměnné „i“ přičte součet „j.“ bytu tabulky „S“ a pomocné tabulky „T“. Nyní se proměnné „i“ přiřadí její bitový součin s hodnotou „FFh“ (použije se pouze nejnižší byte tohoto čísla). Nakonec se bytu tabulky „S“ na pozici „j“ přiřadí hodnota jeho exluzivního součtu s bytem na pozici „i“, bytu na pozici „i“ se přiřadí výsledná hodnota jeho nonekvivalence s bytem na pozici „j“ a bytu na pozici „j“ se přiřadí opět výsledek jeho exkluzivního součtu s bytem na pozici „i“. Po dokončení tohoto for-cyklu je klíčový proud připraven a již není třeba byty hesla ani pomocnou tabulku „T“ dále uchovávat.

```
public byte[] zasifrovat(byte[] text) {
    byte[] sifra = new byte[text.length];
    int tmp_a = 0, tmp_b = 0, tmp_c, tmp_d;
    for(int i = 0; i < text.length; i++) {
        tmp_a++;
        tmp_a &= 0xFF;

        tmp_b += s[tmp_a];
        tmp_b &= 0xFF;

        s[tmp_a] ^= s[tmp_b];
        s[tmp_b] ^= s[tmp_a];
        s[tmp_a] ^= s[tmp_b];

        tmp_c = ((s[tmp_a] + s[tmp_b]) & 0xFF);
        tmp_d = s[tmp_c];

        sifra[i] = (byte)(text[i] ^ tmp_d);
    }
    return sifra;
}
```

Třída RC4 obsahuje dvě metody zasifrovat, z nichž jedna má vstupní parametr typu `String` a druhá má vstupní parametr typu `byte[]`. Popsána zde bude pouze druhá metoda se vstupním parametrem typu `byte[]`, protože metoda se vstupním parametrem `String` volá pouze metodu druhou, které předá vstupní řetězec jako pole bytů a vrátí šifrovaný text jako pole bytů, které vrací tato druhá metoda.

Nainicializuje se pole byte pro šifrovaný text o délce otevřeného textu a čtyři dočasné pomocné proměnné „tmp_a“, „tmp_b“, „tmp_c“ a „tmp_d“. Proměnným „tmp_a“ a „tmp_b“ se přiřadí hodnota „0“. Následuje for-cyklus, jehož smyčka se provede pro každý znak otevřeného textu. Na počátku každé smyčky se inkrementuje hodnota proměnné „tmp_a“ a přiřadí se jí hodnota jejího bitového logického součinu s číslem „FFh“ (hodnota proměnné „tmp_a“ jsou opakující se posloupností hodnot „00h“ až „FFh“). K proměnné „tmp_b“ se přičte hodnota bytu na pozici „tmp_a“ v tabulce „S“ a proměnné „tmp_b“ se též přiřadí hodnota jejího bitového logického součinu s číslem „FFh“ (smí nabývat pouze hodnot „00h“ až „FFh“). Nyní se mění tabulka „S“ tím způsobem, že bytu na pozici „tmp_a“ se přiřadí hodnota bitového exkluzivního součtu mezi bytem na pozici „tmp_a“ a „tmp_b“ tabulky „S“, bytu na pozici „tmp_b“ se přiřadí výsledná hodnota nonekvivalence mezi bytem na pozici „tmp_b“ a „tmp_a“ tabulky „S“ a bytu na pozici „tmp_a“ se přiřadí opět výsledná hodnota operace XOR mezi bytem na pozici „tmp_a“ a „tmp_b“ tabulky „S“. Nyní se pomocné proměnné „tmp_c“ přiřadí hodnota logického bitového součinu mezi číslem „FFh“ a součtem bytů tabulky „S“ na pozici

„tmp_a“ a „tmp_b“ (pomocná proměnná „tmp_c“ smí nabývat pouze hodnot „00h“ až „FFh“). Pomocné proměnné „tmp_d“ se přiřadí hodnota bytu tabulky „S“ na pozici „tmp_c“. Nakonec se do pole bytů šifry sifra na pozici „i“ přiřadí výsledná hodnota nonekvivalence mezi dočasnou pomocnou proměnnou „tmp_d“ a bytem otevřeného textu na pozici „i“.

Až for-cyklus projde všechny znaky otevřeného textu, metoda zasifrovat vrátí pole bytů šifrovaného textu. Dešifrování šifrovaného textu lze uskutečnit také metodou zasifrovat. Přesněji řečeno, znovuzašifrováním šifrovaného textu, za použití správně substituční tabulky „S“, bude výstupem opět otevřený text (symetrické šifrování).

5.5 Implementace asymetrické šifry RSA

Přestože se algoritmus šifry RSA jeví jako nejsložitější z implementovaných šifrovacích algoritmů, jeho implementace není nijak velmi složitá. Díky hotovým třídám z balíčků java.security a javax.crypto se samotná implementace velice zjednodušuje. Po ošetření vyjímek, které mohou nastat, se o generovaní klíčového páru i samotné šifrování postarají již hotové třídy od společností Oracle a Sun Microsystems.

Třída RSA uchovává jako privátní proměnnou maximální velikost bloku maxVelikostBloku. Dále uchovává instanci třídy Cipher sifra, která slouží k šifrování a dešifrování, veřejný a osobní klíč.

```
public RSA(int velikostKlice) {
    try {
        sr = new SecureRandom();

        try {
            sifra = Cipher.getInstance("RSA");
        } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        } catch (NoSuchPaddingException ex) {
            Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        }

        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(velikostKlice, sr);

        KeyPair par = kpg.generateKeyPair();
        osobni = par.getPrivate();
        verejny = par.getPublic();

        maxVelikostBloku = (velikostKlice / 8) - 11;
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Vstupním parametrem konstruktoru je velikost klíče v bitech. Vytvoří se nová instance třídy SecureRandom sc. Dále se vytvoří nová instance třídy Cipher pro šifrování šifrou RSA. Blokem try-catch je samozřejmě potřeba ošetřit výjimky NoSuchPaddingException a NoSuchAlgorithmException pro případ, že by se instance šifry špatně inicializovala. Nyní je třeba vygenerovat klíčový pár. K tomu je zapotřebí vytvořit instanci třídy

`KeyPairGenerator` pro algoritmus RSA. Tento generátor klíčového páru ještě nutné nainicializovat pro zvolenou délku klíče (min. 512 b) a pro danou instanci třídy `SecureRandom` (pro generování opravdu náhodných klíčů). Vytvoří se nová instance třídy `KeyPair` a do ní se vygeneruje klíčový pár prostřednictvím instance třídy `KeyPairGenerator`. Z tohoto klíčového páru se do privátních proměnných třídy RSA uloží veřejný a osobní klíč. I tento proces generování klíčů musí být obalen blokem try-catch, který odchytává výjimku `NoSuchAlgorithmException`, kdyby nebyl v generátoru klíčového páru k dispozici algoritmus pro tvorbu klíčového páru šifry RSA. Zbývá už jen spočítat maximální velikost bloku otevřeného textu vstupujícího do procesu kódování tak, že se vyjádří velikost klíče v bytech a odečte se 11 bytů, protože větší zpráva nemůže do šifrovacího procesu vstoupit.

```
private byte[][] rozblokovat(String zprava) {
    byte[] puvodni = zprava.getbajts();
    if(zprava == null) {
        return null;
    }
    if(!(zprava.length() > 0)) {
        return null;
    }
    int pocetBloku = puvodni.length / maxVelikostBloku;
    pocetBloku++;
    byte[][] blokyZpravy = new byte[pocetBloku][sifra.getBlockSize()];
    for(int i = 0; i < pocetBloku; i++) {
        for(int j = 0; j < sifra.getBlockSize(); j++) {
            blokyZpravy[i][j] = puvodni[(i*sifra.getBlockSize()) + j];
        }
    }
    return blokyZpravy;
}
```

Před samotným šifrováním zprávy je třeba otevřený text rozdělit do bloků o maximální velikosti bloku otevřeného textu šifry RSA pro danou velikost klíče. Vytvoří se tedy dvourozměrné pole, v němž jednotlivé řádky budou byty daného bloku šifrovaného textu.

```
public byte[][] sifrovat(String textZpravy) throws InvalidKeyException {
    try {
        sifra.init(Cipher.ENCRYPT_MODE, verejny, sr);
        byte bloky[][] = rozblokovat(textZpravy);

        if(bloky == null) {
            return null;
        }

        for(int i = 0; i < bloky.length; i++) {
            bloky[i] = sifra.doFinal(bloky[i]);
        }

        return bloky;

    } catch (BadPaddingException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    } catch (IllegalBlockSizeException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    } catch (InvalidKeyException ex) {
```

```
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    }
}
```

Nejprve se inicializuje proměnná `sifra` na šifrovací mód s používaným veřejným klíčem a instancí třídy `SecureRandom`. Otevřený text se prostřednictvím metody `rozblokovat` rozdělí do bloků a pak se pouze zavolá metoda třídy `Cipher doFinal`, jejímž vstupním parametrem je pole bytů otevřeného textu, a ta daný blok otevřeného textu zašifruje. Je třeba zachytit výjimky `BadPaddingException`, `IllegalBlockSizeException` a `InvalidKeyException`, pro případ, že by velikost bloku otevřeného textu byla větší než maximální velikost bloku pro aktuální klíč nebo byl s instancí třídy `SecureRandom` použit špatný klíč.

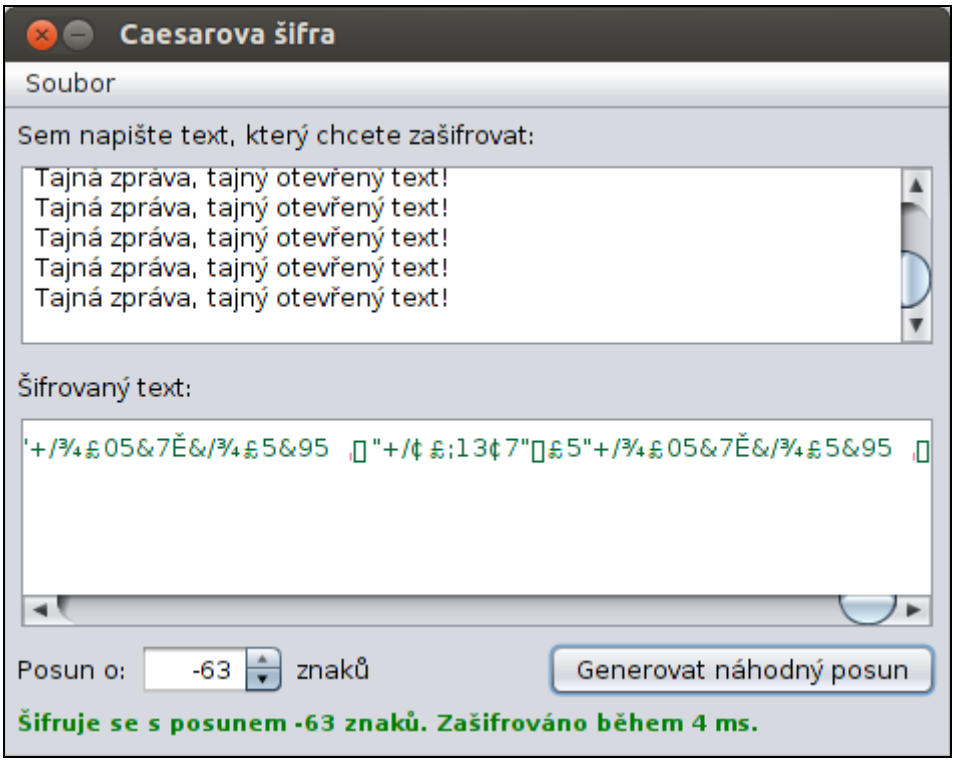
```
public byte[] desifrovat(byte[] sifrovanyText) {
    try {
        sifra.init(Cipher.DECRYPT_MODE, osobni);
    } catch (InvalidKeyException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        return sifra.doFinal(sifrovanyText);
    } catch (IllegalBlockSizeException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    } catch (BadPaddingException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    }
}
```

Otevřený text je šifrován za pomocí veřejného klíče, dešifrování šifrovaného textu je prováděno pomocí osobního klíče (asymetrická kryptografie). Metoda `desifrovat` má vstupní parametr blok šifrovaného textu a výstupem je blok otevřeného textu. Nejprve je třeba inicializovat proměnnou `sifra` třídy `Cipher` na dešifrovací mód s osobním klíčem. V případě, že osobní klíč nebude odpovídat dané šifře, je třeba zachytit vyjímku `InvalidKeyException`. Nyní metoda `desifrovat` vrátí pole bytů otevřeného textu, které je výstupem metody `doFinal`, volané prostřednictvím instance třídy `Cipher sifra`. I zde je třeba zachytit případné výjimky `IllegalBlockSizeException` a `BadPaddingException`.

Pro názornost asymetrického šifrování šifrou RSA v uživatelském rozhraní jsou ve třídě RSA k dispozici veřejné metody, které vrátí řetězce informací o veřejném a osobním klíči voláním metody `toString()` instance daného klíče.

6 Obsluha uživatelského rozhraní

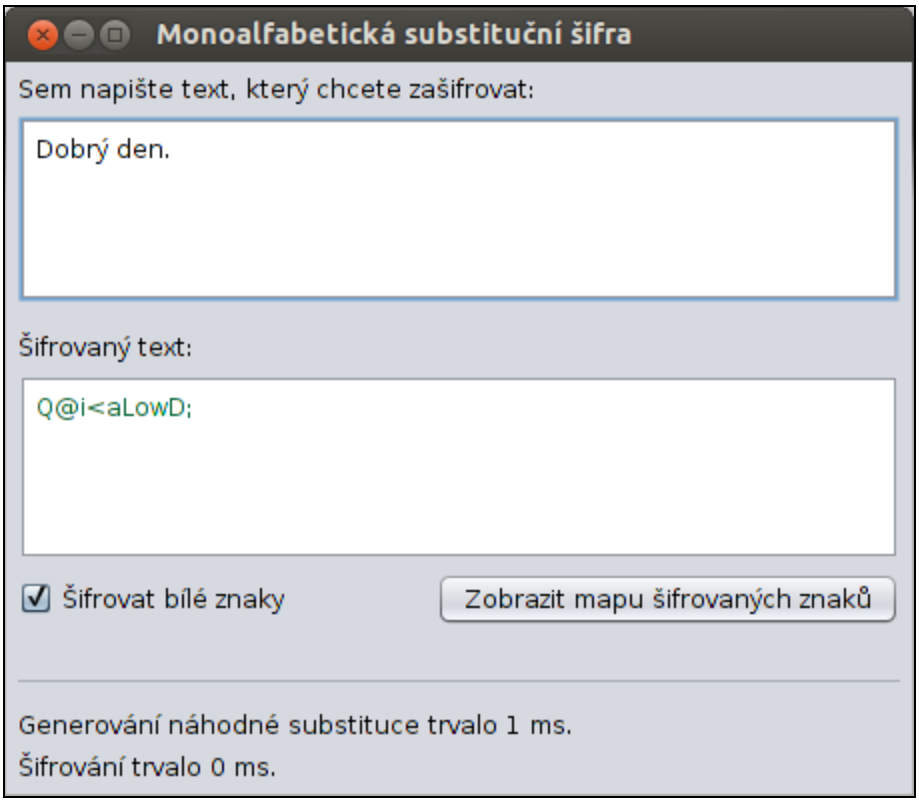
6.1 Uživatelské rozhraní Caesarovy šifry



Obrázek č. 17 – Uživatelské rozhraní Caesarovy šifry [zdroj: vlastní tvorba]

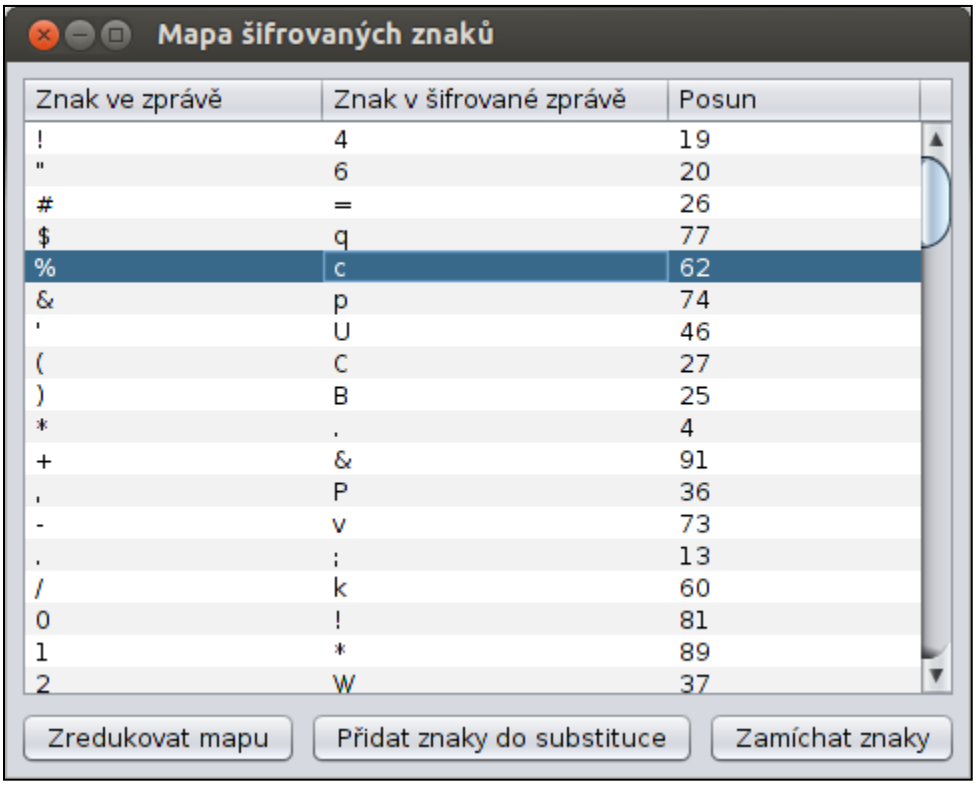
Horní textové pole je určeno k zadání otevřeného textu, který má být zašifrován. Dolní textové pole není editovatelné, ale slouží k zobrazení šifrovaného textu. V levém dolním rohu lze zadat, o kolik znaků budou posunuty znaky šifrovaného textu oproti znakům otevřeného textu. Pomocí tlačítka „Generovat náhodný posun“ od -250 do 249. Prostřednictvím menu „Soubor“ lze šifrovaný text uložit nebo otevřít textový soubor šifrovaného textu. Spodní řádek zobrazuje dobu procesu šifrování (využitý strojový čas). Trvá-li kódování méně než 1 ms, zobrazí se hodnota „0 ms“.

6.2 Uživatelské rozhraní monoalfabetické substituční šifry s náhodnou transformací



Obrázek č. 18 – Uživatelské rozhraní monoalfabetické šifry s náhodnou transformací [zdroj: vlastní tvorba]

Vrchní textové pole je editovatelné a uživatel do něj může vložit či napsat otevřený text, v dolním needitovatelném textovém poli se zobrazuje šifrovaný text. Lze zvolit možnost, zda budou bílé znaky otevřeného textu šifrovány také nebo zda každý bílý znak otevřeného textu bude stejným bílým znakem šifrovaného textu. Ve spodní části okna je zobrazen strojový čas, jakou dobu trvala náhodná transformace a jakou dobu trvalo kódování. Tlačítko „Zobrazit mapu šifrovaných znaků“ vyvolá okno s mapou znaků substituce.



Obrázek č. 19 – Mapa znaků šifrovaného a otevřeného textu

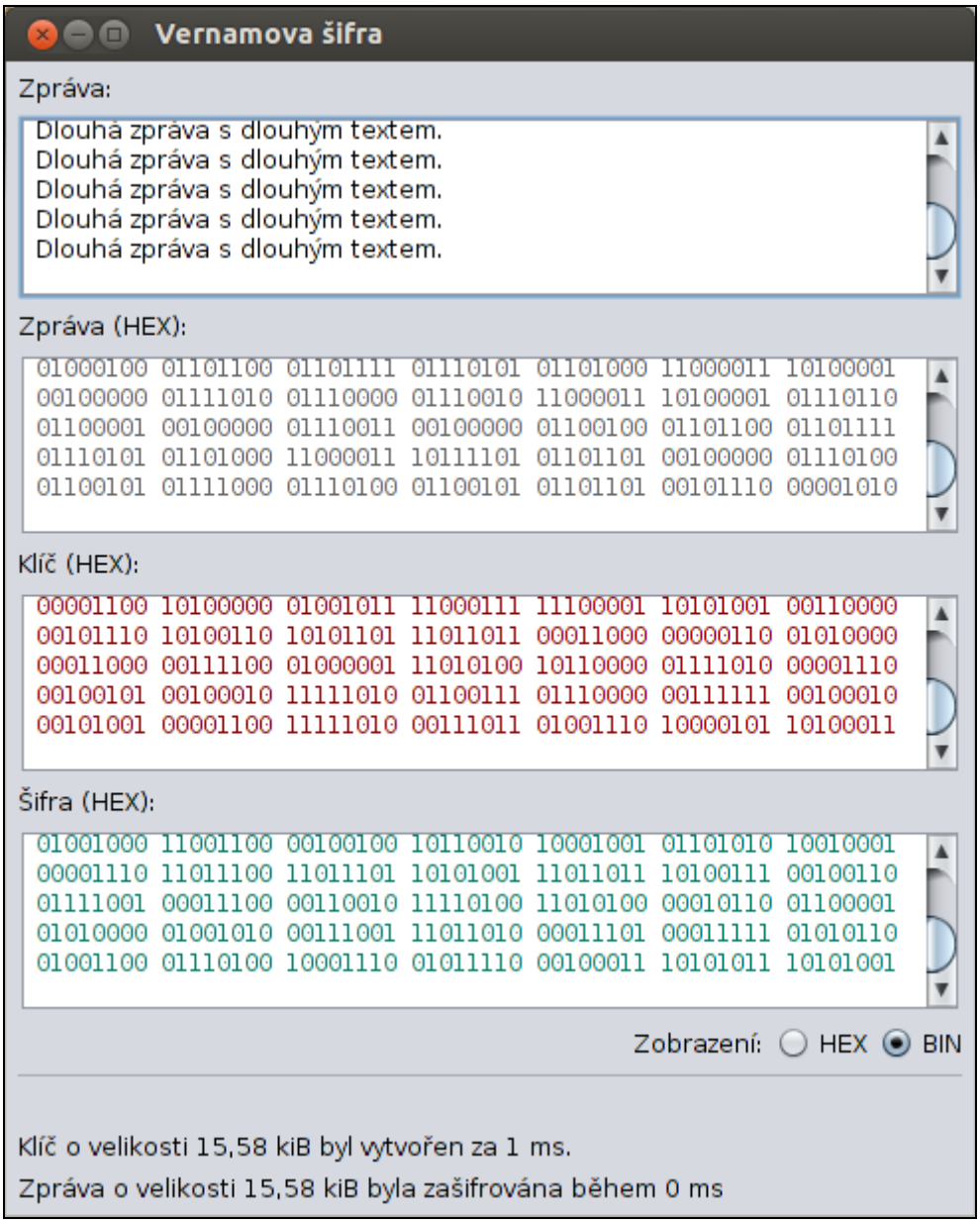
[zdroj: vlastní tvorba]

6.3 Uživatelské rozhraní Vernamovy šifry

Uživatelské rozhraní Vernamovy šifry je nejjednodušším grafickým rozhraním této práce, co se týče obsluhy. Vrchní editovatelné textové pole slouží k vkládání otevřeného textu. Zbylá textová pole slouží pouze k zobrazování. Uživatel nemusí vymýšlet jakékoliv heslo, neboť klíčový proud se generuje sám. Druhé textové pole zobrazuje jednotlivé byty zprávy, ve třetím poli se je automaticky generovaný klíč ke zprávě a ve čtvrtém poli je šifrovaný text.

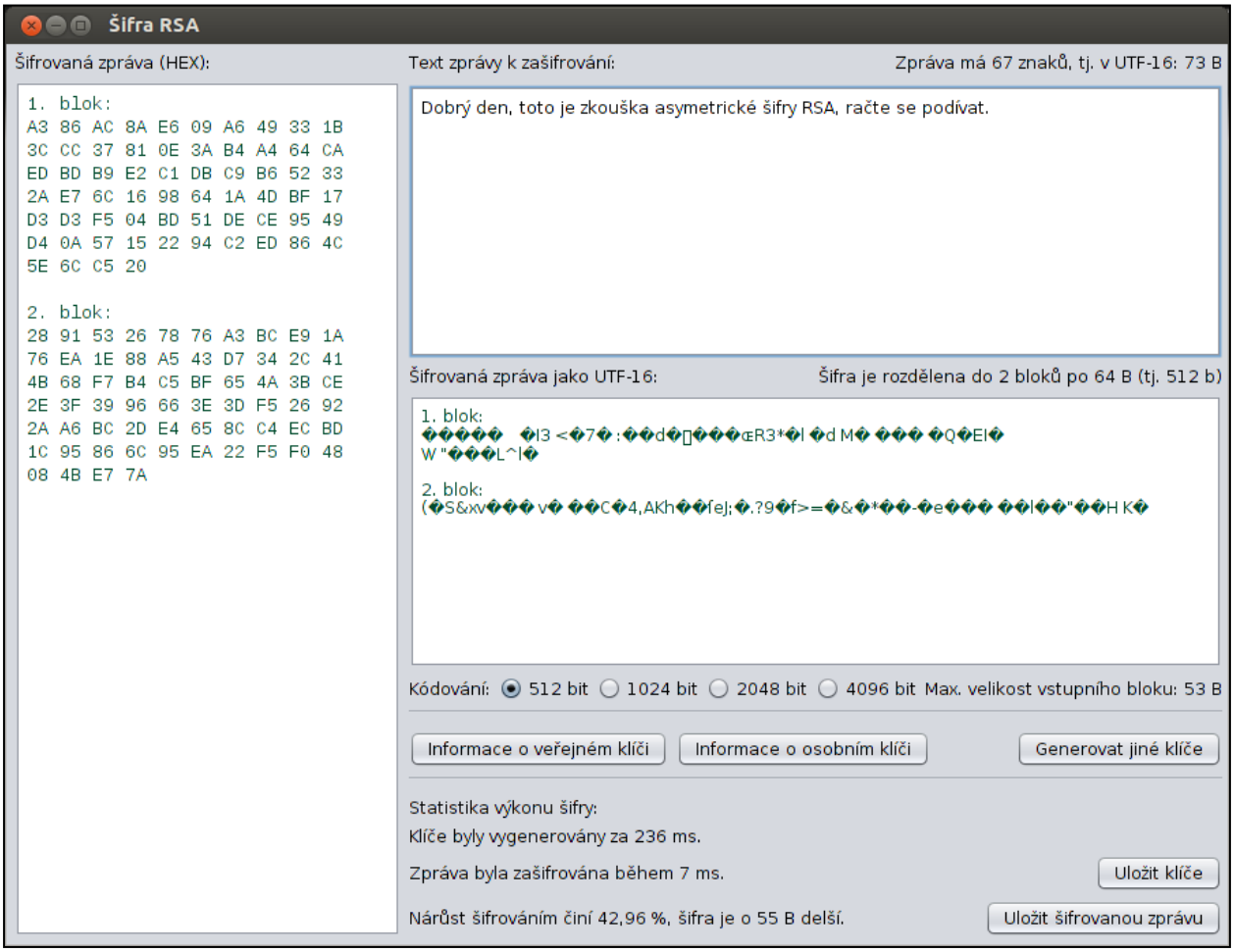
Obrázek č. 20 – Uživatelské rozhraní Vernamovy šifry [zdroj: vlastní tvorba]

K dispozici jsou dvě varianty zobrazení jednotlivých bytů otevřeného textu, klíčového proudu a šifry. Uživatel si může prohlížet texty a klíč ve dvojkové nebo šestnáctkové soustavě, které jsou pro zřejmost algoritmu pravděpodobně nejvhodnější.



Obrázek č. 21 – Vernamova šifra při zobrazení dvojkovou soustavou [zdroj: vlastní tvorba]

6.4 Uživatelské rozhraní šifry RSA



Obrázek č. 22 – Uživatelské rozhraní asymetrické šifry RSA [zdroj: vlastní tvorba]

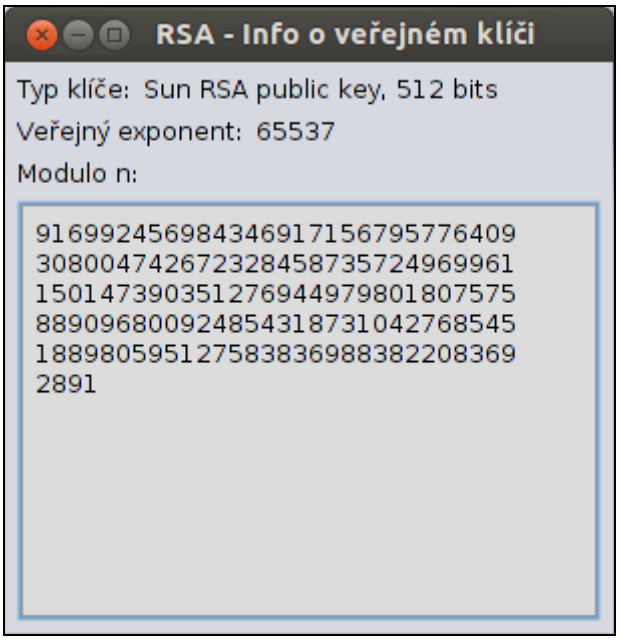
V případě zobrazení bytů otevřeného textu, klíčového proudu a šifrovaného textu ve dvojkové soustavě je princip šifry zřejmý prakticky na první pohled. Šifrovaný text je výsledkem operace nonekvivalence mezi jednotlivými bity klíčového proudu a otevřeného textu.

V okně šifry RSA se vpravo nahoře nachází jediné editovatelné textové pole tohoto okna, do nějž může uživatel vložit či napsat zprávu, kterou chce zašifrovat. Zpráva se během procesu šifrování sama rozdělí do jednotlivých bloků tak, aby splňovala podmínky šifrování. Aby tyto bloky byly zřejmé, okna šifrovaného textu zobrazují šifrovaný text v jednotlivých blocích. Textové pole vlevo znázorňuje bloky šifrovaného textu jako jednotlivé byty v šestnáctkové soustavě. Pro zdůraznění faktu, že každý blok šifrovaného je stejně dlouhý, bylo zvoleno takové písmo, jehož symboly jsou stejné široké, čili neproporcionální. Třetí textové pole, nacházející se pod textovým polem otevřeného textu, zobrazuje jednotlivé bloky jednotlivé znaky šifrovaného textu jako znaky UTF-16.

Pod textovým polem šifrovaného textu (zobrazeného jako UTF-16) jsou k dispozici přepínače, jimiž může uživatel zvolit způsob šifrování. Tyto přepínače umožňují zvolit šifrování v rozsahu od 512 bitů do 4096 bitů, přesněji úroveň zabezpečení šifry v počtu bitů (2^8 b, 2^9 b, 2^{10} b, 2^{11} b).

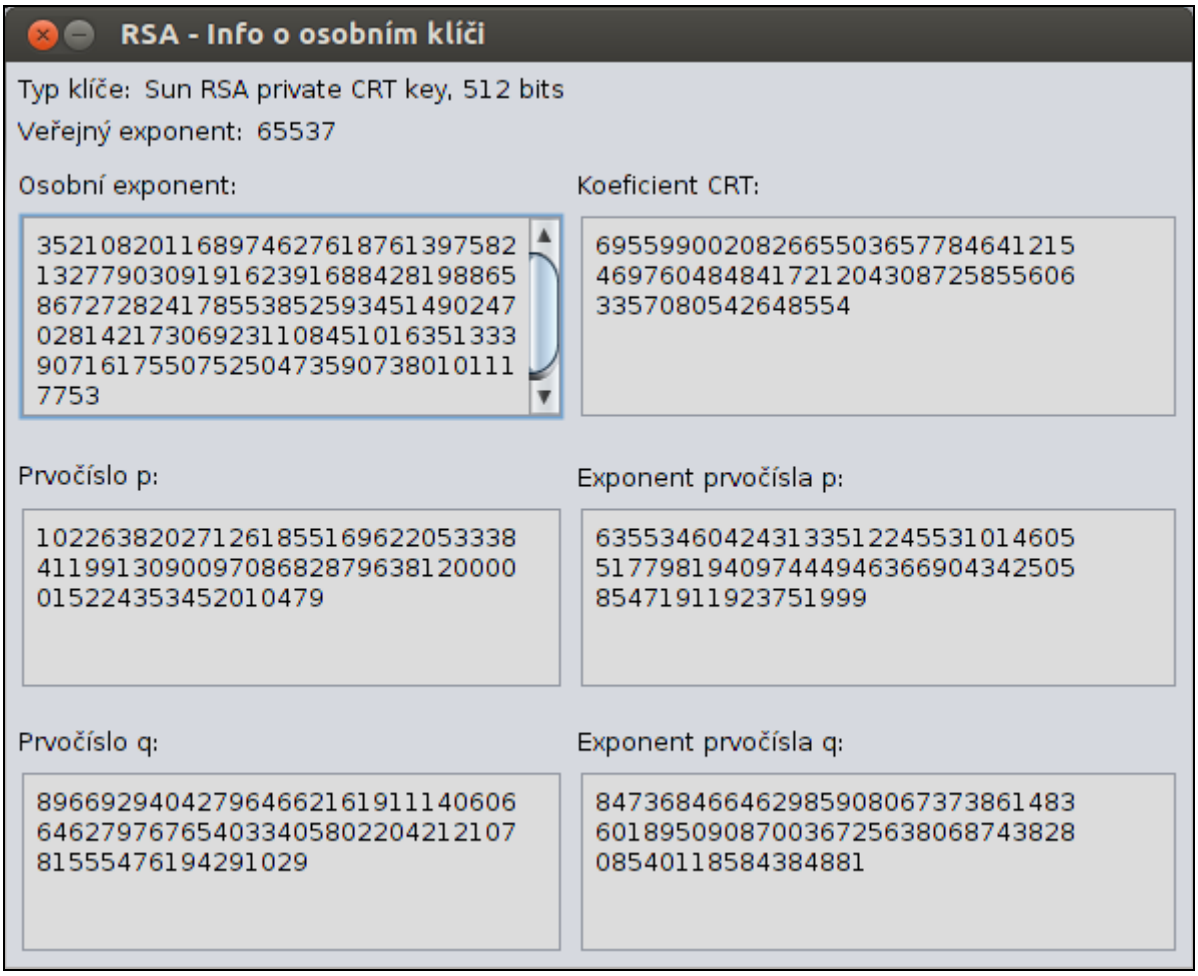
Každé přepnutí přepínačů kódování generuje nový klíčový pár. Tento klíčový pár lze zobrazit prostřednictvím tlačítek „Informace o veřejném klíči“ a „Informace o osobním klíči“. Vpravo od těchto tlačítek se nachází tlačítko „Generovat jiné klíče“, které zavolá konstruktor a vygeneruje se pro šifrování nový soukromý a veřejný klíč. V pravém dolním rohu jsou tlačítka, která umožňují uložit šifrovaný text a k němu klíčový pár.

K oknu šifry RSA patří ještě okna zobrazující klíčový pár, tedy okno s veřejným klíčem a okno se soukromým klíčem.



Obrázek č. 23 – Okno informací o veřejném klíči šifry RSA [zdroj: vlastní tvorba]

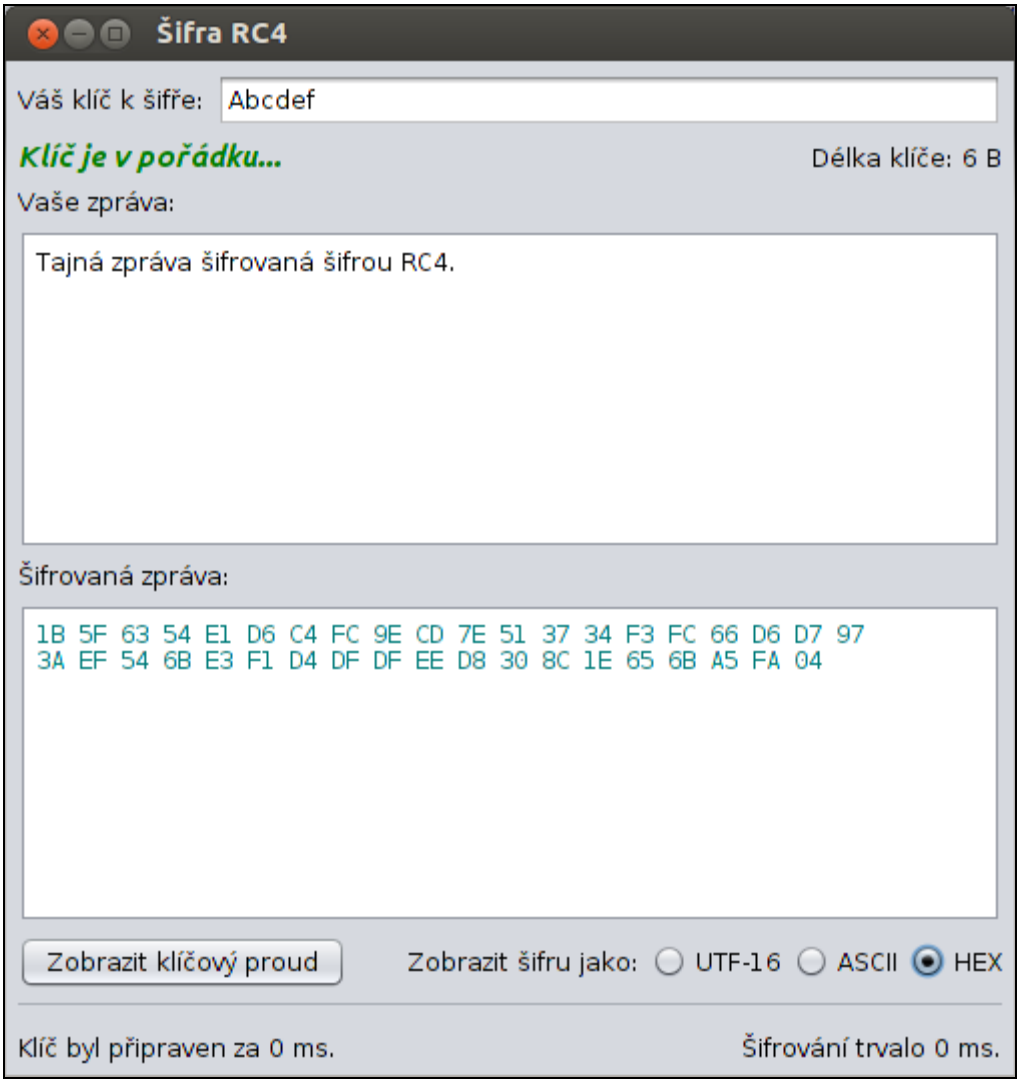
Okno s informacemi o veřejném klíči ukazuje pouze tři informace: o jaký klíč jde, veřejný exponent a vysoké číslo *n* (součin velkých prvočísel *p* a *q*).



Obrázek č. 24 – Okno informací o osobním klíči šifry RSA [zdroj: vlastní tvorba]

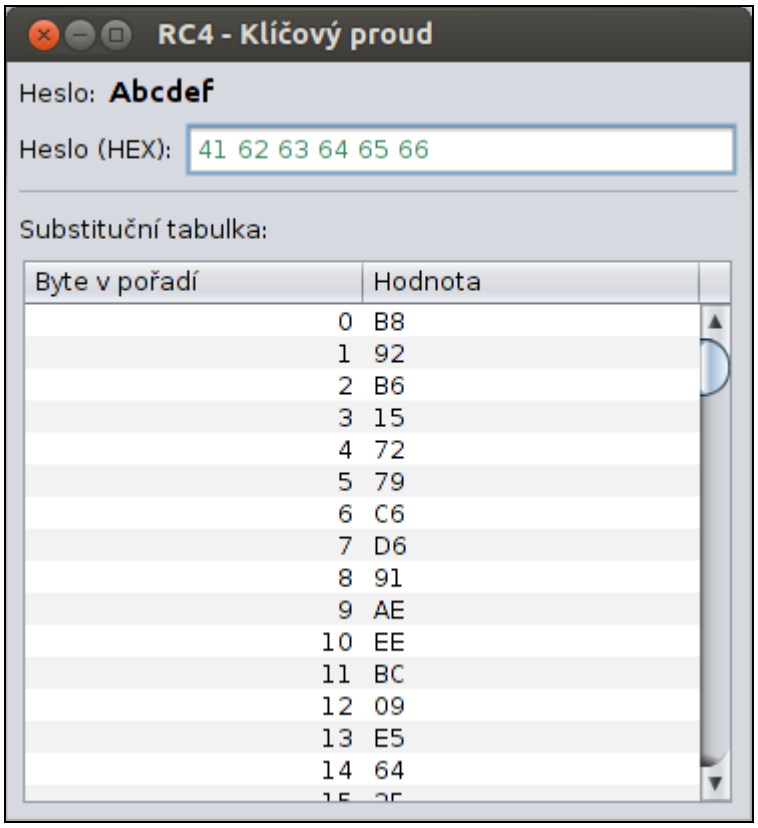
Okno s informacemi o soukromém klíči, zobrazitelné prostřednictvím tlačítka „Informace o osobním klíči“ v hlavním okně šifry RSA, zobrazuje složky osobního klíče, který již veřejně k dispozici není. První popisek uvádí, o jaký typ klíče se jedná a druhý popisek uvádí veřejný exponent, který je užíván k dešifrování. Jednotlivá needitovatelná textová pole zobrazují složky daného soukromého klíče, což je privátní exponent, CRT koeficient, velké prvočíslo *p* s jeho exponentem a druhé velké prvočíslo *q* i jeho exponent.

6.5 Uživatelské rozhraní šifry RC4



Obrázek č. 25 – Uživatelské rozhraní šifry RC4 [zdroj: vlastní tvorba]

Aby se dalo šifrování realizovat, uživatel musí zadat takové heslo, jehož velikost musí být v rozsahu 1 B až 256 B. Po zadání hesla se k šifrování vytvoří klíčový proud. Do vrchního textového pole může uživatel vložit či vepsat otevřený text zprávy, ve spodním textovém poli se bude zobrazovat šifrovaný text. Pomocí přepínačů lze šifrovaný text zobrazit v textovém poli jako znaky UTF-16, ASCII znaky nebo jednotlivé byty v šestnáctkové soustavě. Tlačítko „Zobrazit klíčový proud“ zobrazí substituční tabulku S.



Obrázek č. 26 – Zobrazení substituční tabulky „S“ šifry RC4 [zdroj: vlastní tvorba]

V horní části okna s informacemi o klíčovém proudu je zobrazeno heslo, jak jej uživatel zadal a jeho interpretace v jednotlivých bytech v šestnáctkové soustavě. Zbytek okna je vyplněn tabulkou zobrazující jednotlivé byty tabulky S.

7 Závěr

Monoalfabetické a polyalfabetické šifry jsou v dnešní době dávno minulostí. Vzhledem k rychlosti frekvenční analýzy provedené prostřednictvím výpočetní techniky je téměř jisté, že otevřený text šifrovaný jednou z monoalfabetických či polyalfabetických šifer nemůže být v bezpečí. Ze starších šifer tvoří vyjímku Vernamova šifra, která za předpokladu náhodně vygenerovaného klíče, který bude užit k šifrování pouze jednou, je bez znalosti klíče neprolomitelná. Důkazem neprolomitelnosti této šifry jsou objevené historické texty, ke kterým není k dispozici klíč a jejich obsah tak pravděpodobně zůstane navždy neznámý. V praxi se však Vernamova nevyužije, neboť generování náhodného, nikoliv pseudonáhodného, klíče je náročné na strojový čas a klíč ke zprávě zabere stejný objem dat, jako otevřený text sám. Navíc zde vzniká problém bezpečného přenosu klíče.

Moderní symetrické šifry bývají bohužel velmi často zranitelné, zato velmi rychlé a nenáročné na strojový čas. Moderní asymetrické šifry by byly snadno zranitelné pouze tehdy, pokud by byl lidstvu znám algoritmus, který by velká čísla rozkládal na prvočinitele v polynomiální rychlosti. Asymetrické šifry jsou velmi silné, avšak výpočet klíčů a šifrování s nimi je náročné na strojový čas. Proto se v praxi dospělo ke kompromisu, kdy data pro přenos jsou šifrována symetrickou šifrou, ale výměna klíčů k symetrické šifře probíhá prostřednictvím asymetrické šifry. V současnosti je to nejlepší možné řešení, které zajistí bezpečný a rychlý přenos dat. Budoucnost však může být jiná a nelze vyloučit, že za několik let bude tento trend překonán a rychlé bezpečné šifrování dat bude řešeno jiným způsobem a zcela odlišnými algoritmy.

Tato bakalářská práce splňuje zadání po praktické i teoretické části. Teoretická část se zabývala informacemi o vývoji šifrovacích algoritmů od starověkých šifer až po vybrané současné šifrovací algoritmy a to včetně popisu jejich algoritmů i zranitelnosti těchto šifer. V praktické části byli implementováni zástupci šifer opět od starověku po současnost, s důrazem na názornost provedení těchto algoritmů, i s možností vyzkoušet a pokusit se prolomit starší šifry. Uživatel může zjistit, jakým způsobem lze zabezpečit data a navazovat na tuto práci by neměl být problém.

8 Literatura

8.1 Knižní zdroje

[1] BISHOP, David. Introduction to cryptography with Java™ applets. Vyd. 1. Sudbury, MA: Jones and Bartlett Publishers, Inc. ISBN 0-7637-2207-3.

[2] SCHNEIER, Bruce. Applied cryptography: protocols, algorithms and source code in C. Vyd. 2. New York: John Wiley, 1996, 758 s. ISBN 04-711-1709-9.

[3] ZELENKA, Josef. Ochrana dat: kryptologie. Vyd. 1. Hradec Králové: Gaudeamus, 2003, 198 s. ISBN 80-704-1737-4.

8.2 Akademické zdroje

[4] HURAJ, Ladislav. Nebojme sa šifrovania. Bratislava, 2002. Dostupné z: Studijní opory. Metodicko-matematické centrum v Bratislave.

[5] KLÍMA, Vlastimil. Základy moderní kryptologie – Symetrická kryptografie II. Dostupné z: Studijní opory.

[6] PETRÍK, Radoslav, Zuzana PRIŠČÁKOVÁ a Alena SAMUELOVÁ. Úvod do kryptografie. Nitra, 2010. Dostupné z: Semestrální práce. Univerzita Konštantína Filozófa v Nitre – Fakulta prírodných vied.

[7] KLIMEŠ, Cyril. Kurz: Základy kryptografie pro učitele.

[8] VANĚK, Tomáš. Monoalfabetické substituční šifry. Praha, 2006. Dostupné z: Studijní opory.

8.3 Internetové zdroje

[9] Caesarova šifra - Algoritmy.net. [online]. [cit. 2013-05-03]. Dostupné z: http://www.algoritmy.net/article/34/Caesarova-sifra

[10] How many alphabets are there in Sanskrit. [online]. [cit. 2013-05-03]. Dostupné z: http://wiki.answers.com/Q/How_many_alphabets_are_there_in_Sanskrit

[11] How many letters are in the Hebrew alphabet. [online]. [cit. 2013-05-03]. Dostupné z: http://wiki.answers.com/Q/How_many_letters_are_in_the_Hebrew_alphabet

[12] How many letters are there in the Arabic alphabet? - Yahoo! UK & Ireland Answers. [online]. [cit. 2013-05-03]. Dostupné z: http://uk.answers.yahoo.com/question/index?qid=20110723070125AAsjO2F

[13] How many letters does the Japanese alphabet have. [online]. [cit. 2013-05-03]. Dostupné z: http://wiki.answers.com/Q/How_many_letters_does_the_Japanese_alphabet_have

[14] Chinese characters - Wikipedia, the free encyclopedia. [online]. [cit. 2013-05-03]. Dostupné z: http://en.wikipedia.org/wiki/Chinese_characters

[15] Univerzita Karlova – Antonin Jancarik. JANČAŘÍK, Antonín [online]. [cit. 2013-05-03]. Dostupné z: http://class.pedf.cuni.cz/Jancarik/DesktopDefault.aspx?tabindex=5&tabid=26&PrvekID=260&portalsekce=2&KategorieID=101&Nezobrazovat=ano

[16] RSA (algorithm) - Wikipedia, the free encyclopedia. [online]. [cit. 2013-05-03]. Dostupné z: http://en.wikipedia.org/wiki/RSA_(algorithm)

[17] A5/1 - A GSM stream cipher algorithm. YAKUT, İbrahim. [online]. *ANADOLU ÜNİVERSİTESİ*, Turecko. [cit. 2013-05-04]. Dostupné z: http://home.anadolu.edu.tr/~nat/A5_1.ppt

[18] Cryptanalysis of the Bluetooth E0 Cipher using OBDD's. SHAKED, Yaniv a Avishai WOOL. *SCHOOL OF ELECTRICAL ENGINEERING SYSTEMS, Tel Aviv University*. [online]. Tel Aviv, Izrael, 2006 [cit. 2013-05-04]. Dostupné z: http://eprint.iacr.org/2006/072.pdf

[19] SNOW 3G Stream Cipher Operation and Complexity Study. ORHANOU, Ghizlane, Said EL HAJJI a Youssef BENTALEB. *LABORATOIRE MATHEMATIQUES, Informatique et Applications, Universite Mohammed V Agdal, Faculté des Sciences*. [online]. Rabat, Maroko, 2010 [cit. 2013-05-04]. Dostupné z: http://www.m-hikari.com/ces/ces2010/ces1-4-2010/orhanouCES1-4-2010.pdf

[20] Ultra Encryption Algorithm (UEA): Bit level Symmetric key Cryptosystem with Randomized Bits and Feedback Mechanism. SATYAKI, Roy, Agarwal SHALABH, Nath ŁASOKE, Maitra NAVAJIT a Nath JOYSHREE. *INTERNATIONAL JOURNAL OF COMPUTER APPLICATIONS*. [online]. St. Xavier's College, Kolkata, Indie, Calcutta University, Kolkata, Indie, 2012 [cit. 2013-05-04]. Dostupné z: http://research.ijcaonline.org/volume49/number5/pxc3880687.pdf

[21] Blowfish encryption. [online]. [cit. 2013-05-04]. Dostupné z: http://www.splashdata.com/splashid/blowfish.htm

[22] Horst Feistel - Wikipedia, the free encyclopedia. [online]. [cit. 2013-05-04]. Dostupné z: http://en.wikipedia.org/wiki/Horst_Feistel

[23] Triple DES cryptography software. *D.I. Management Services Pty Limited* [online]. Sydney, Austrálie, 2009 [cit. 2013-05-04]. Dostupné z: http://www.cryptosys.net/3des.html

[24] Lucifer, A Cryptographic Algorithm. SORKIN, Arthur. *LAWRENCE LIVERMORE NATIONAL LABORATORY*. [online]. Livermore, CA, USA, 1983 [cit. 2013-05-04]. Dostupné z: http://fuseki.com/lucifer.pdf

[25] DATA ENCRYPTION STANDARD (DES). DALEY, William M. a Raymond G. KAMMER. *U. S. DEPARTMENT OF COMMERCE/NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*. [online]. Gaithersburg, MD, USA, 1999 [cit. 2013-05-04]. Dostupné z: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[26] ADVANCED ENCRYPTION STANDARD (AES). *NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*. [online]. 2001 [cit. 2013-05-04]. Dostupné z:

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[27] Vernam Cipher, a perfect cipher. ROSENBERG, Burton. *UNIVERSITY OF MIAMI, Department of Computer Science*. [online]. Miami, FL, USA, 2004 [cit. 2013-05-04]. Dostupné z: http://www.cs.miami.edu/~burt/learning/Csc609.051/notes/02.html

[28] The Alberti Cipher. SERVOS, William. *TRINITY COLLEGE, Department of Computer Science*. [online]. Hartford, Connecticut, USA, 2006 [cit. 2013-05-04]. Dostupné z: http://www.cs.trincoll.edu/~crypto/historical/alberti.html

[29] Napoleonova šifra. ŠTRÁFELDA, Jan. [online]. 2010 [cit. 2013-05-04]. Dostupné z: http://www.shaman.cz/sifrovani/napoleonova-sifra.htm

[30] RSA (algorithm) - Wikipedia, the free encyclopedia. [online]. [cit. 2013-05-04]. Dostupné z: http://en.wikipedia.org/wiki/RSA_(algorithm)

[31] RSA algorithm (Rivest-Shamir-Adleman). *TECHTARGET*. 2005 [online]. [cit. 2013-05-04]. Dostupné z: http://searchsecurity.techtarget.com/definition/RSA

[32] RSA Algorithm. RIIKONEN, Pekka. [online]. 2002 [cit. 2013-05-04]. Dostupné z: http://xtrmntr.org/priikone/docs/rsa.pdf

[33] RSA Algorithm. IRELAND, David. *DI MANAGEMENT SERVICES PTY LIMITED*. [online]. Sydney, Austrálie, 2011 [cit. 2013-05-04]. Dostupné z: http://www.di-mgt.com.au/rsa_alg.html

[34] RC4 Encryption. RISE, Ralph, Suk-Hyun CHO a Devin KAYLOR. *UNIVERSITY OF WASHINGTON, Department of Mathematics*. [online]. [cit. 2013-05-05]. Dostupné z: http://www.math.washington.edu/~nichifor/310_2008_Spring/Pres_RC4%20Encryption.pdf

[35] Evaluation of the RC4 Algorithm for Data Encryption. MOUSA, Allam a Ahmad HAMAD. AN-*NAJAH UNIVERSITY a PalTel Company*. [online]. Nablus, Palestina, 2006 [cit. 2013-05-05]. Dostupné z: http://blogimg.chinaunix.net/blog/upfile2/100501133432.pdf

[36] The RC4 Stream Encryption Algorithm. STALLINGS, William. SOUTHERN POLYTECHNIC STATE UNIVERSITY. [online]. Marietta, GA, USA, 2005 [cit. 2013-05-05]. Dostupné z: http://cse.spsu.edu/afaruque/it6833/RC4.pdf

[37] RC4. *WIKIPEDIA*. [online]. [cit. 2013-05-05]. Dostupné z: http://en.wikipedia.org/wiki/RC4

[38] RC4 Tutorial with Animation. LOISEAU, Mark. [online]. 2012 [cit. 2013-05-05]. Dostupné z: http://blog.markloiseau.com/2012/07/rc4-tutorial/

[39] RC4 Encryption Algorithm. *VOCAL TECHNOLOGIES, Ltd.* [online]. Buffalo, New York, USA, 2003 [cit. 2013-05-05]. Dostupné z: http://csis.bits-pilani.ac.in/faculty/murali/netsec-11/seminar/refs/sneha6.pdf

[40] RC4. GRIER, Chris. [online]. San Diego, CA, USA [cit. 2013-05-05]. Dostupné z: http://imchris.org/crypto/html/ch07.html

[41] Cryptography - RC4 Algorithm. GALVANE, Quentin a Baptiste UZEL. *ROCHESTER INSTITUTE OF TECHNOLOGY, Department of Computer Science*. [online]. Rochester, NY, USA, 2012 [cit. 2013-05-05]. Dostupné z: http://www.cs.rit.edu/~ark/winter2011/482/team/u4/report.pdf

[42] Applicability of RC4 Algorithm in Bluetooth Data Encryption Method for Achieving better Energy efficiency of Mobile Devices. PARVEEN, Sharmin, Dr. a Sivalingham LATCHMANAN. *UNIVERSITY OF MALAYA, Department of Computer System and Technology, Faculty of Computer Science and Information Technology*. [online]. Kuala Lumpur, Malajsie, 2011 [cit. 2013-05-05]. Dostupné z: http://informatics.fsktm.um.edu.my/cameraready/Informatics_003.pdf