UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Kryptografie a šifrovací algoritmy

Jan Pillár

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan PILLÁR**
Osobní číslo:	**I07751**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Kryptografie a šifrovací algoritmy**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Teoretická část bude obsahovat přehled jednotlivých způsobů šifrování včetně vysvětlení principů algoritmů a jednoduchých příkladů. Algoritmy budou zpracovány po jednotlivých kategoriích od nejjednodušších (substituční, aditivní, transpoziční) po moderní šifry asymetrické i symetrické, přičemž větší prostor bude věnován RSA kódování.
Obsahem aplikační části bude program na šifrování/dešifrování textu v Delphi, který použije vybrané principy a algoritmy z teoretické části (např. Caesarova šifra, Vigenerova šifra). Demonstrováno bude i použití RSA algoritmu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***DENIS, T.; Johnson, S. Cryptography for Developers. Syngress, 2007. ISBN 1-59749-104-7.**
***STALLINGS, W. Cryptography and Network Security: Principles and Practice. Prentice Hall, 2003. ISBN 0-13-111502-2.**
***WOBST, R. Cryptology Unlocked. Wiley, 2007. ISBN 0-470-06064-6.**
***PIPER, F. C.; Murphy, S. Kryptografie: průvodce pro každého. Dokořán, 2006. ISBN 80-7363-074-5.**

Vedoucí bakalářské práce: **RNDr. Iva Rulicová**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

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 13. 04. 2011 Jan Pillár

Poděkování

Na tomto místě bych rád poděkoval paní RNDr. Ivě Rulicové za to, že ve mě vložila svou důvěru a byla ochotná vést moji bakalářskou práci na to toto téma. Obrovské díky patří všem členům mé rodiny za obětavou a nepřetržitou podporu jak během psaní této práce, tak především po celou dobu studia.

"The gratification comes in the doing, not in the results." -- James Dean

Anotace

Teoretická část bude obsahovat přehled jednotlivých způsobů šifrování včetně vysvětlení principů algoritmů a jednoduchých příkladů. Algoritmy budou zpracovány po jednotlivých kategoriích od nejjednodušších (substituční, aditivní, transpoziční) po moderní šifry asymetrické i symetrické, přičemž větší prostor bude věnován RSA kódování. Obsahem aplikační části bude program na šifrování/dešifrování textu v Delphi, který použije vybrané principy a algoritmy z teoretické části (např. Caesarova šifra, Vigenèrova šifra). Demonstrováno bude i použití RSA algoritmu.

Klíčová slova

RSA, AES, DES, kryptografie, šifrování, zabezpečení, klíče

Title

Cryptography and cryptographic algorithms

Annotation

Theoretical part will contain comprehensive survey of encryption including encryption algorithms principles explanation and giving some simple examples. The algorithms will be ordered by their complexness – from the simplest to the most recent ones, both symmetric and asymmetric. RSA algorithm will be covered in more depth. Result of practical part will be an application for text encrypting/decrypting written in Delphi which will show chosen principles and algorithms of the theoretical part (e.g. Caesar cipher, Vigenère cipher). It will contain a demo of RSA algorithm too.

Keywords

RSA, AES, DES, cryptography, encryption, security, keys

Obsah

Seznam zkratek

AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
CALTECH	California Technology
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CTR	Counter
DARPA	Defense Advanced Research Projects Agency
DES	Data Encryption Standard
EBC	Electronic Code Book
GCHQ	Government Communications Headquarters
GPL	General Public License
MIT	Massachusetts Institute of Technology
NBS	National Bureau of Standards
NSA	National Security Agency
NIST	National Institute of Standards and Technology
OFB	Output Feedback
QUIC	Quantum Information Center
RC4	Ron's Code 4
RSA	Rivest, Shamir, Adleman
SHA	Secure Hash Algorithm

Seznam obrázků

Seznam tabulek

1 Úvod

1.1 Úvodní slovo

Hlavním cílem této práce je přinést ucelený pohled na historii a vývoj kryptografie – vědy o utajování zpráv – od jejích počátků před více jak dvěma tisíci let až po současnost, včetně dnes používaných moderních šifrovacích algoritmů, z nichž některé se staly standardy, zatímco jiné byly naopak „vyřazeny“ a přestaly se používat, jelikož jejich bezpečnost byla prolomena.

Praktickou částí této práce je aplikace ve vývojovém prostředí Delphi 2005, jež má sloužit jako demonstrace vybraných šifrovacích algoritmů. Dané vývojové prostředí a jazyk Object Pascal byl zvolen z několika důvodů. Tím prvním je možnost tvorby grafické a uživatelsky přívětivé aplikace. Druhým je pak snadné pochopení zdrojového kódu, což značně ulehčí případným čtenářům jejich vlastní implementaci daných algoritmů.

Práce je rozdělena do desíti kapitol. Úvodní kapitola seznamuje čtenáře s kryptografií a vysvětluje nezbytné pojmy potřebné pro snadné pochopení zbytku práce. Druhá kapitola pojednává stručně o historii kryptografie a dobových postupech. Ve třetí kapitole se probírá rozdělení šifrovacích algoritmů, kdy jsou algoritmy rozděleny dle své složitosti a také podle logického hlediska. Čtvrtá kapitola představuje symetrické algoritmy od těch nejjednodušších až po ty moderní používané v současnosti, zatímco pátá kapitola se zaobírá algoritmy asymetrickými. Samostatná šestá kapitola je pak věnována algoritmu RSA, nejpoužívanějším algoritmu asymetrického šifrování. V sedmé kapitole jsou shrnuty rozdíly mezi symetrickým a asymetrickým přístupem z nejrůznějších hledisek. Osmá kapitola pojednává o možné budoucnosti kryptografie a kvantové kryptografii. Devátá kapitola se zabývá praktickou částí – aplikací pro šifrování textu demonstrující vybrané šifrovací algoritmy. Poslední kapitolu tvoří závěr a závěrečné shrnutí.

1.2 Terminologie

Před tím, než se začnu věnovat historii kryptografie, považuji za vhodné zmínit se o některých terminologických označeních a názvech, které budu nadále používat a které lze najít například v [COBB, 2004], [MURPHY, et al., 2006], [SINGH, 2009].

1.2.1 Kryptografie

Jak jsem již zmiňoval, historie kryptografie sahá cca 2500 let zpátky. Slovo kryptografie jako takové je složenina ze dvou řeckých slov: *kryptos*, což znamená „skrytý“ nebo „tajný“, a *gráph*, jehož význam je „psaní“. Kryptografie je tedy vědecká disciplína, jež se zabývá možnostmi utajování významu zpráv.

Ve 21. Století se s kryptografií setkáváme v praxi dennodenně, ačkoliv si to často ani neuvědomujeme – při výběru peněz z bankomatu (PIN i číslo účtu jsou na kartě

uložené v zašifrované podobě), volání z mobilního telefonu, přihlášení k webové e-mailové aplikaci, při zamykání a odemykání auta na dálku pomocí centrálního zamykání či dokonce při přehrávání filmů pomocí DVD přehrávače (DVD přehrávač obsahuje čip s dešifrovacím klíčem, díky čemuž lze přehrávat jen DVD z regionu, pro něž je tento přehrávač vyroben).

Jak zmiňuje nejen Vaudenay, kryptografie jako věda se soustředí na tři základní cíle a problémy [VAUDENAY, 2006 str. 10]:

1) ***důvěrnost*** – k informaci nesmí mít přístup nikdo, kdo k ní nemá oprávnění,
2) ***integrita*** – informace musí být chráněna proti neoprávněným úpravám,
3) ***autentifikace*** – z informace musí být jasně patrné, kdo je jejím autorem.

1.2.2 Kryptoanalýza

Kryptoanalýza je vědecká disciplína, která se naopak zabývá způsoby, jak ze zašifrované zprávy získat původní informaci bez znalosti použitého hesla, kódu nebo klíče. Jinými slovy, kryptoanalýza se snaží nalézt cesty, jak prolomit použitý šifrovací algoritmus nebo obejít způsob, který je běžně nezbytný k získání nezašifrované informace.

1.2.3 Kryptologie

Kryptologie je poté vědní obor, pod který spadají kryptografie i kryptoanalýza a který studuje bezpečnost a integritu dat, jejich přenos, ale i způsoby autentifikace. Dá se tedy říci, že zkoumá šifrování v širším kontextu – od tvorby šifer, přes jejich prolomení, nasazení v praxi, zajištění integrity a tak dále.

1.2.4 Další nezbytné pojmy v oblasti kryptologie

- *otevřený text* (v angličtině *plaintext*) – nezabezpečená informace; informace před zašifrováním,
- *šifrový text* (v angličtině *ciphertext*) – zabezpečená informace; informace po zašifrování,
- *šifrování* (v angličtině *encryption*) – proces zabezpečení informace; převod otevřeného textu do podoby šifrového textu za pomocí šifrovacího algoritmu,
- *šifrovací algoritmus* – sada pravidel použitých pro převod otevřeného textu do podoby šifrového textu,
- *dešifrování* (v angličtině *decryption*) – zpětný převod šifrového textu za pomoci dešifrovacího algoritmu do čitelné podoby, tj. do podoby otevřeného textu.

1.2.5 Vybrané základní definice

Zde si dovolím uvést vybrané definice některých základních pojmů z matematiky a informatiky, které budou používány dále v textu již bez bližšího vysvětlení [KÁLDY, 2000], [ONDRÁČKOVÁ, 2002].

Funkce na množině $D \subset \mathbb{R}$ je předpis, který každému číslu z množiny D přiřazuje právě jedno reálné číslo.

Tělesem rozumíme množinu T se dvěma binárními operacemi $+$ a $\cdot$ (sčítání, násobení), která je alespoň dvouprvková a ve které platí následující:

- $\forall a, b \in T: a + b = b + a$
- $\forall a, b, c \in T: (a + b) + c = a + (b + c)$
- $\exists 0 \in T\ \forall a \in T: a + 0 = 0 + a = a$
- $\forall a \in T\ \exists -a \in T: a + (-a) = (-a) + a = 0$
- $\forall a, b, c \in T: (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- $\exists 1 \in T\ \forall a \in T: a \cdot 1 = 1 \cdot a = a$
- $\forall a \in T - \{0\}\ \exists a^{-1} \in T: a \cdot a^{-1} = a^{-1} \cdot a = 1$
- $\forall a, b, c \in T: (a + b) \cdot c = a \cdot c + b \cdot c$

Dolní celá část reálného čísla x je největší celé číslo, které je menší nebo rovno x. Dolní celou část reálného čísla x značíme $\lfloor x \rfloor$.

Faktoriál kladného čísla n, značen jako $n!$, je součin všech celých kladných čísel menších či rovných n: $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$. Pokud je $n = 0$, pak je $n! = 1$.

Permutace množiny n prvků, kde $n \in \mathbb{N}$, je každá uspořádaná n-tice vytvořená z těchto prvků a to tak, že se každý z n prvků v této n-tici vyskytuje právě jednou.

Pokud a **beze zbytku dělí** b (tedy $\exists k \in \mathbb{Z}$ že $a \cdot k = b$), píšeme $a \mid b$ (číslo a je dělitelem čísla b).

Fermatovo číslo je takové číslo, které má tvar $F_n = 2^{2^n} + 1$, kde $n \in \mathbb{N}$.

Mějme dvě celá čísla a (dělenec) a b (dělitel), kdy platí, že $b \neq 0$. Pak operace ***a modulo b*** (zkráceně $a\ mod\ b$) vrací zbytek po dělení čísla a číslem b.

Jednorozměrné **pole** je homogenní datová struktura skládající se z prvků stejného datového typu. Tyto prvky jsou přístupné pomocí indexu.

Pojem **slovo** označuje v informatice nejmenší počet bitů, se kterými umí počítač pracovat. Většinou je tato velikost rovna velikosti registrů procesoru.

ASCII tabulkou rozumíme vzájemně jednoznačné přiřazení mezi znaky a čísly. Čísla mohou nabývat hodnot z intervalu $\langle 0, 127 \rangle$; pro potřeby národních sad se tento interval rozšiřuje a čísla tak mohou nabývat hodnot z intervalu $\langle 0, 255 \rangle$.

Logická operace **XOR**, též označovaná jako exkluzivní disjunkce, značená symbolem $\oplus$, je taková logická operace se dvěma operandy, která nabývá hodnoty „pravda“ pouze v tom případě, že se oba operandy nerovnají. To ukazuje následující pravdivostní tabulka:

Tabulka 1 – Pravdivostní tabulka logické operace XOR

a	b	a ⊕ b
0	0	0
1	0	1
0	1	1
1	1	0

.

2 Historie

Šifrování a utajení obsahu zprávy či zprávy samotné bylo používáno lidstvem odpradávna s jediným cílem – aby se k dané informaci dostala jen oprávněná osoba. Proto se v průběhu historie vystřídalo mnoho různých druhů šifrování, jejichž vývoj byl vždy dán dobovými možnostmi. Ve všech obdobích se kryptografie uplatňovala především v oblasti armády a státní správy. Až od konce 20. století lze její význam pozorovat i v běžném životě a každodenních situacích. Historii šifrování se detailně věnují především [MURPHY, et al., 2006], [SINGH, 2009].

Ve svých počátcích šifrování spočívalo v záměně jednotlivých písmen zprávy. Zpráva se tak stala pro toho, kdo neznal správné heslo nebo použitý šifrovací algoritmus, naprosto nečitelná. Méně často se pak lidé uchylovali k utajení zprávy samotné, například pomocí speciálního inkoustu – to je již ale záležitost jiného vědního oboru, steganografie.

Za nejstarší šifrovací nástroj je považován tzv. **skytale**, který používali již tři sta let před naším letopočtem starověcí Řekové a Sparťané během válečných operací. Jednalo se o váleček, na který se namotal pruh kůže či látky a na něj se psala po řádcích zpráva. Po sejmutí z válečku se text na pruhu látky jeví jako nesrozumitelný. Z toho vyplývá, že k dešifrování je zapotřebí:

1) vědět, že byl použit tento způsob šifrování,
2) mít váleček o stejném průměru.

Obrázek 1 – Skytale[1]

[1] Zdroj obrázku: http://upload.wikimedia.org/wikipedia/commons/5/51/Skytale.png

Až do 18. století [MURPHY, et al., 2006] patřily k vrcholným šifrám tzv. monoalfabetické substituční šifry. Jejich princip spočívá v záměně jednotlivých písmen podle určitého pravidla (či zcela náhodně). Prvním zdokumentovaným příkladem takovéto šifry je Caesarova šifra. Jak již její název napovídá, používal ji římský císař Gaius Iulius Caesar. Ten pro komunikaci se svým vojskem používal šifrování, kdy každé písmeno otevřeného textu nahradil písmenem o tři pozice v abecedě dál. Tento způsob šifrování byl po následujících téměř 800 let považován za nerozluštitelný. Až v 9. století našeho letopočtu našli Arabové způsob, jak jednoduchou substituční šifru prolomit.

Jednoduchá substituční šifra přežila více jak 1700 let, než ji nahradila polyalfabetická substituční šifra. O její vznik se přičinili učenci Leon Battista Alberti, Johannes Trithemius, Giovanni Porta a především Blaise de Vigenère, po němž dostal tento způsob svůj název. Jestliže monoalfabetické substituční šifry používaly k šifrování jedinou abecedu, tak Vigenèrova šifra jich používala hned několik. Tím se podařilo odstranit některé nevýhody jednoduché substituční šifry, o čemž bude hlouběji pojednávat čtvrtá kapitola. I přes svoji mnohonásobně vyšší složitost ani Vigenèrova šifra neodolala náporu kryptoanalytiků. Nezávisle na sobě se ji v 19. století podařilo rozluštit pruskému veliteli Friedrichu Kasiskému a anglickému inženýrovi Charlesovi Babbagovi.

Dalším mezníkem v evoluci kryptografie byl vynález telegrafu a rádia. V obou případech totiž přijdou do styku se zprávou další osoby. U telegrafu minimálně osoba odesílající a přijímající telegram, v případě rádia je počet možných narušitelů ještě vyšší. Do toho došlo k 1. světové válce, kdy se význam kryptologie ještě znásobil. Zachycování a dešifrování zpráv protivníka mělo v mnoha případech fatální důsledky na výsledek válečného tažení a v důsledku i války jako takové.

Nedlouho po 1. světové válce spatřil světlo světa jeden z nejznámějších šifrovacích systémů – Enigma. Jeho vynálezce, Němec Arthur Scherbius, vymyslel tak důmyslný mechanický stroj, že na více jak deset let znemožnil ostatním státům odposlouchávat německou komunikaci. Jen pro představu uvedu, že první verze tohoto přístroje umožňovala použít k šifrování či dešifrování jeden z 10 000 000 000 000 000 možných klíčů. Pozdější revize používaná během 2. světové války počet klíčů ještě znásobila 16 000krát. I přes tuto enormní složitost se ale spojencům podařilo kód prolomit, což mělo ve výsledku vliv na konec války – dle Sira Harryho Hinsleyho *„urychlilo prolomení kódu Enigmy konec války o celé tři roky“* [SINGH, 2009 str. 180].

Obrázek 2 – Šifrovací stroj Enigma[2]

Samostatná kapitola šifrování a šifer se začala psát ve 20. století po 2. světové válce, kdy do hry vstoupily – na dnešní dobu primitivní – počítače. Ty i přes svůj nízký výkon umožnily vznik takových šifrovacích algoritmů, o nichž se dřívějším generacím mohlo jen zdát. V této době se tak mohlo opustit od zaměňování písmen; místo toho šifrovací algoritmy pracovaly se samotnými kousky informací – s bity.

V polovině 70. letech 20. století došlo také k jednomu z nejvýznamnějších novodobých objevů v oblasti kryptografie, jenž výrazně přispěl k vyřešení problému distribuce šifrovacích klíčů – vznik asymetrického šifrování a asymetrických šifrovacích algoritmů. K tomu došly nezávisle na sobě týmy kryptologů a matematiků pracující v USA a ve Velké Británii.

V současnosti, ve 21. století, navíc nutnost šifrování výrazně narůstá a to především kvůli rostoucímu rozšiřování internetu, tedy nezabezpečeného přenosového média. Jelikož je nezbytné některé informace ochránit před možným zneužitím, jeví se použití šifrování jako ideální volba. Je to levný, rychlý, relativně snadný a především bezpečný způsob, jak připravit data k nezabezpečenému přenosu.

[2] Zdroj obrázku: http://upload.wikimedia.org/wikipedia/commons/thumb/4/44/EnigmaMachine.jpg/250px-EnigmaMachine.jpg

3 Klasifikace šifrovacích algoritmů

V této kapitole bude nastíněno, jakým způsobem je možné rozdělit šifrovací algoritmy. Než se k tomu ale dostanu, bylo by vhodné definovat, co to vlastně algoritmus a posléze šifrovací algoritmus je.

- **Algoritmus** je efektivní způsob, jak dosáhnout řešení určitého problému v konečném množství přesně definovaných kroků či instrukcí.
- **Šifrovací algoritmus** je poté speciálním případem algoritmu, který provádí šifrování a často je založen na nějakém matematickém problému. Vstupní data pro šifrovací algoritmus tvoří otevřený text a šifrovací klíč; výstupem šifrovacího algoritmu je šifrový text.

Šifrovací algoritmy je možné klasifikovat z nejrůznějších pohledů. V této práci jsou rozděleny do dvou velkých skupin podle způsobu použití šifrovacích klíčů – na symetrické a asymetrické. Tyto dvě skupiny jsou pak dále hlouběji probrány, přičemž v dalších kapitolách jsou algoritmy detailně představeny.

Jelikož je ale cílem této práce podat pohled na šifrovací algoritmy také od těch nejjednodušších po nejsložitější, je jako první uvedeno toto rozdělení, protože kopíruje historický vývoj.

Jen upozorním, že následující dvě podkapitoly obsahují pouze samotné rozdělení algoritmů. Algoritmy jako takové budou popsány až v následujících kapitolách.

3.1 Rozdělení dle složitosti

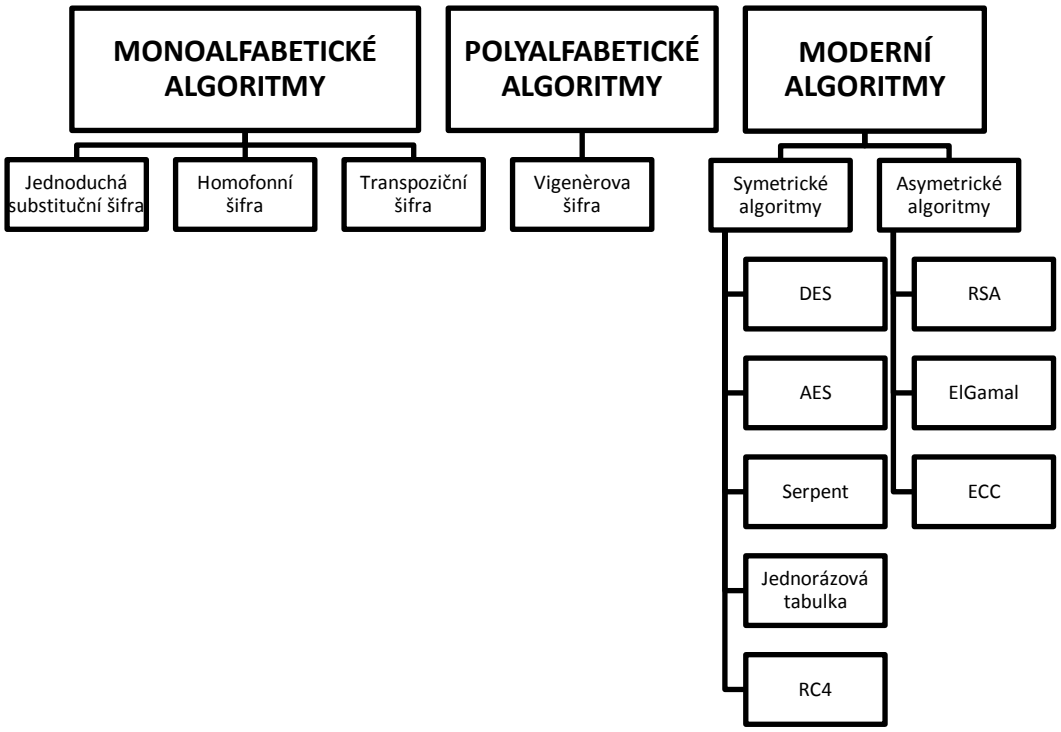
Níže uvedené schéma ukazuje rozdělení šifrovacích algoritmů na základě jejich složitosti. Jelikož toto rozdělení kopíruje historický vývoj, tak nejjednodušší algoritmy pracují se samotnými písmeny. Až s příchodem počítačů se objevily moderní algoritmy, které již pochopitelně pracují s daty v binární podobě.

Tu úplně nejzákladnější skupinu představují *monoalfabetické šifrovací algoritmy*, které podle určitého pravidla nahrazují písmena za jiná (v případě jednoduché substituční šifry), nahrazují je hodnotou z určité množiny (homofonní šifra) či mění pořadí písmen (transpoziční šifra).

O stupínek složitější jsou *polyalfabetické algoritmy*, jež stále pracují se samotnými písmeny, nicméně k náhradě písmen používají hned několik abeced, čímž se vyvarují některých nevýhod monoalfabetických algoritmů.

Nejdokonalejší, a nyní v praxi běžně používanou, skupinou algoritmů jsou *moderní algoritmy*. Ty již nepracují se znaky jako takovými, ale s jednotlivými bity otevřeného

textu, tedy se samotnými jedničkami a nulami. Moderní algoritmy můžeme rozdělit na dvě velké podskupiny a to v závislosti na tom, jestli používají k šifrování a dešifrování jeden jediný klíč (symetrické algoritmy), nebo klíče dva (asymetrické algoritmy).

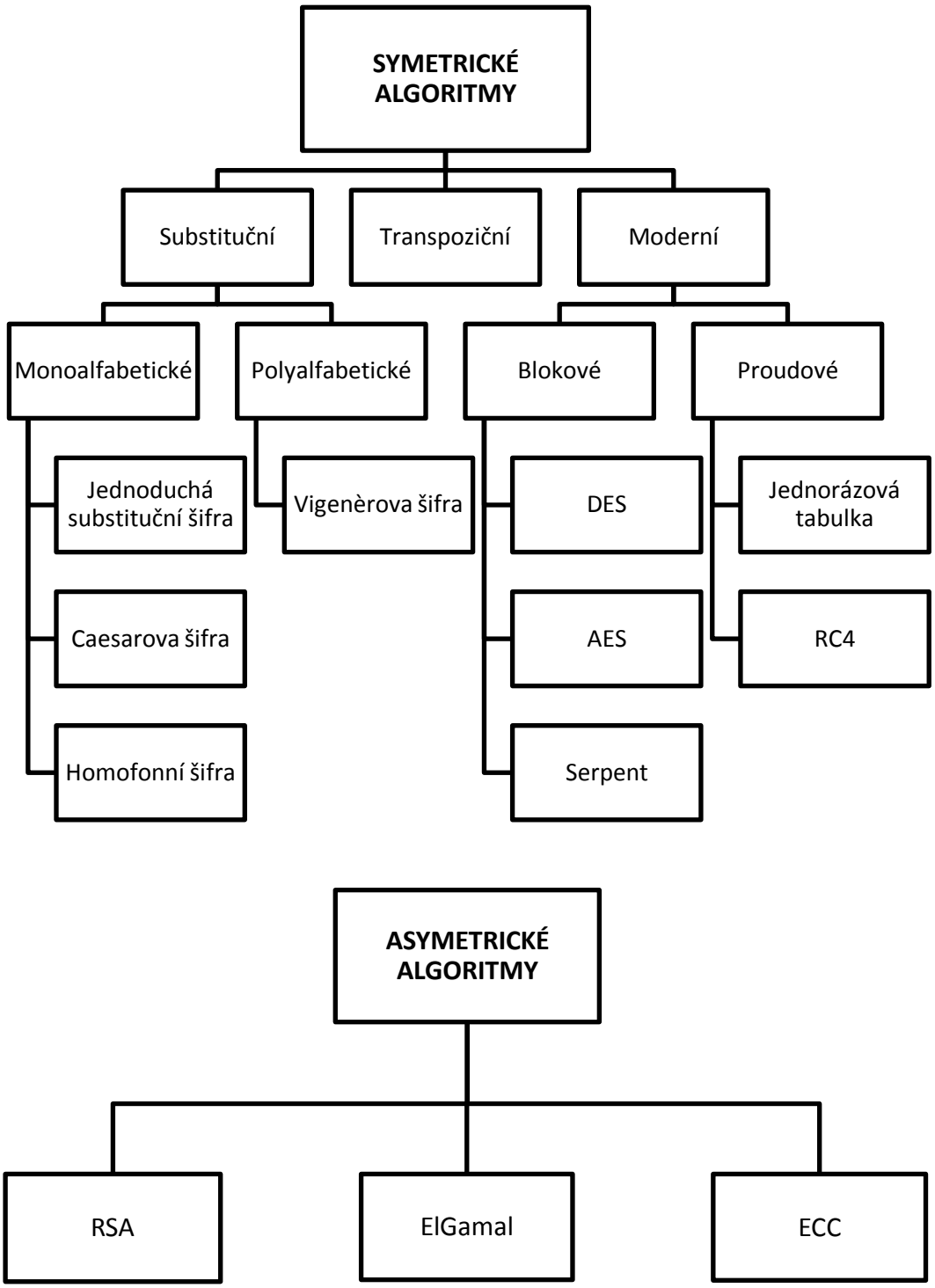


Obrázek 3 – Schéma rozdělení šifrovacích algoritmů dle složitosti

3.2 Rozdělení dle přístupu k šifrování

Mnohem logičtější rozdělení šifrovacích algoritmů je však podle toho, jestli používají k šifrování a dešifrování stejný klíč (symetrické algoritmy), či dva klíče rozdílné (asymetrické algoritmy).

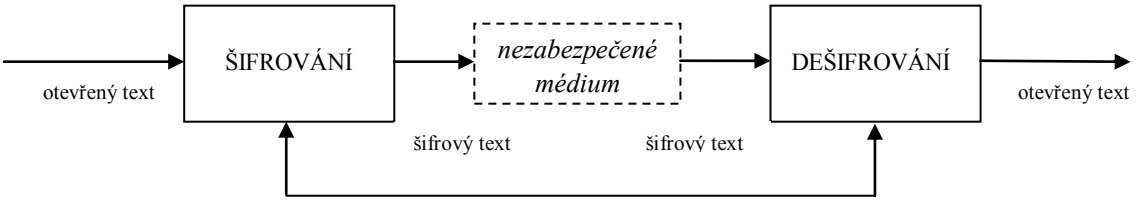
Zatímco druhá jmenovaná skupina se již dále nedělí, první můžeme rozčlenit do několika podskupin: Substituční algoritmy nahrazují jednotlivé znaky otevřeného textu jinými. Transpoziční algoritmy znaky nikterak nenahrazují, nýbrž mění jejich pořadí. Poslední skupinou jsou v současnosti používané moderní algoritmy. Ty lze dále rozdělit na blokové algoritmy, jež zpracovávají bity otevřeného textu po blocích identické délky, a proudové algoritmy, které data otevřeného textu šifrují „za běhu“ po jednotlivých bitech či bytech. Do této skupiny patří jediná dokonale bezpečná šifra, jednorázová tabulka.



Obrázek 4 – Schéma rozdělení šifrovacích algoritmů dle přístupu k šifrování

4 Symetrické šifrovací algoritmy

Všechny kryptografické systémy spoléhaly výhradně na symetrické algoritmy od svých historických počátků až do 20. století, konkrétně do roku 1976. Důvodem bylo – a stále je – to, že jsou mnohem jednodušší na implementaci matematického podtextu a ke svému fungování vyžadují jeden jediný šifrovací klíč. Proto se těmto algoritmům také někdy říká „*algoritmy s jedním klíčem*“, „*algoritmy s tajným klíčem*“ či „*algoritmy se symetrickým klíčem*“ [PAAR, et al., 2010].



Obrázek 5 – Schéma šifrování pomocí symetrických algoritmů

Stejný klíč tedy slouží k převodu otevřeného textu na šifrový při šifrování i v opačném procesu při dešifrování. Soukromý klíč proto musí znát jak odesílatel (autor) zprávy, tak i její příjemce. Z toho vyplývá jedna zřetelná nevýhoda těchto algoritmů: je nutné zajistit bezpečný přenos klíče mezi oba účastníky komunikace. Tím se přenáší problém bezpečného přenosu zprávy na problém bezpečného přenosu klíče.

4.1 Substituční algoritmy

Nejjednodušší podskupinu symetrických algoritmů tvoří substituční algoritmy. Právě tyto algoritmy se používaly od začátků lidstva, protože jejich použití je snadné a protože pracují pouze s písmeny otevřeného textu. Jejich princip spočívá v nahrazení jednoho písmena písmenem jiným – na základě určitého klíče.

4.1.1 Monoalfabetické versus polyalfabetické substituční algoritmy

Je to patrné již z názvů, v čem se liší monoalfabetické a polyalfabetické substituční algoritmy. Tím rozdílem je počet abeced, které dané algoritmy používají při šifrování či dešifrování otevřeného textu.

Monoalfabetické substituční algoritmy používají během celého šifrovacího/dešifrovacího procesu jen jednu jedinou abecedu. To znamená, že všechny stejné znaky otevřeného textu budou převedeny vždy na určitý a stejný znak.

Naopak polyalfabetické substituční algoritmy využívají během šifrování/dešifrování abeced hned několik. Mohou to být dvě abecedy, tři ale klidně i tisíc. To, která abeceda se při šifrování či dešifrování konkrétního znaku použije, závisí na

nějakém, předem domluveném, pravidle. Více abeced má tu výhodu, že stejné znaky otevřeného textu mohou být převedeny do rozdílných podob a to právě díky tomu, že jsou z různých abeced.

4.1.2 Jednoduchá substituční šifra

Jednoduchá substituční šifra je nejstarší a zároveň nejjednodušší zástupce monoalfabetických substitučních šifer. Její princip spočívá v postupném nahrazování jednotlivých písmen otevřeného textu jinými písmeny abecedy [MURPHY, et al., 2006].

Před začátkem šifrování je tedy nezbytné zvolit substituční tabulku, jež obsahuje seznam znaků a znaků, které je nahrazují. Ta může vypadat například takto:

Tabulka 2 – Substituční tabulka pro jednoduchou substituční šifru

Znak	Nahrazující znak
A	K
B	C
…	…
Ž	F

Pomocí této tabulky by se při šifrování všechny znaky „A“ otevřeného textu nahradily znakem „K“, všechny znaky „B“ otevřeného textu by byly nahrazeny znakem „C“ a tak dále.

Například otevřený text „**ABBA**“ by byl při použití této tabulky zašifrován do podoby „**KCCK**“. Obdobným způsobem se provede dešifrování, které je inverzí k výše zmíněnému postupu.

Caesarova šifra

Specifickým případem jednoduché substituční šifry je takzvaná Caesarova šifra, pojmenovaná po římském císaři G. I. Caesarovi. Ten zjednodušil proces nahrazování znaků pomocí substituční tabulky tak, že každý znak abecedy nahradil znakem, který je v abecedě o 3 pozice za ním [SINGH, 2009]. „A“ tak nahradil písmenem „D“, „B“ písmenem „E“ a tak dále, až konečně „X“ [3] nahradil písmenem „C“:

Tabulka 3 – Substituční tabulka při použití Caesarovy šifry

Znak	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	V	X
Nahra-zující znak	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	V	X	A	B	C

[3] V době G. I. Caesara obsahovala latinská abeceda jen 21 písmen (VAUDENAY, 2006).

Caesarovu šifru ale můžeme zobecnit a bude pak fungovat s libovolným posunem, dokonce i se záporným. V případě záporného klíče se jen budou písmena posouvat opačným směrem.

Výhodou Caesarovy šifry je to, že klíč tvoří číselná hodnota posunu. Je tak velmi jednoduché si stanovený klíč pamatovat a navíc není nutné ručně tvořit substituční tabulku. Naopak nevýhodou této šifry je fakt, že existuje jen tolik klíčů, kolik má zvolená abeceda písmen. Šifru tak není žádný problém prolomit pomocí hrubé síly (viz níže).

Prolomení a slabiny

Pokud by chtěl útočník či kryptoanalytik prolomit jednoduchou substituční šifru, má několik možností. Prvně může zkusit útok hrubou silou, tedy vyzkoušet všechny možné permutace znaků substituční tabulky. Druhou možností je využití frekvenční analýzy.

Útok hrubou silou však u takovéto šifry nepřipadá v úvahu (pokud se nejedná o Caesarovu šifru); lépe řečeno, k výsledku se pomocí něj útočník v dohledném čase nedobere – vznikají totiž hned dva problémy:

1) Počet možných klíčů je enormní. Anglickou abecedu tvoří 26 znaků, českou potom 34, a počet možných klíčů je roven faktoriálu daného čísla. Počet možných klíčů při použití anglické abecedy je tedy roven $26! = 4 \cdot 10^{26}$, v případě české abecedy dokonce $34! = 2{,}95 \cdot 10^{38}$.

2) Zejména u krátkých šifrových textů nastává problém, že při použití daného zkoušeného klíče vznikne otevřený text, který dává smysl, nicméně není tím správným. Útočník pak není schopný určit, který ze všech smysluplných textů je ten pravý.

S druhou metodou prolomení jednoduché substituční šifry přišli v 9. století našeho letopočtu arabští učenci. Všimli si toho, že se každý znak v průměrném vzorku textu vyskytuje s určitou četností – a na tom je založen princip **frekvenční analýzy**.

Pro útočníka je stěžejní znalost, v jakém jazyce je daná zpráva napsaná, jelikož každý jazyk používá určitá písmena častěji a jiná méně často. Poté je nezbytné spočítat počet výskytů jednotlivých písmen v šifrovém textu a za pomoci celkového počtu znaků šifrového textu určit jejich četnost.

V anglicky psaném textu je nejčastěji se vyskytujícím znakem písmeno „e“, kdežto v česky psaném textu je to písmeno „o“. Četnost výskytu písmen v česky psaném textu zobrazuje Tabulka 4.

Tabulka 4 – Procentuální četnost výskytu písmen české abecedy [KRÁLÍK, 2001]

Znak	Četnost	Znak	Četnost	Znak	Četnost
o	8,67 %	m	3,23 %	ž	1,00 %
e	7,70 %	u	3,14 %	č	0,95 %
n	6,54 %	á	2,24 %	š	0,81 %
a	6,22 %	z	2,20 %	ů	0,69 %
t	5,73 %	j	2,12 %	f	0,27 %
v	4,66 %	y	1,91 %	g	0,27 %
s	4,52 %	ě	1,65 %	ú	0,10 %
i	4,35 %	c	1,61 %	ň	0,08 %
l	3,84 %	b	1,56 %	x	0,08 %
k	3,74 %	é	1,33 %	ť	0,04 %
r	3,70 %	h	1,27 %	ó	0,03 %
d	3,60 %	ř	1,22 %	ď	0,02 %
p	3,41 %	ch	1,17 %	w	0,01 %
í	3,27 %	ý	1,07 %	q	0,00 %

V této chvíli přichází na řadu nejtěžší část – porovnávání jednotlivých četností znaků šifrového textu a nějakého statistického vzorku četností písmen abecedy daného jazyka. Není však možné strojově přiřadit písmeno, jež se nejčastěji vyskytuje v šifrovém textu, k písmenu s největší četností v daném jazyce a podobně.

Proto je nutné všímat si například krátkých slov – jednopísmenných či dvoupísmenných. Například v češtině mohou stát samostatně pouze písmena „A“, „I“, „K“ „O“ „S“, „U“, „V“ a „Z“. V angličtině jsou to dokonce jen písmena „A“ a „I“.

Z předchozích řádků vyplývá, že téměř každá jednoduchá substituční šifra se dá prolomit pomocí frekvenční analýzy. Pouze v případech, kdy šifrový text obsahuje méně jak 100 znaků [SINGH, 2009], vzrůstá bezpečnost šifry, nicméně i tak ji nelze označit za bezpečnou.

4.1.3 Homofonní šifra

Jednoduchá substituční šifra má svoje slabiny, přičemž největší je právě četnost výskytu jednotlivých znaků, čehož úspěšně využívá frekvenční analýza. Bezpečnější verzí patřící také do skupiny monoalfabetických substitučních šifer je takzvaná homofonní šifra, která se zaměřuje právě na výše zmíněnou slabinu. Jejímu popisu se věnují například [MURPHY, et al., 2006] a [SINGH, 2009].

Její princip spočívá v tom, že se každému písmenu abecedy přiřadí množina možných reprezentací a to v závislosti na jeho četnosti. Písmenům s největší četností se přiřadí reprezentací více a naopak písmena, která se v textu téměř neobjevují, budou mít přidělenu jen jednu zástupnou reprezentaci.

Jako zástupná reprezentace se může použít například skupina dvou či tří různých znaků nebo číslic. Při samotném šifrování se poté dané písmeno zašifruje tak, že se náhodně zvolí jeho libovolná reprezentace z dané množiny.

Tabulka 5 – Možné reprezentace písmen při použití homofonní šifry

Znak	Možné reprezentace
A	01 11 88 93 09 12 34 45
B	03 74
…	…
Ž	63

Jako příklad si ukažme zašifrování slova „**ABBA**“ pomocí výše uvedené tabulky. Toto slovo může být zašifrováno například do podoby „**34 03 74 88**“. Můžeme se tak snadno přesvědčit, že v tomto případě nelze nikterak poznat, že se původní slovo skládá jen ze dvou různých písmen.

Účelem homofonní šifry je ztížit frekvenční analýzu, což se jí z velké části daří. I když se dá frekvenční analýza použít, musí se útočník zaměřit na mnohem menší detaily. Zkoumat proto musí vztahy mezi jednotlivými písmeny, například to, která písmena se vyskytují za kterými.

4.1.4 Vigenèrova šifra

Vigenèrova šifra je nejznámějším zástupcem polyalfabetických šifer. Na konci 16. století, přesněji roku 1586, publikoval Blaise de Vigenère svoji práci nazvanou *Traicté des chiffres*, v níž popsal svůj návrh polyalfabetické šifry. Blaise de Vigenère svoji polyalfabetickou šifru navrhl velmi obecně; v praxi se však o Vigenèrově šifře hovoří v takové podobě, jak ji popsal Murphy [MURPHY, et al., 2006] a další autoři a jak je vysvětlena na následujících řádcích.

Je poměrně zajímavé, že ač byla ve své době Vigenèrova šifra velmi bezpečná, tak se neujala. Tehdejší kryptografové ji shledávali zbytečně složitou a nevěnovali jí patřičnou pozornost následujících dvě stě let.

Bohužel pro ně, v okamžiku kdy se k Vigenèrově šifře konečně obrátili, přišel Charles Babbage a po něm především Friedrich Wilhelm Kasiski se způsobem, jak ji prolomit [SINGH, 2009].

Jakožto polyalfabetická šifra používá Vigenèrova šifra více abeced. O tom, kolik abeced bude využívat, rozhoduje počet znaků abecedy. Princip totiž spočívá v tom, že pro každé písmeno původní abecedy se vytvoří pomocná abeceda, jejíž znaky jsou posunuté o určitou hodnotu. O kolik jsou posunuté, určuje vzdálenost písmena původní abecedy od začátku abecedy. Posun tedy činí $N - 1$, kde N je pozice písmena v abecedě. Proto

například pomocná abeceda pro písmeno „B“ je posunuta o jednu pozici, pro písmeno „C“ o dvě pozice a tak dále.

Pomocnou abecedu pro libovolné písmeno lze elegantně vytvořit za pomoci Caesarovy šifry.

Pokud vytvoříme pomocnou abecedu pro všechna písmena hlavní abecedy, vznikne takzvaný Vigenèrův čtverec. Na něm se dá velmi názorně ukázat šifrování i dešifrování

Při šifrování znaku x podle klíče k se přesuneme na řádek mající v prvním, tzv. klíčovém, sloupci písmeno k. V tomto řádku najdeme písmeno, které leží v x-tém sloupci. Toto písmeno je zašifrovanou podobou znaku x podle klíče k.

Analogicky při dešifrování znaku y podle klíče k se podíváme na řádek mající v klíčovém sloupci písmeno k a v tomto řádku najdeme znak y. Všimneme si toho, v jakém se nachází sloupci. Písmeno v prvním řádku Vigenèrova čtverce na pozici y-tého sloupce je potom dešifrovanou podobou znaku y podle klíče k.

Tabulka 6 – Vigenèrův čtverec

	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž
a	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž
b	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	o	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a
c	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	p	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b
č	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	q	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c
d	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	r	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č
ď	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	ř	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d
e	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	s	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď
f	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	š	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e
g	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	t	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f
h	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	ť	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g
i	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	u	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h
j	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	v	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i
k	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	w	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j
l	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	x	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k
m	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	y	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k	l
n	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	z	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	m
ň	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	ž	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n
o	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	a	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň
p	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	b	b	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o
q	q	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	c	c	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p
r	r	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b	č	č	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q
ř	ř	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c	d	d	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r
s	s	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č	ď	ď	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř
š	š	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d	e	e	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s
t	t	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď	f	f	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š
ť	ť	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e	g	g	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t
u	u	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f	h	h	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť
v	v	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g	i	i	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u
w	w	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h	j	j	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v
x	x	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i	k	k	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w
y	y	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j	l	l	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x
z	z	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k	m	m	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y
ž	ž	a	b	c	č	d	ď	e	f	g	h	i	j	k	l	n	n	ň	o	p	q	r	ř	s	š	t	ť	u	v	w	x	y	z

Ukažme si i Vigenèrovu šifru na příkladu, nicméně tentokrát zvolme otevřený text „**VIGENERE**“, jelikož si na něm snadno můžeme demonstrovat silné stránky Vigenèrovy šifry. Jako klíč zvolme slovo „**ŠIFRA**“ a k samotnému šifrování použijme Vigenèrův čtverec uvedený výše.

Tabulka 7 – Vigenèrova šifra – příklad použití

Otevřený text	V	I	G	E	N	E	R	E
Klíč	Š	I	F	R	A	Š	I	F
Šifrový text	O	R	N	U	N	X	Y	L

Z otevřeného textu jsme tedy při šifrování dostali šifrový text v podobě „**ORNUNXYL**“. Jak jsem zmiňoval, na tomto příkladu jdou vidět výhody Vigenèrovy šifry:

- v otevřeném textu se vyskytuje třikrát písmeno „E“, nicméně v šifrovém textu je reprezentováno třemi rozdílnými podobami,
- v šifrovém textu se vyskytuje dvakrát písmeno „N“, ačkoliv nahrazuje dvě rozdílná písmena otevřeného textu.

Prolomení a slabiny

Jako jiné substituční algoritmy i Vigenèrova šifra má své slabiny. Tou největší slabinou je fakt, že je klíč tvořen slovem, které – pokud je otevřený text delší než toto slovo – se periodicky opakuje. To znamená, že kryptoanalytik nemusí zjistit konkrétní podobu klíče. Bohatě mu stačí, pokud se dopracuje k délce klíče, protože všechny znaky šifrového textu na pozici $i + k \cdot n$, kde

- k je celé nezáporné číslo,
- n je délka klíče,
- i je pozice v šifrovém textu,

jsou zašifrovány pomocí totožného znaku klíče. Nad těmito znaky lze poté uplatnit frekvenční analýzu stejným způsobem, jako tomu je u jednoduché substituční šifry.

Hlouběji, včetně velmi podrobné praktické ukázky, se kryptoanalýzou Vigenèrovy šifry zabývá například Singh [SINGH, 2009].

4.2 Transpoziční algoritmy

Na rozdíl od substitučních algoritmů, které vyměňují písmena otevřeného textu za jiná, transpoziční algoritmy písmena nikterak nezaměňují. Místo toho podle určitého pravidla mění jejich pořadí tak, aby šifrový text nedával smysl, i když sestává z původních písmen.

Jak bylo zmíněno výše, tuto záměnu je nezbytné provádět podle určitého, předem domluveného, pravidla. Algoritmů proto existuje velmi mnoho a popisuje je Helen Fouché Gaines [GAINES, 1956] či Christensen [CHRISTENSEN, 2006], z jejichž prací bylo čerpáno.

4.2.1 Algoritmus „podle plotu“

Transpoziční algoritmus „podle plotu“ (z anglického „*rail fence*“) spočívá v tom, že na začátku šifrování si zvolíme kladné číslo udávající počet řádků, do nichž budeme vpisovat otevřený text. Toto číslo slouží jako první část tajného klíče. Druhou část tvoří takzvaný *offset*, neboli odsazení. Offset určuje, o kolik řádků je posunut začátek otevřeného textu. Offset může nabývat hodnot v rozmezí $\langle 0; 2R - 3 \rangle$, kde R je počet řádků.

Při šifrování postupujeme tak, že znaky otevřeného textu nepíšeme za sebou, ale po napsání znaku se přesuneme na další řádek. Pokud jsme zapsali znak na nejnižší řádek, začínáme psát směrem nahoru, ale to jen do chvíle, než se dostaneme k řádku prvnímu. Poté se opět mění směr psaní na směr dolů a tak stále dokola.

Po vepsání otevřeného textu do řádků získáme šifrový text tak, že po řádcích sepíšeme znaky. Tedy nejdříve znaky z prvního řádku, k nim připojíme znaky z druhého řádku a tak dále.

Dešifrování je mnohem ošemetnější záležitost, jelikož v šifrovém textu neznáme hranice jednotlivých řádků. Asi nejjednodušším a nejuniverzálnějším způsobem je zjistit délku šifrového textu a poté zašifrovat pomocí daného tajného klíče nový otevřený text tvořený z čísel $1, 2, \ldots, N$, kde N je počet znaků původního šifrového textu [WIDHIASTRA, 2010]. Vznikne nový, pomocný šifrový text, který sestává z pozic znaků šifrového textu v původním otevřeném textu.

Ukažme si tento algoritmus na jednoduchém příkladu. Zašifrujeme pomocí něj slovo „**`ALGORITMUS`**“, přičemž za klíč zvolíme číslo 4 a offset 2.

Tabulka 8 – Algoritmus „podle plotu“ – příklad použití

#				R				
	#			O		I		S
		A		G		T		U
			L				M	

Vznikne tak šifrový text v podobě „**`ROISAGTULM`**“. Znaky # do šifrového textu nepíšeme, slouží jen pro snadnější pochopení offsetu.

Jelikož je dešifrování trošku ošemetné, jak jsem zmiňoval výše, ukažme si na tomto příkladu i proces dešifrování.

Použijme šifrový text vzniklý výše („**ROISAGTULM**“) a stejnou hodnotu klíče.

Délka šifrového textu je 10 znaků, a tudíž se do řádků vepíší čísla 1 až 10.

Tabulka 9 – Algoritmus „podle plotu“ – pomocná tabulka při dešifrování

#				5					
	#			4		6			10
		1		3			7		9
			2					8	

Vznikne tak pomocný šifrový text „**5,4,6,10,1,3,7,9,2,8**“ (šifrový text nebude obsahovat čárky, zde byly vloženy pro snadnější rozlišení čísel):

Tabulka 10 – Algoritmus „podle plotu“ – 1. krok dešifrování

Původní šifrový text	**R**	**O**	**I**	**S**	**A**	**G**	**T**	**U**	**L**	**M**
Pomocný šifrový text	5	4	6	10	1	3	7	9	2	8

Tím jsme dosáhli toho, že nyní už víme, že znak „R“ se vyskytuje v otevřeném textu na 5. pozici, znak „O“ na 4. pozici a tak dále. Proto setřídíme pomocný šifrový text podle velikosti od nejmenšího k největšímu, přičemž odpovídajícím způsobem zaměníme i písmena původního šifrového textu:

Tabulka 11 – Algoritmus „podle plotu“ – 2. krok dešifrování

Původní šifrový text	**A**	**L**	**G**	**O**	**R**	**I**	**T**	**M**	**U**	**S**
Pomocný šifrový text	1	2	3	4	5	6	7	8	9	10

4.2.2 Sloupcová transpozice

Sloupcová transpozice pravděpodobně patří mezi nejjednodušší transpoziční algoritmy. Její princip spočívá v tom, že na začátku šifrování se zvolí kladné celé číslo, jež je menší než délka otevřeného textu. Toto číslo tvoří tajný klíč.

Otevřený text píšeme při šifrování do řádků. Na každý řádek napíšeme tolik znaků, kolik je hodnota klíče. Poté se přesuneme na další řádek a tento postup opakujeme.

Při dešifrování je nejprve nutné zjistit, do kolika řádků bude vpisován šifrový text. Toto číslo (označme ho R) zjistíme z délky šifrového textu (označme ji T) a klíče (označme ho K): $R = \frac{T}{K}$. Tento podíl zaokrouhlíme na nejbližší číslo nahoru.

Poté je nutné určit, kolik sloupců je úplných (označme je S). Jejich počet určíme v závislosti na tom, zdali podíl $R = \frac{T}{K}$ vyšel beze zbytku, či jsme zaokrouhlovali nahoru:

- podíl vyšel beze zbytku, pak $S = T - (K \cdot R)$,
- podíl vyšel se zbytkem, pak $S = T - (K \cdot (R - 1))$.

Nyní, když známe počet řádků a počet úplných sloupců, můžeme po sloupcích zapisovat šifrový text. Pokud ho přečteme po řádcích, získáme text otevřený.

Ukažme si tento algoritmus na stejném otevřeném textu jako v minulém příkladu („**ALGORITMUS**“). Za klíč opět zvolíme číslo 4.

Tabulka 12 – Sloupcová transpozice – příklad použití

A	**L**	**G**	**O**
R	**I**	**T**	**M**
U	**S**		

V tomto případě vznikne šifrový text v podobě „**ARULISGTOM**“.

4.2.3 Směrový algoritmus

Směrový algoritmus spadá spíše mezi manuální transpoziční algoritmy, nicméně dá se aplikovat i pomocí výpočetní techniky. Takový postup by byl ale poměrně těžkopádný.

Pro správné fungování je nutné na začátku zvolit rozměry mřížky, do níž se bude vpisovat jak otevřený text, tak i šifrový text. Často se stane, že text nezaplní mřížku celou, a proto se zbylá místa doplní náhodnými znaky.

Šifrování probíhá tak, že se zvolí klíč, jenž určuje směr, jakým se budou písmena vybírat, a počáteční bod. Jak takový klíč může vypadat, ukazuje příklad níže.

Dešifrování je s pomocí správného klíče také velmi jednoduché. Nalezne se koncový bod a od něj se šifrový text vpisuje pozpátku v přesně opačném směru klíče.

Stejně jako předchozí příklady, i tento algoritmus si demonstrujme s pomocí otevřeného textu („**ALGORITMUS**“). Tentokrát však zvolíme klíč „*spirála, proti směru, levý dolní*“. Klíč nám říká, že začneme písmenem v levém dolním rohu, budeme postupovat proti směru hodinových ručiček a cesta bude kopírovat tvar imaginární spirály.

Tabulka 13 – Směrový algoritmus – příklad použití

A	**L**	**G**	**O**
R	**I**	**T**	**M**
U	**S**	**A**	**L**

V tomto případě vznikne šifrový text „**USALMOGLARIT**“. Od předchozích příkladů se – kromě podoby, pochopitelně – liší počtem písmen, což je důsledek doplnění náhodnými znaky (označeny červeně) při vyplňování mřížky.

4.2.4 Myszkowskiho algoritmus

Tento algoritmus vymyslel roku 1902 Francouz Émile Victor Théodore Myszkowski, po němž také získal jméno [GAINES, 1956].

K použití tohoto algoritmu je nezbytné zvolit takové slovo, jež obsahuje alespoň dvě stejná písmena. Toto slovo poslouží jako klíč. Počet znaků klíče určuje počet sloupců použitých při šifrování a dešifrování.

Šifrování pomocí tohoto algoritmu je velmi podobné tomu u sloupcové transformace. Zde se však jednotlivé sloupce označí čísly. Tato čísla získáme tak, že u klíčového slova označíme písmena podle vzdálenosti od počátku abecedy. Písmenu nejblíže počátku abecedy přidělíme číslo jedna, dalšímu písmenu číslo dva a tak dále. Ta písmena, jež se vyskytují v klíčovém slově vícekrát, budou mít čísla stejná.

Nyní lze přistoupit k transpozici a přepisu do podoby šifrového textu. Začneme u sloupce označeného číslem jedna, poté číslem dva a tak dále. Pokud se dané číslo vyskytuje v klíči jen jednou, je směr zapisování shora dolů. Pokud však má několik sloupců stejná čísla, tak se postupuje takto: postupujeme horizontálně a zapisujeme písmena ze sloupců označených stejným číslem.

Ukažme si i tento algoritmus na příkladu s otevřeným textem „**ALGORITMUS**“. Zvolme klíčové slovo „**JANA**“, jež nám umožní demonstrovat rozdíl oproti sloupcové transformaci.

Tabulka 14 – Myszkowskiho algoritmus – příklad použití

J	*A*	*N*	*A*
2	*1*	*3*	*1*
A	**L**	**G**	**O**
R	**I**	**T**	**M**
U	**S**		

Na základě Myszkowskiho algoritmu postupujeme u sloupců číslo jedna horizontálně a u zbylých sloupců potom vertikálně. Tímto postupem dostaneme šifrový text „**LOIMSARUGT**“.

4.2.5 Transpoziční mřížky

Přestože tuto skupinu transpoziční algoritmů zařazuji až na konec, používají se již od roku 1550 [CHRISTENSEN, 2006]. Jejich princip spočívá v tom, že se otevřený text vpisuje do mřížky určitých rozměrů přes jakési stínítko. To má na určitých pozicích

vystřižená políčka (jak ukazuje obrázek níže). Vždy po vyplnění políček se stínítko otočí o 90° a pokračuje se ve psaní. Stínítko se může otočit celkem třikrát; tedy včetně první existují celkem čtyři pozice. Zbylá (prázdná) políčka mřížky se poté vyplní náhodnými znaky. Dešifrování se docílí jednoduchým přiložením stínítka na mřížku a postupným otáčením.

U tohoto způsobu jen podotknu, že vystřižená políčka musí být vhodně zvolena tak, aby po otočení neukazovala na již vyplněné pozice.

Obrázek 6 – Ukázka šifrovací mřížky[4]

Transpoziční mřížky používala během 1. světové války například i německá armáda. Pro komunikaci používala mřížky různých velikostí a kódových jmen. Například mřížka o rozměrech 5 x 5 písmen nesla kódové jméno „ANNA“ [CHRISTENSEN, 2006].

4.2.6 Prolomení a slabiny transpozičních šifer

Aby bylo možné transpoziční šifru prolomit, je nejprve nutné zjistit, zdali se o transpoziční šifru skutečně jedná. To je velmi jednoduché. Jelikož se při šifrování mění vždy jen pozice písmen, je procentuální zastoupení znaků v otevřeném i šifrovém textu totožné. Tento fakt nám napoví, že se nejedná o substituční šifru, nýbrž o transpoziční.

Transpoziční šifry nejsou příliš bezpečné a to především proto, že možných klíčů je velmi málo. Proto není problém použít útok hrubou silou a šifry tak prolomit. Na druhou stranu je pravda, že transpozičních algoritmů existuje mnohem víc než například substitučních. Bezpečnost šifry by však nikdy neměla záviset na neznalosti použitého algoritmu, a proto nemá počet algoritmů výraznější význam na bezpečnost.

Další slabinou je to, že ve skutečnosti nedochází při šifrování k záměně dat, ale jen k jejich přeuspořádání. Tudíž i zašifrovaná data obsahují stejné hodnoty jako data

[4] Zdroj obrázku: http://upload.wikimedia.org/wikipedia/commons/b/b9/Tangiers2.png

nezašifrovaná. Díky tomu může být v některých případech použití transpoziční šifry naprosto nevyhovující; například při transpozici několika stejných znaků bude šifrový i otevřený text vypadat totožně.

4.3 Moderní algoritmy

Až s nástupem počítačové éry na konci 70. let můžeme hovořit o skutečně moderních algoritmech. Ty již bez výjimky pracují se samotnými jednotkami informace – s jednotlivými bity. Zpočátku byly algoritmy navrhovány především pro hardwarové zpracování, kdy provádění jednotlivých instrukcí měly na starosti speciální šifrovací čipy. Až později se algoritmy začaly navrhovat primárně pro softwarovou implementaci, kdy nejzářnějším příkladem je algoritmus AES.

4.3.1 Dělení moderních algoritmů

Podle toho, jakým způsobem daný algoritmus pracuje s otevřeným textem, dělíme moderní algoritmy na dvě skupiny:

1) proudové algoritmy (anglicky *stream algorithms*),

2) blokové algoritmy (anglicky *block algorithms*).

Asi nejlépe shrnul rozdíl mezi proudovými a blokovými algoritmy Rueppel [SIMMONS, 1992]:

„Blokové algoritmy pracují s velkými bloky otevřeného textu a transformují je do bloků šifrového textu nezávisle na čase. Proudové algoritmy naopak mění jednotlivé bity či byty otevřeného textu na šifrový v závislosti na čase šifrování.“

Proudové algoritmy

Proudové algoritmy používají k šifrování tzv. *key stream*, což se dá přeložit jako „klíčovací tok“. Podle toho, jak je tento klíčovací tok odvozen, dělíme proudové algoritmy na [PAAR, et al., 2010 str. 30]:

- synchronní,
- asynchronní.

U synchronních proudových algoritmů je klíčovací tok odvozen pouze na základě daného šifrovacího klíče. Oproti tomu asynchronní proudové algoritmy odvozují klíčovací tok jak ze šifrovacího klíče, tak také z již zašifrovaných dat.

Jak zmiňuje nejen Rueppel [SIMMONS, 1992], tak u proudových algoritmů hraje významnou roli také čas šifrování. Tím je myšleno, že pokud bude otevřený text tvořený ze dvou stejných slov, tak po zašifrování prvního slova vznikne šifrový text určitého tvaru a druhé slovo bude poté zašifrováno do jiné podoby.

Proudové algoritmy jsou obecně rychlejší než blokové algoritmy a využívají se například v telekomunikacích, jelikož jsou méně náročné na výpočetní výkon a jsou efektivnější než blokové algoritmy. V telekomunikacích se konkrétně jedná o algoritmy **A5**, který slouží k šifrování telefonních hovorů v pásmu GSM, a **E0**, jenž se používá při šifrování komunikace přes Bluetooth [VAUDENAY, 2006].

Blokové algoritmy

Blokové algoritmy pracují vždy s určitým, pevně daným, počtem bitů otevřeného textu, které zašifrují. Následně pak pokračují s dalším blokem bitů a tak stále dokola, až dosáhnout konce otevřeného textu. Typicky je velikost bloku 64 nebo 128 bitů.

Ačkoliv jsou blokové algoritmy mnohem pomalejší než proudové, patří většina standardních šifrovacích algoritmů právě do této skupiny.

Speciálně blokové algoritmy mohou operovat v několika módech, z nichž některé jsou více vhodné než jiné. Jedná se o tyto módy, které dopodrobna popisují například Paar [PAAR, et al., 2010] , Vaudenay [VAUDENAY, 2006] či Schneier [SCHNEIER, 1995]:

- ECB (z anglického *Electronic Code Book*),
- CBC (z anglického *Cipher Block Chaining*),
- CFB (z anglického *Cipher Feedback*),
- OFB (z anglického *Output Feedback*),
- CTR (z anglického *Counter*).

Módů však existuje daleko více než těchto pět. Ty méně známé zmiňuje například právě Schneier [SCHNEIER, 1995]. Než se ale začnu jednotlivým módům věnovat, bylo by dobré definovat operátor zřetězení:

- Operátor zřetězení (značený ||) slouží k převzetí dvou řetězců a jejich spojení do jednoho jediného řetězce.

Nejjednodušším módem je **ECB**, který spočívá v tom, že se šifrový text x rozdělí na bloky požadované délky: $x = x_1 || x_2 || \dots || x_n$. Každý blok je poté samostatně zašifrován – označme operaci šifrování i-tého bloku jako $y_i = C(x_i)$. Výsledná podoba šifrového textu y je pak dána spojením jednotlivých zašifrovaných bloků: $y = y_1 || y_2 || \dots || y_n$.

ECB mód má však tři nevýhody. První je to, že stejné bloky otevřeného textu budou zašifrovány do stejných bloků šifrového textu. Druhou nevýhodou je fakt, že lze ze zašifrované zprávy některé bloky odstranit či je zaměnit, aniž by došlo k tomu, že se zpráva stane nesrozumitelnou. Příjemce tak mylně bude považovat upravenou zprávu za pravou, ačkoliv může mít kompletně jiný význam. Poslední nevýhodou je možnost vytvořit

si slovník (od toho název *Electronic Code Book*) obsahující bloky šifrového textu spolu s odpovídajícími bloky otevřeného textu.

Pokročilejší metodu šifrování představuje mód **CBC**. U tohoto módu nezávisí podoba zašifrovaného bloku dat y_i jen na bloku otevřeného textu x_i, nýbrž také na všech předchozích zašifrovaných blocích, které jsou k šifrovanému bloku dat připojeny pomocí operace XOR. Jistou náhodnost do šifrování vnáší také inicializační vektor IV.

Mějme opět otevřený text x rozdělený na jednotlivé bloky: $x = x_1 || x_2 || \dots || x_n$. Potom operace šifrování i-tého bloku vypadá takto: $y_i = C(y_{i-1} \oplus x_i)$. Pokud se $i = 1$, pak je hodnota $y_{i-1} = y_0 = IV$. Šifrový text y je stejně jako v případě ECB tvořen zřetězením jednotlivých zašifrovaných bloků $y = y_1 || y_2 || \dots || y_n$.

Při použití tohoto módu je nutné volit náhodně inicializační vektor IV, jinak by došlo k tomu, že u různých otevřených textů se stejným prvním blokem x_1 budou tyto první bloky zašifrovány do stejné podoby.

Je zajímavé si všimnout, že až doposud se procesu šifrování pomocí šifrovacího klíče účastnil blok dat otevřeného textu. V následujících módech tomu už bude jinak; zašifrována budou určitá „pomocná“ data, která se k šifrovanému bloku dat připojí pomocí operace XOR.

Módy **CFB** a **OFB** jsou si velmi podobné, jelikož oba dva využívají k šifrování klíčovací tok podobný tomu použitému u proudových šifer. Opět mějme otevřený text x rozdělený na jednotlivé bloky: $x = x_1 || x_2 || \dots || x_n$, tentokráte však mají bloky délku d-bitů. Šifrový text y má poté podobu $y = y_1 || y_2 || \dots || y_n$.

Klíčovací tok s_1 je inicializován hodnotou inicializačního vektoru IV, přičemž pro další průchody se hodnota klíčového toku s_i odvíjí od hodnot s_{i-1} a $C(s_{i-1})$ (v případě módu OFB) či s_{i-1} a y_{i-1} (v případě módu CFB). $C(s_{i-1})$ opět představuje šifrování, v tomto případě klíčového toku s_{i-1}. Hodnota $C(s_{i-1})$ však musí být ještě oříznuta zleva na délku d-bitů. Podobně musí být oříznuta, v tomto případě zprava, i hodnota s_i – a to v závislosti na požadované bitové délce, se kterou daný blokový algoritmus pracuje. Výstupem je v obou případech zašifrovaný blok $y_i = x_i \oplus C(s_i)$.

Posledním módem je **CTR** mód, který ke své práci vyžaduje takzvaný „čítač“. Čítačem můžeme rozumět funkci, jež generuje takové hodnoty, které se v dohledné době ani jednou nezopakují.

Na vstupu je opět šifrový text x rozdělený do bloků o délce d-bitů: $x = x_1 || x_2 || \dots || x_n$.

Nyní lze k šifrování přistoupit dvěma způsoby: s použitím inicializačního vektoru IV, či bez něj. Pokud se zvolí varianta s inicializačním vektorem IV, jak doporučuje Paar [PAAR, et al., 2010], tak by tento vektor měl mít méně než d-bitů. Zbylé bity totiž pro

každý blok vygeneruje právě čítač (označme ho T). Blok otevřeného textu x_i tak bude zašifrován do podoby $y_i = x_i \oplus C(IV \,||\, T_i)$, kde $C(IV \,||\, T_i)$ značí operaci šifrování bloku dat vytvořeného zřetězením inicializačního vektoru a hodnoty čítače pro daný blok dat.

Vaudenay (VAUDENAY, 2006) zmiňuje jednodušší postup, který nepočítá s inicializačním vektorem. Blok otevřeného textu x_i bude převeden do podoby $y_i = x_i \oplus C(T_i)$, kde $C(T_i)$ představuje šifrování hodnoty čítače pro daný blok dat.

V obou případech je nutné oříznout hodnotu $C(IV \,||\, T_i)$ či $C(T_i)$ na požadovanou bitovou délku d-bitů.

4.3.2 Jednorázová tabulka

Na konci 1. světové války byla vynalezena šifra, která je doposud jedinou skutečně neprolomitelnou šifrou. Patří do skupiny proudových algoritmů a jméno dostala po svém autorovi, americkém inženýrovi Gilbertovi Sandfordu Vernamovi. Přesto se častěji nazývá podle specifického šifrovacího klíče, který používá a jímž je takzvaná jednorázová tabulka. Detailně se jí věnuje většina autorů, například [MURPHY, et al., 2006], [PAAR, et al., 2010] či [SINGH, 2009].

Stejně jako mnoho autorů šifrovacích algoritmů, i Gilbert S. Vernam považoval svoji šifru za nerozluštitelnou. V roce 1949 však americký matematik Claude Elwood Shannon dokázal, že je tento algoritmus skutečně „**dokonale bezpečný**“. To znamená, že:

- bez znalosti klíče nelze získat z šifrového textu otevřený text ani v případě, že bychom měli neomezenou výpočetní kapacitu (a mohli vyzkoušet všechny klíče),
- po zašifrování nelze z šifrového textu odvodit žádné dodatečné informace o otevřeném textu, které by případný útočník mohl použít ke kryptoanalýze.

V praxi je však jednorázová tabulka téměř nepoužitelná. Můžou za to tři podmínky vztahující se ke klíči. Ty jsou nezbytné pro dokonalou bezpečnost:

1) klíč musí být dlouhý minimálně jako zpráva samotná,
2) klíč musí být vygenerován naprosto náhodně; v počítačích používané generátory pseudonáhodných čísel jsou nevyhovující,
3) klíč může být použit maximálně jednou.

Princip fungování

Mějme otevřený text O v bitové podobě o určité délce a zároveň klíč K v bitové podobě o odpovídající délce. Šifrový text S vznikne z klíče a otevřeného textu pomocí operace XOR, která je provedena postupně nad každou dvojicí odpovídajících bitů podle vzorce: $S = O \oplus K$.

Ukažme si tento algoritmus na jednoduchém příkladu. Mějme otevřený text v podobě „`1001010001110`“ a klíč o stejné délce „`1010101010101`“. Poté po použití operace XOR dojdeme k šifrovému textu, jejž ukazuje tato tabulka:

Tabulka 15 – Jednorázová tabulka – příklad použití

Otevřený text	1	0	0	1	0	1	0	0	0	1	1	1	0
operace XOR	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Klíč	1	0	1	0	1	0	1	0	1	0	1	0	1
Šifrový text	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**

Stejný postup lze analogicky použít i pro dešifrování. Na šifrový text S použijeme stejný klíč K, který připojíme operací XOR. Vznikne otevřený text O: $O = S \oplus K$.

Nedostatky

Ačkoliv má Vernamova šifra nespornou výhodu v dokonalé bezpečnosti, disponuje také několika nedostatky, které znemožňují její použití v běžné praxi.

Tím prvním je, že se problém bezpečného přenesení zprávy přesouvá na problém bezpečného přenosu klíče. Takový způsob je nesmírně obtížný a nákladný. Dovolit si ho tak mohou například jen vlády pro tu nejdůvěryhodnější komunikaci – Vernamova šifra se používá při komunikaci pomocí horké linky mezi Washingtonem a Moskvou, jak uvádí většina autorů, například [WOBST, 2007 str. 59]. Navíc klíč musí být dlouhý nejméně jako zpráva samotná, tudíž například k zašifrování jediného e-mailu s jednou přiloženou digitální fotografií ve vysokém rozlišení bychom potřebovali klíč dlouhý klidně i 4096 kiB, což je značně nepraktické.

Dalším nedostatkem je fakt, že se každý klíč může použít jen jednou. To má jednoduchý důvod: pokud se stejný klíč použije vícekrát, mohou kryptoanalytici eventuelně rozšifrovat některé útržky šifrového textu. To už každopádně ukázala historie, když Američané během studené války rozluštili určité fragmenty komunikace sovětských špionů právě díky tomu, že Sověti používali Vernamovu šifru a občas užili vícekrát stejný klíč, což se proslavilo pod názvem „*Projekt Venona*“ [WOBST, 2007 str. 58].

Jistým nedostatkem je také to, že pro generování klíče nelze použít současné generátory pseudonáhodných čísel. Tyto generátory totiž negenerují doopravdy náhodná čísla, nýbrž mezi těmito čísly existuje určitá, leč velmi nepatrná, závislost. S příchodem digitálních měřičů však lze získat skutečně náhodná data, například měřením času rozpadu jader radioaktivních prvků. To je ovšem velmi pracné a nákladné.

4.3.3 RC4

RC4 je v praxi nejvíce zastoupeným proudovým algoritmem. Vznikl již roku 1987 na půdě MIT. Název dostal podle svého tvůrce Ronalda Rivesta – **RC4** je zkratka ze slov „*Ron's Code 4*“, tedy „Ronův kód číslo 4“ [RIVEST, 199-?]. Uplatnění našel například

v e-mailovém nástroji Lotus Notes. V současnosti používá RC4 například internetová vrstva **SSL** [VAUDENAY, 2006 str. 47] či zabezpečení bezdrátových sítí **WEP** [WOBST, 2007 str. 274].

Princip fungování

Při použití RC4 jsou potřebné dvě proměnné i, j každá o velikosti jednoho bytu a dále pole 256 prvků o velikosti taktéž jednoho bytu označené jako S. Nezbytně nutný je pochopitelně šifrovací klíč K o délce n bitů.

Samotné šifrování i dešifrování se dá rozdělit do dvou částí, jak detailně popisují například [PAAR, et al., 2010], [VAUDENAY, 2006], [WOBST, 2007]:

1) inicializace klíčovacího toku,

2) generování klíčovacího toku a následné šifrování (analogicky dešifrování).

V inicializační části se nejprve hodnoty proměnných i, j nastaví na nulu. Následně se všechny prvky pole S naplní hodnotami, které se rovnají indexu daného prvku v poli. Poté se do tohoto pole „přimíchá“ šifrovací klíč a to tak, že se cyklicky zvětšuje hodnota proměnné i o jedničku a vždy se spočte hodnota $j = j + S[i] + K[i \bmod n]$. Následně se prohodí hodnoty v poli na pozicích $S[i]$ a $S[j]$ a cyklus pokračuje od začátku.

V druhé fázi se nastaví hodnoty i, j na hodnotu nula a následně dochází v cyklu ke generování klíčovacího toku. Nejprve se aktualizuje hodnota proměnné i o jedničku a poté hodnota proměnné $j = j + S[i]$. Nyní se opět prohodí hodnoty pole na pozicích $S[i]$ a $S[j]$. V této chvíli dochází ke generování hodnoty klíče z klíčovacího toku. Klíč má hodnotu $K = S[S[i] + S[j]]$. Tato hodnota se pomocí operace XOR připojí k šifrované (či dešifrované) části otevřeného textu (či šifrového textu). Tím končí průchod cyklem a začíná průchod nový.

Slabiny a nevýhody

RC4 se potýká se stejným problémem jako jiné symetrické algoritmy – tím je distribuce šifrovacího klíče. Na další nevýhodu upozornil Andrew Roos v roce 1995. Roos přišel s tím, že i přes počáteční inicializaci není klíčovací tok dostatečně náhodný, a to především na svém začátku [ROOS, 1995].

4.3.4 Další proudové algoritmy

Jednorázová tabulka a především RC4 jsou nejvýznamnějšími zástupci proudových symetrických šifrovacích algoritmů. Podobný význam mají taktéž algoritmy A5 a E0. Kromě nich však existuje mnoho dalších proudových šifrovacích algoritmů. Jejich seznam lze nalézt například v [Stream Cipher, 2006].

4.3.5 DES

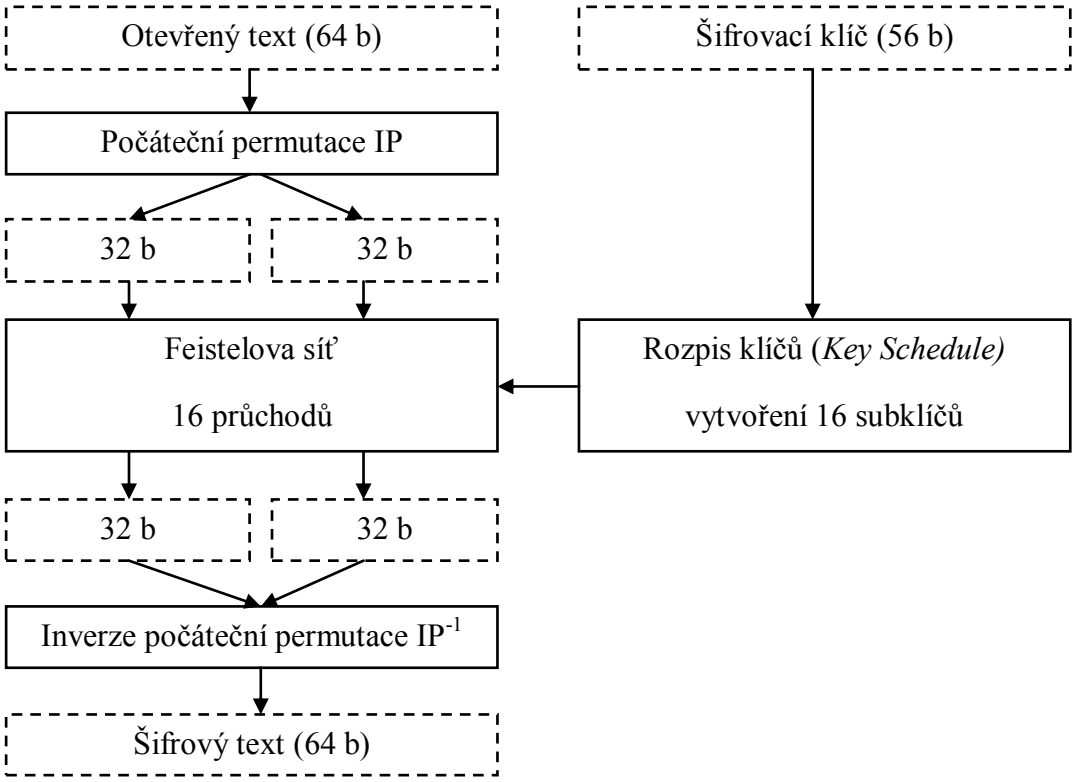
Algoritmus DES (zkratka z anglického *Data Encryption Standard* – Standard pro šifrování dat) vznikl v 70. letech na zakázku amerického úřadu pro standardy NBS, což

byla obdoba dnešní NIST. Jedná se o blokový algoritmus, s nímž přišla společnost IBM. Ta využila znalostí svého zaměstnance Horsta Feistela, jenž se podílel na tvorbě jiného šifrovacího algoritmu nazvaného LUCIFER. V roce 1977 byl algoritmus DES po dlouhém zkoumání a testování schválen jako standard. DES byl navržen k hardwarové implementaci, o čemž napovídá i to, že používá pouze operace XOR, jež jsou na hardwaru snadno a rychle proveditelné [VAUDENAY, 2006].

Algoritmus DES vydržel na výsluní přibližně dvě dekády. Až na konci 90. let byl shledán nedostatečně bezpečným.

DES patří mezi blokové šifrovací algoritmy. Pracuje s bloky dat o velikosti 64 bitů a klíčem dlouhým 56 bitů, ačkoliv se často používá klíč o délce 64 bitů – nejvýznamnější bit z každé osmice bitů slouží jako paritní. Díky svému významu v letech minulých je princip algoritmu DES popsán mnoha autory, z nichž velmi detailně o něm napsali například Paar, Vaudenay či Wobst [PAAR, et al., 2010], [VAUDENAY, 2006], [WOBST, 2007].

Proces šifrování se dá rozdělit do několika částí, jak to ukazuje schéma níže:



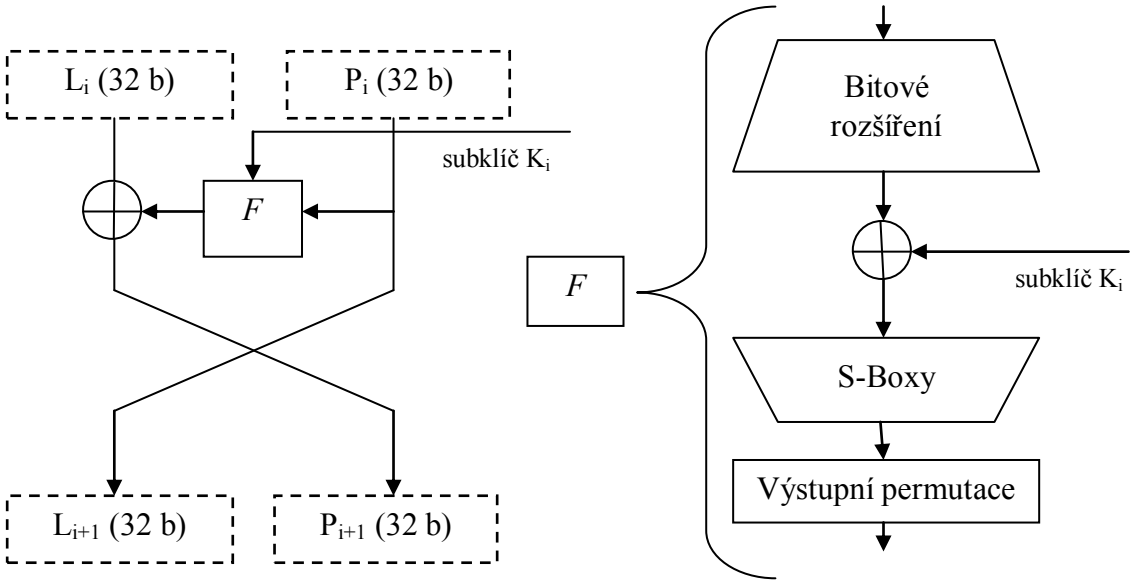
Obrázek 7 – Schéma šifrovacího postupu u algoritmu DES – vytvořeno na základě [VAUDENAY, 2006]

Na začátku a konci šifrování dochází k permutaci bitů. Odborní autoři však zmiňují to, že tyto dvě sekce *„nemají na bezpečnost algoritmu žádný vliv“* [PAAR, et al., 2010], [VAUDENAY, 2006], [WOBST, 2007].

Feistelova síť

Srdce algoritmu DES tvoří Feistelova síť spolu s unikátní funkcí, která se provádí při každém průchodu. Vstupem je blok 64 bitů dat, který se rozdělí na dvě poloviny – levou a pravou polovinu. Nad těmito polovinami se následně provedou různé operace. Na konci průchodu se levá a pravá polovina prohodí a poslouží jako výstup (či vstup pro další průchod). Feistelova síť v algoritmu DES se skládá z 16 průchodů.

Podívejme se nyní blíž na to, jak vypadá i-tý průchod Feistelovo sítí.

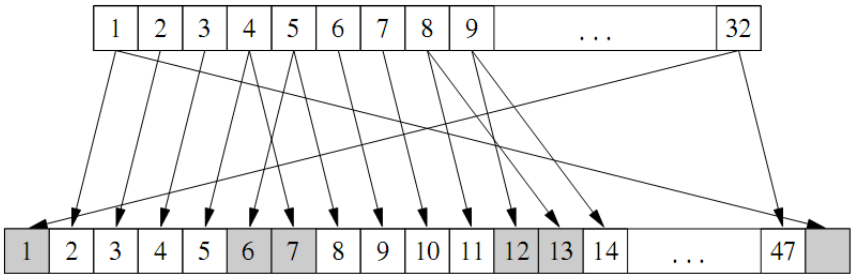


Obrázek 8 – Schéma i-tého průchodu Feistelovo sítí včetně schématu funkce *F* – vytvořeno na základě [WOBST, 2007]

Jak je vidět na levé části schématu uvedeného výše, tak v i-tém průchodu Feistelovo sítí se pravá polovina vstupních dat P_i stane levou polovinou výstupních dat L_{i+1}. Zároveň se nad pravou polovinou vstupních dat P_i provede s pomocí subklíče K_i funkce *F*. Její výstup se připojí k levé polovině vstupních dat L_i pomocí operace XOR. Výsledek pak tvoří pravou část výstupních dat P_{i+1}. Jak tedy vidíme, obě poloviny vstupních dat se na výstupu opravdu prohodily.

Mnohem zajímavější je však to, co se děje uvnitř funkce *F*, což znázorňuje pravá část schématu výše. Do ní vstupuje pravý blok P_i o velikosti 32 bitů a subklíč K_i o velikosti 48 bitů vygenerovaný dle rozpisu klíčů. Nejprve dojde k tomu, že se 32 bitů vstupních dat rozšíří na 48 bitů.

To má prostý důvod. Změna byť jediného bitu má díky tomuto rozšíření mnohem větší vliv na zbytek bloku, jelikož se polovina bitů rozkopíruje. Cílem tedy je maximalizovat vliv nepatrných změn otevřeného textu na výslednou podobu šifrového textu.



Obrázek 9 – Bitové rozšíření ve funkci *F* algoritmu DES[5]

Blok dat se poté skombinuje se subklíčem K_i pomocí operace XOR. Nyní přichází na řadu substituce jednotlivých bitů v takzvaných **S-Boxech**, jejichž název je odvozen právě od slova *substitution*. DES obsahuje osm S-Boxů, které jsou tvořeny předem danými tabulkami o 16 sloupcích a 4 řádcích, jež určují, jak se budou měnit vstupní data na výstupní. Jejich cílem je především samotné šifrování vstupních dat a zároveň redukce počtu bitů bloku dat zpět na 32.

S-Box přijímá na vstupu šest bitů a na výstupu vrací bity pouze čtyři. Vstup, který označíme jako $b_1b_2b_3b_4b_5b_6$, se rozdělí na dvě skupiny. První tvoří bity b_1b_6 a druhou $b_2b_3b_4b_5$. První skupina může nabývat jen $2^2 = 4$ hodnot. Tato hodnota určuje číslo řádku S-Boxu. Druhá skupina může nabývat $2^4 = 16$ hodnot. Tato hodnota určuje číslo sloupce S-Boxu. Každý řádek S-Boxu obsahuje hodnoty $0, 1, \ldots, 15$, které jsou v dvojkové soustavě reprezentovány právě čtyřmi bity. A tak výstupem S-Boxu je bitová podoba čísla na pozici v řádku číslo b_1b_6 a sloupci číslo $b_2b_3b_4b_5$.

Tabulka 16 – S-Box číslo 1 [PAAR, et al., 2010]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

[5] Zdroj obrázku: (PAAR, et al., 2010)

Ač se to nemusí na první pohled zdát, S-Boxy jsou extrémně pečlivě navržené tak, aby vnesly do šifrovaných dat co největší zmatek a nelineárnost. Jednotlivé hodnoty v S-Boxech jsou rozmístěny tak, aby splňovaly složitá pravidla, která zmiňuje Christof Paar [PAAR, et al., 2010]. Tvůrci algoritmu DES dokonce údajně *„nechali pracovat své počítače několik měsíců jen na samotném návrhu podoby S-Boxů“*, což zmiňuje jako zajímavost [WOBST, 2007 str. 140].

Podobu všech S-Boxů naleznete v Příloze B na straně 91.

Poslední operací ve funkci *F* je výstupní permutace, jež má za úkol dále rozptýlit jednotlivé bity.

Rozpis klíčů (Key Schedule)

Pomocí rozpisu klíčů vzniká z šifrovacího klíče 16 různých „subklíčů“ o velikosti 48 bitů – pro každý průchod Feistelovo sítí jeden. Než se však šifrovací klíč dostane do procesu přeměny na subklíče dle rozpisu klíčů, vynechá se každý osmý bit, který slouží jen jako paritní. Paar se zmiňuje, že *„dosud není jasné, proč byl takto šifrovací klíč pro DES navrhnut“* [PAAR, et al., 2010 str. 67].

Po tomto vynechání osmi bitů vznikne klíč o 56 bitech, který se rozdělí na dvě poloviny o 28 bitech a ty poté vstupují do procesu přeměny dle rozpisu klíčů.

Dle rozpisu se bity každé poloviny posouvají vždy o jednu, či dvě pozice vlevo a to v závislosti na tom, o kolikátý se jedná průchod:

- při 1., 2., 9. a 16. průchodu se bity posouvají o jednu pozici,
- při 3., 4., 5., 6., 7., 8., 10., 11., 12., 13., 14. a 15. průchodu se bity posouvají o dvě pozice.

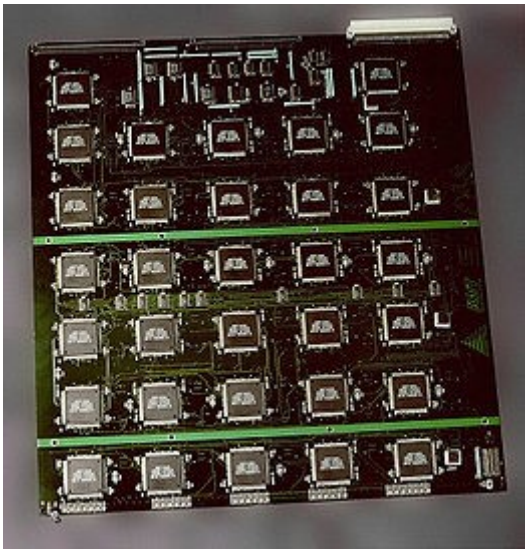
Po posunu se dvě poloviny spojí zpět v 56 bitovou sekvenci, která projde skrz zhušťovací substituční tabulku, která je pevně daná a slouží k redukci z 56 na 48 bitů. Výsledek pak poslouží jako subklíč pro daný průchod Feistelovo sítí.

Tímto se dosáhne toho, že každý bit původního klíče je použit přibližně ve 14 subklíčích [PAAR, et al., 2010], [WOBST, 2007].

Slabiny

Největší slabinou byla od počátku délka šifrovacího klíče. V 80. letech to nepředstavovalo větší problém, jelikož bylo tehdy nemožné – či to bylo extrémně nákladné – prohledat všech 2^{56} možných klíčů. To se změnilo v 90. letech. Jako první přišel s návrhem přístroje, který by byl schopný prolomit DES za jeden a půl dne, v roce 1993 *Michael Wiener*. Postavit tento přístroj by tehdy stálo jeden milion dolarů. V roce 1998 však s podobným funkčním strojem přišla společnost *Electronic Frontier Foundation*. Nazýval se *Deep Crack* a jeho cena se pohybovala „jen“ okolo čtvrt milionu

dolarů [PAAR, et al., 2010]. Tento přístroj byl posledním hřebíčkem do rakve algoritmu DES jako bezpečného šifrovacího algoritmu.



Obrázek 10 – EFF DES Cracker (Deep Crack)[6]

Další z nevýhod bylo, že DES byl navržen jako algoritmus vhodný pro hardware. Existovaly tak čipy, které dokázaly šifrovat a dešifrovat pomocí tohoto algoritmu mnohem rychleji než softwarově napsaný program.

Zajímavosti

Ve všech materiálech, z nichž jsem čerpal, vyjadřují autoři pochyby, zdali americká Národní bezpečnostní agentura (NSA) nezasáhla do vývoje tohoto algoritmu. První nejasností je to, že původní návrh od IBM počítal s klíčem o délce 128 bitů. Snížení délky na 56 bitů – a tím pádem i odolnosti proti prohledávání všech klíčů – bylo právě na žádost NSA [PAAR, et al., 2010].

Druhá věc, nad kterou se autoři zastavují, je fakt, že tabulky pro S-Boxy byly uchovány v tajnosti a nebyly přístupné veřejnosti – což odporuje kryptologickému pravidlu, že síla algoritmu nemá spočívat v neznalosti postupu, ale v neznalosti klíče. Toto vedlo ke spekulacím, zdali NSA neumístila právě do S-Boxů nějaký analytický nástroj – zadní vrátka –, pomocí něhož by šlo rozšifrovat libovolná data bez znalosti klíče [PAAR, et al., 2010], [VAUDENAY, 2006], [WOBST, 2007].

[6] Zdroj obrázku: http://upload.wikimedia.org/wikipedia/commons/thumb/b/bd/Board300.jpg/260px-Board300.jpg

4.3.6 3DES

Jelikož od konce 90. let nebyl algoritmus DES považován za bezpečný, přišlo se s jeho alternativou označovanou jako *3DES* nebo „*trojitý DES*“. Její princip spočívá v tom, že se data šifrují pomocí standardního algoritmu DES třikrát s rozdílnými – nebo volitelně stejnými – klíči o standardní délce 56 bitů. Podrobnému popisu algoritmu se věnuje například [PAAR, et al., 2010].

Označme nyní pro názornost ot jako otevřený text, st jako šifrový text, DES_{K_i} jako šifrování pomocí algoritmu DES a klíče K_i a konečně $DES_{K_i}^{-1}$ jako dešifrování pomocí algoritmu DES a klíče K_i.

Mějme tedy tři klíče K_1, K_2 a K_3. Potom je lze použít těmito způsoby:

1) pokud $K_1 \neq K_2 \neq K_3$, potom $st = DES_{K_1}\left(DES_{K_2}\left(DES_{K_3}(ot)\right)\right)$ a klíč má délku $3 \cdot 56 = 168$ bitů,

2) pokud $K_1 = K_2 = K_3$, potom $st = DES_{K_1}\left(DES_{K_2}^{-1}\left(DES_{K_3}(ot)\right)\right)$ a klíč má délku $1 \cdot 56 = 56$ bitů.

Vaudenay sem navíc řadí ještě jednu variantu [VAUDENAY, 2006]:

3) pokud $K_1 = K_3, K_2$, potom $st = DES_{K_1}\left(DES_{K_2}^{-1}\left(DES_{K_3}(ot)\right)\right)$ a klíč má délku $2 \cdot 56 = 112$ bitů.

Tento algoritmus má však jednu zásadní nevýhodu, která pramení z toho, že se jedná o několik šifrování pomocí standardního algoritmu DES. 3DES je při šifrování dvakrát až třikrát pomalejší (v závislosti na zvolených klíčích) než standardní DES.

4.3.7 AES

Zkratka AES zastupuje anglická slova *Advanced Encryption Standard* – Standard pro pokročilé šifrování. Potřebu nového standardu v šifrování vyjádřil americký úřad NIST na začátku roku 1997, jelikož bylo jasné, že stávající standard (DES) nebyl dostatečně bezpečný.

NIST proto vyhlásil otevřenou soutěž, které se mohl zúčastnit kdokoliv a přijít se svojí podobou nového šifrovacího algoritmu. Ten musel splňovat stanovené podmínky:

- bude se jednat o symetrický blokový algoritmus pracující s velikostí bloku 128 bitů,
- bude možné používat klíče o velikosti 128, 192 a 256 bitů,
- a další, viz [WOBST, 2007 str. 263].

Do finálního kola nakonec postoupila pětice algoritmů:

- MARS,
- RC6,
- Rijndael,
- Serpent,
- Twofish.

V první polovině roku 2001 oznámil NIST vítěze – standardem AES se stal algoritmus **Rijndael**, jenž vytvořili dva belgičtí kryptografové Joan Daemen a Vincent Rijmen. Z jejich příjmení také vznikl název tohoto algoritmu.

Jelikož bude v práci dále pracováno výhradně s termínem AES, upozorním tedy dopředu, že se pod touto zkratkou skrývá právě algoritmus Rijndael.

Jak vyplývá z požadavků vymezených pro tvorbu AES, jedná se o blokový algoritmus, jenž pracuje s bloky dat o velikosti 128 bitů. Na rozdíl od předešlého standardu nenahlíží AES na blok dat jako na jednotlivé bity, nýbrž jako byty. Zdrojový kód AES byl uveřejněn široké veřejnosti, a tudíž ho mohl kdokoliv podrobit zkoumání a zkusit ho prolomit. Dalším důsledkem je fakt, že je jeho princip popsán ve většině knih o kryptografii. Věnují se mu například [DENIS, et al., 2007], [PAAR, et al., 2010], (VAUDENAY, 2006], (WOBST, 2007].

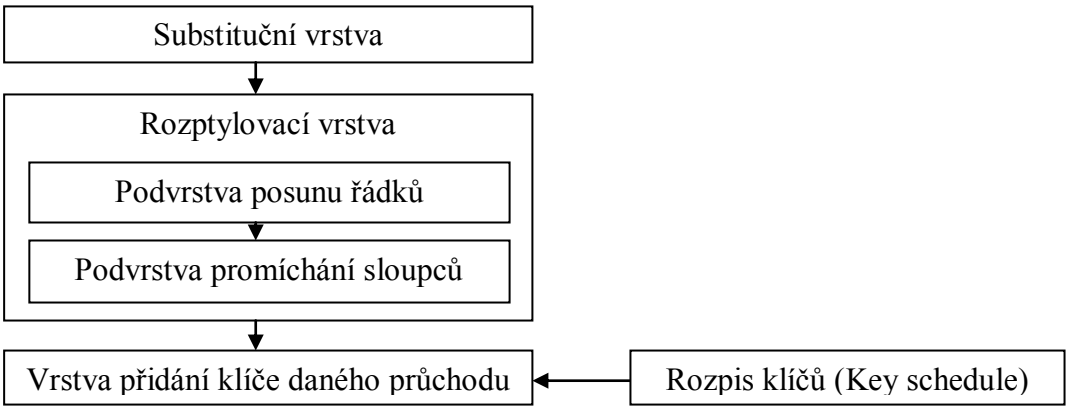
Princip algoritmu je založen na teorii konečných těles, konkrétně tělesa $GF(2)$ se dvěma prvky a dále tělesa $GF(2^8)$ s 256 prvky. Stěžejní jsou v rámci těchto těles operace sčítání, násobení a získání inverzního prvku. Jelikož je však tématika algebry konečných těles nad rámec této práce, odkazuji proto čtenáře například na stručný úvod do této problematiky od Christofa Paara [PAAR, et al., 2010 stránky 90-99].

Na rozdíl od jiných blokových algoritmů není AES postavený na Feistelovo sítích. Šifrování a dešifrování pomocí tohoto algoritmu probíhá ve vrstvách, které pracují s daty uspořádanými do čtverce o rozměrech 4 krát 4 byty (jelikož velikost bloku je 128 bitů, tak $128 : 8 = 4 \cdot 4 = 16$ bytů). Tento design je označován jako čtverec (v angličtině *SQUARE*) a jeho autory jsou právě Joan Daemen, Vincent Rijmen a Lars Knudsen [VAUDENAY, 2006]. Přes tyto vrstvy se projde 10krát, 12krát nebo 14krát a to v závislosti na tom, zdali je šifrovací klíč dlouhý 128, 192 nebo 256 bitů.

Mějme tedy blok dat otevřeného textu o velikosti 128 bitů. Proces šifrování se poté skládá z těchto kroků:

- přidání klíče nultého průchodu,
- 9, 11 či 13 průchodů (v závislosti na délce klíče) přes substituční vrstvu, rozptylovací vrstvu a vrstvu přidání klíče daného průchodu,
- průchod přes rozptylovací vrstvu,
- přidání klíče posledního průchodu.

Detailní pohled na jednotlivé vrstvy algoritmu AES ukazuje následující schéma:



Obrázek 11 – Schéma jednoho průchodu vrstvami algoritmu AES – vytvořeno na základě [PAAR, et al., 2010]

Než se pustím do popisu jednotlivých vrstev, je vhodné ukázat, jak vypadá ona čtvercová tabulka o rozměrech 4 krát 4 byty. Na 128 bitový blok dat je nutné nahlížet jako na posloupnost bytů $B_0, B_1, \dots, B_{15}$. Těchto 16 bytů je uspořádáno v tabulce následujícím způsobem:

Tabulka 17 – Tabulka uspořádání bytů šifrovaných dat při použití algoritmu AES [PAAR, et al., 2010]

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Obdobně jsou uspořádány také byty klíče. Taková tabulka má vždy čtyři řádky a, v závislosti na délce klíče, 4, 6 nebo 8 sloupců.

Substituční vrstva

Substituční vrstvu tvoří 16 nelineárních S-Boxů podobných těm použitým u algoritmu DES. V tomto případě však S-Box na vstupu očekává 8 bitů a výstupem je opět 8 bitů pozměněných podle substituční tabulky. Tato tabulka je stejná pro všech 16 S-Boxů a obsahuje hodnoty $0, 1, \ldots, 255$ v hexadecimální podobě rozmístěných v 16 řádcích a 16 sloupcích. Hodnota na vstupu S-Boxu se vyjádří taktéž v hexadecimální podobě, a skládá se tedy ze dvou znaků. První znak určuje číslo řádku substituční tabulky a druhý znak definuje číslo sloupce. Výstupem je poté hodnota na souřadnicích daných číslem řádku a sloupce.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Obrázek 12 – Substituční tabulka S-Boxu algoritmu AES[7]

Rozptylovací vrstva

Jak ukazuje Obrázek 11 výše, skládá se rozptylovací vrstva ze dvou podvrstev. Jejím úkolem je, jak už název napovídá, rozptýlit význam jednoho bytu do zbytku šifrovaných dat.

První na řadu přijde podvrstva posunu řádků. Její princip spočívá v tom, že se byty na druhém řádku posunou o jednu pozici doleva, na třetím řádku o dvě pozice doleva a na čtvrtém řádku o tři pozice doleva.

[7] Zdroj obrázku: http://edipermadi.files.wordpress.com/2008/03/sbox.png

Tabulka 18 – Tabulka bytů před průchodem a po průchodu podvrstvou posunu řádků – vytvořeno na základě [PAAR, et al., 2010]

Původní stav				Po průchodu podvrstvou posunu řádků			
B_0	B_4	B_8	B_{12}	B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}	B_5	B_9	B_{13}	B_1
B_2	B_6	B_{10}	B_{14}	B_{10}	B_{14}	B_2	B_6
B_3	B_7	B_{11}	B_{15}	B_{15}	B_3	B_7	B_{11}

Jako druhá se aplikuje podvrstva promíchání sloupců, která má nejzásadnější vliv na rozptýlení. V této podvrstvě se každý sloupec čtvercové tabulky maticově vynásobí s konstantní maticí M, jež má tvar

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

Pokud bychom uvažovali druhý sloupec čtvercové tabulky, jak ho ukazuje Tabulka 18 výše, vznikla by po průchodu podvrstvou promíchání sloupců čtvercová tabulka, jejíž druhý sloupec by vypadal následovně:

$$\begin{pmatrix} C_4 \\ C_5 \\ C_6 \\ C_7 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_4 \\ B_9 \\ B_{14} \\ B_3 \end{pmatrix}$$

Vrstva přidání klíče daného průchodu

V této vrstvě se ke zpracovávanému bloku dat připojí pomocí operace XOR klíč daného průchodu. Jeho generování má na starosti rozpis klíčů (Key Schedule).

Rozpis klíčů (Key Schedule)

Úkolem rozpisu klíčů je vygenerovat $n + 1$ klíčů pro jednotlivé průchody, kde n představuje 10, 12 či 14 průchodů v závislosti na délce klíče.

Klíče pro jednotlivé průchody jsou uloženy ve speciálním poli, jež má jednotlivé prvky o velikosti jednoho slova, což je v tomto případě 32 bitů. Toto pole má, opět v závislosti na délce klíče, 44, 52 nebo 60 prvků.

Ke generování klíčů dochází rekurzivně, a proto je k určení klíče K_i nezbytné znát klíč předchozí, tj. K_{i-1}. Stěžejní význam má při generování funkce g, která jednak používá substituční S-Boxy a také přidává ke zpracovávaným datům tzv. koeficient průchodu KP, což je osmibitová hodnota, která se pro každý klíč průchodu liší.

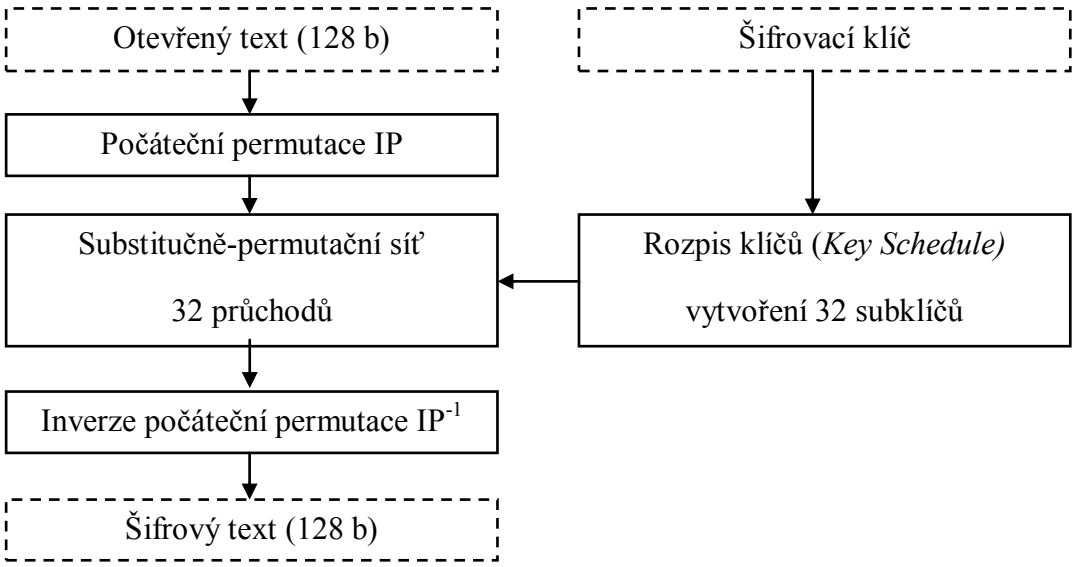
Nevýhody a slabiny

AES byl navržen tak, aby byl bezpečný nejen z pohledu útoku hrubou silou, ale také aby odolal diferenciální kryptoanalýze. Proto dodnes není znám žádný úspěšný útok na tento algoritmus.

Jedinou slabinou tak zůstává fakt, že se jedná o symetrický algoritmus, a proto je nutné nějakým způsobem zajistit bezpečný přenos šifrovacího klíče.

4.3.8 Serpent

Algoritmus Serpent vytvořili Ross Anderson, Eli Biham a Lars Knudsen. Serpent byl jejich kandidátem v soutěži o standard AES. Serpent nakonec skončil na druhém místě za algoritmem Rijndael, a proto je uveden i v této práci. Tento algoritmus je poskytován pod licencí GPL, která zajišťuje, že je jeho použití bezplatné a zdrojové kódy jsou volně dostupné. Stejně tak má veřejnost přístup k popisu algoritmu, jejž vytvořili sami autoři [ANDERSON, et al., 1998?].



Obrázek 13 – Schéma algoritmu Serpent

Podobně jako u algoritmu DES i zde neslouží počáteční permutace a její inverze k posílení kryptografické síly algoritmu, nýbrž k optimalizaci výkonu. To zmiňují i autoři algoritmu Serpent [ANDERSON, et al., 1998?].

Srdce algoritmu tvoří substitučně-permutační síť, kterou projde šifrovaný blok dat 32krát. Substitučně-permutační síť se skládá ze tří základních částí:

1) přidání klíče daného průchodu,
2) substituce v S-Boxech,
3) lineární transformace.

Přidání klíče daného průchodu

Klíč pro daný průchod, který je vytvořen dle rozpisu klíčů, je připojen ke zpracovávaným datům pomocí operace XOR.

Substituce v S-Boxech

S-Boxy algoritmu Serpent přijímají na vstupu data o velikosti 4 bitů a na výstupu poskytují data o stejné velikosti. Serpent disponuje celkem osmi rozdílnými S-Boxy, přičemž každý z nich je tvořen 16 hodnotami uspořádanými do jediného řádku. Podobu všech osmi S-Boxů lze najít v [ANDERSON, et al., 1998? str. 21].

V každém z průchodů se použije jeden z těchto osmi S-Boxů. Jelikož mají zpracovávaná data velikost 128 bitů, je nutné použít onen S-Box $128 : 4 = 32$ krát paralelně.

Lineární transformace

V této fázi se na šifrovaná data nahlíží jako na čtyři slova X_0, X_1, X_2, X_3, z nichž každé má velikost 32 bitů. Tyto čtyři slova se lineárně promíchají (v tomto pořadí):

- $X_0 = X_0 \lll 13$,
- $X_2 = X_2 \lll 3$,
- $X_1 = X_1 \oplus X_0 \oplus X_2$,
- $X_3 = X_3 \oplus X_2 \oplus (X_0 \ll 3)$,
- $X_1 = X_1 \lll 1$,
- $X_3 = X_3 \lll 7$,
- $X_0 = X_0 \oplus X_1 \oplus X_3$,
- $X_2 = X_2 \oplus X_3 \oplus (X_1 \ll 7)$,
- $X_0 = X_0 \lll 5$,
- $X_2 = X_2 \lll 22$.

V operacích uvedených výše značí ⋘ bitovou rotaci vlevo a ≪ bitový posun vlevo.

Výstupem je poté čtveřice slov X_0, X_1, X_2, X_3. Ta byla díky lineárním transformacím promíchána a bity jednotlivých slov se ovlivnily navzájem.

Rozpis klíčů (Key schedule)

Rozpis klíčů v tomto případě není tak přívětivý jako například u algoritmu DES. Ačkoliv lze zvolit šifrovací klíč o délce kratší než 256 bitů, je v takovém případě nejprve provedeno „doplnění" na délku 256 bitů. Toho se dosáhne tak, že se za nejvýznamnější bit přidá hodnota „1" a za ní se zbývající místa do 256 bitů doplní nulami.

V této chvíli se klíč dlouhý 256 bitů rozdělí na 8 slov o délce 32 bitů, která označíme jako $w_{-8}, \dots, w_{-1}$.

Pomocí nich lze vytvořit 132 tzv. „*přechodných klíčů*" [ANDERSON, et al., 1998?] o velikosti jednoho slova a to jednoduše tak, že se pro $i = 0, 1, \dots, 131$ dosadí do tohoto vztahu

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11$$

kde ⋘ je opět bitová rotace vlevo a ϕ je hexadecimální hodnota $0x9E3779B9$.

Nyní se využije S-Boxů (zmiňovaných výše), zde značených jako S, a vzniklých přechodných klíčů k vytvoření 132 slov, která označíme jako $k_0, \dots, k_{131}$ a která později poslouží k sestavení klíče pro daný průchod:

- $\{k_0, k_1, k_2, k_3\} = S_3(w_0, w_1, w_2, w_3)$,
- $\{k_4, k_5, k_6, k_7\} = S_2(w_4, w_5, w_6, w_7)$,
- $\{k_8, k_9, k_{10}, k_{11}\} = S_1(w_8, w_9, w_{10}, w_{11})$,
- $\{k_{12}, k_{13}, k_{14}, k_{15}\} = S_0(w_{12}, w_{13}, w_{14}, w_{15})$,
- $\{k_{16}, k_{17}, k_{18}, k_{19}\} = S_7(w_{16}, w_{17}, w_{18}, w_{19})$

...

- $\{k_{128}, k_{129}, k_{130}, k_{131}\} = S_3(w_{128}, w_{129}, w_{130}, w_{131})$.

Pakliže jsme získali všech 132 slov $k_0, \dots, k_{131}$, můžeme z nich konečně vytvořit klíč pro i-tý průchod K_i:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

Nevýhody a slabiny

Hlavní slabinou a nevýhodou tohoto algoritmu, je vysoký počet průchodů (celkem 32) při šifrování/dešifrování jednoho bloku dat. To má za následek mnohonásobně pomalejší šifrování a dešifrování než například algoritmus Rijndael. Rychlost šifrování tak byla kamenem úrazu, proč se Serpent nestal standardem AES [NECHVATAL, 2000].

4.3.9 Další blokové algoritmy

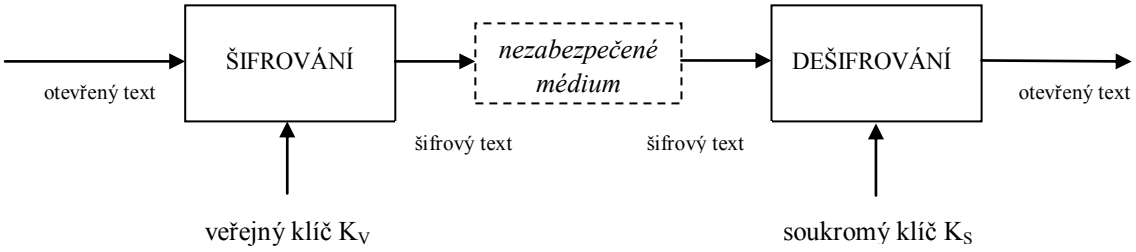
V předchozích čtyřech podkapitolách jsem zmínil nejvýznamnější zástupce blokových symetrických algoritmů. Bylo by však mylné předpokládat, že už do této skupiny žádné další algoritmy nepatří. Blokových algoritmů je velmi mnoho. Seznam blokových algoritmů lze najít například v [Block cipher, 2006].

5 Asymetrické šifrovací algoritmy

5.1 Důvod vzniku, princip

Do doby vzniku asymetrického šifrování byla největším problémem všech šifrovacích algoritmů distribuce klíčů. Jelikož se používal tentýž klíč jak k šifrování, tak i k dešifrování, bylo nezbytné ho bezpečným způsobem dopravit k oběma účastníkům, kteří spolu chtěli komunikovat. To je často velmi nákladný proces, což však nebyl až takový problém do doby, než narostl objem přenášených šifrovaných zpráv na neúnosné množství.

Proto se od začátku 60. let 20. století objevovaly snahy přijít s novým přístupem k šifrování, jenž by toto vyřešil. Vědci se inspirovali analogií z reálného světa: obyčejným mechanickým zámkem. Ten je totiž praktickým příkladem tzv. **jednocestné funkce se zadními vrátky**. Zaklapnout ho – a tím ho i zamknout – dokáže každý. Naopak odemknout ho může jedině ten, kdo má nějakou informaci navíc; v tomto případě správný klíč.



Obrázek 14 – Schéma šifrování pomocí asymetrických algoritmů

Funkce $f: X \to Y$ se nazývá **jednocestnou funkcí** právě tehdy, když:

- pro každé $x \in X$ je snadné spočítat hodnotu obrazu $y = f(x)$,
- pro každý náhodně zvolený obraz $y \in Y$ je výpočetně téměř nemožné najít $x \in X$ tak, že by platilo $f(x) = y$.

Funkce $f_k: X \to Y$ se nazývá **jednocestnou funkcí se zadními vrátky** $(\boldsymbol{k})$ právě tehdy, když:

- je jednocestnou funkcí,
- má takovou vlastnost, že při znalosti určité informace navíc (k), je pro každý obraz $y \in Y$ výpočetně snadné najít $x \in X$ takové, že $f_k(x) = y$, tedy $x = f_k^{-1}(y)$.

Z vlastností jednocestné funkce se zadními vrátky tedy vyplývá, že z její znalosti nelze získat její inverzní funkci. Pokud tento aspekt aplikujeme na případ asymetrické kryptografie, můžeme říci, že ze znalosti veřejného klíče je prakticky nemožné získat klíč privátní. Nebo také, že ze šifrového textu nelze získat otevřený text bez znalosti určité informace navíc – privátního klíče.

Celá asymetrická kryptografie je založena právě na principu jednocestných funkcí. To má za následek, že je nutné používat oproti symetrické kryptografii dva rozdílné klíče:

- **soukromý (privátní) klíč** – tajný klíč, jejž si majitel uchovává pro sebe a v tajnosti a který slouží k dešifrování zprávy,
- **veřejný klíč** – veřejný klíč, jejž majitel může bez obav zveřejnit a který ostatní uživatelé využijí k zašifrování zprávy.

Díky existenci veřejného klíče se také asymetrická kryptografie někdy nazývá „kryptografie s veřejným klíčem“ (v angličtině *public key cryptography*) [VAUDENAY, 2006]

5.2 Historický pohled

Za tvůrce nápadu asymetrického šifrování jsou považováni tři Američané – kryptograf Whitfield Diffie, profesor Stanfordovy univerzity Martin Hellman a Ralph Merkle. Ti se zaměřili na hledání jednosměrných funkcí, jež by byly vhodné pro asymetrické šifrování. V roce 1975 se jim podařilo pomocí modulární aritmetiky vyřešit problém bezpečné výměny klíčů přes nezabezpečený kanál. Svoje řešení představili rok nato, v listopadu 1976 [DIFFIE, et al., 1976]. I přesto, že se jednalo o těžkopádný a nedokonalý způsob, způsobilo jejich řešení obrovský rozruch a odstartovalo boom asymetrické kryptografie [SINGH, 2009].

Obrázek 15 – Ralph Merkle, Martin Hellman, Whitfield Diffie[8]

S použitelným asymetrickým šifrovacím algoritmem přišli až jiní Američané: Ronald Rivest, Adi Shamir a Leonard Adleman. Tato trojice matematiků vymyslela v roce 1977 algoritmus založený na modulární aritmetice, jenž dostal jméno podle prvních písmen příjmení svých tvůrců: **RSA** [MURPHY, et al., 2006], [SINGH, 2009]. Tomuto dominantnímu algoritmu asymetrické kryptografie se podrobně věnuje další kapitola.

[8] Zdroj obrázku: http://www.livinginternet.com/g/diffie_hellman_merkle.jpg

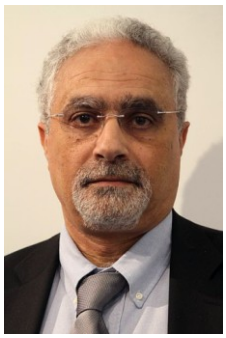
Ačkoliv jsou zásluhy za vznik asymetrické kryptografie připisovány právě zmiňovaným šesti Američanům, jiní tvůrci přišli s tímto principem již o několik let dříve.

Byli jimi James Ellis, Clifford Cocks a Malcom Williamson z anglického Government Communications Headquarters, kteří nezávisle na amerických kolezích navrhli stejné principy a algoritmy. Jelikož ale jejich práce podléhala přísnému utajení a mlčenlivosti, nemohl se svět o jejich objevu dozvědět. Podrobněji se těmito historickými souvislostmi zabývá například [SINGH, 2009].

Od 80. let 20. století se objevovaly ještě další asymetrické algoritmy, z nichž jen některé byly přijaty širokou veřejností. Mezi ně patří především algoritmy **ElGamal**, jenž získal jméno po svém objeviteli, egyptském kryptografovi, Dr. Taherovi Elgamalovi, či **ECC** (z anglického názvu *Elliptic Curve Cryptography*), kryptografie eliptických křivek, s níž přišli nezávisle na sobě američtí matematici Neal Koblitz a Victor Saul Miller. Oba tyto algoritmy spatřily světlo světa roku 1985 a jsou založeny na jiném principu než RSA.

5.3 ElGamal

Asymetrický algoritmus ElGamal dostal jméno po svém objeviteli, egyptském kryptografovi Dr. Taherovi Elgamalovi, jenž v roce 1985 [ELGAMAL, 1985] ve svém postupu opustil problém faktorizace a zaměřil se na problém spočítání diskrétního logaritmu. Kromě výše zmiňovaného zdroje se fungování tohoto algoritmu věnují například [VAUDENAY, 2006] či [WOBST, 2007].



Obrázek 16 – Dr. Taher Elgamal[9]

Algoritmus ElGamal má však oproti RSA jednu nevýhodu, jež ale není až takovou překážkou, zvážíme-li, že se asymetrické algoritmy používají především k šifrování klíčů pro symetrické algoritmy. Zašifrovaná data pomocí tohoto algoritmu mají totiž dvojnásobnou velikost, což je důsledek použití právě diskrétního logaritmu.

[9] Zdroj obrázku: http://upload.wikimedia.org/wikipedia/commons/thumb/5/5c/Taher_Elgamal_it-sa_2010.jpg/225px-Taher_Elgamal_it-sa_2010.jpg

ElGamal patří mezi takzvané pravděpodobnostní šifrovací algoritmy. To znamená, že stejný otevřený text může být zašifrován pomocí stejného šifrovacího klíče do rozdílné podoby šifrového textu [PAAR, et al., 2010 str. 229].

Princip fungování

Definujme nejprve relaci **kongruence** [KALA, 2004]:

- Mějme celá čísla a a b a přirozené číslo n. Pokud platí $n \mid (a - b)$, řekneme, že čísla a a b jsou kongruentní podle modulu n a zkráceně tento vztah píšeme jako $a \equiv b \ (mod\ n)$.

Jelikož princip spočívá ve spočtení diskrétního logaritmu, je vhodné ho definovat:

- Nechť existují přirozená čísla $a, x, n, Y \in \mathbb{N}$, pro něž platí: $Y \equiv a^x \ mod\ n$. Potom každé číslo x, které odpovídá dané rovnici, nazýváme **diskrétním logaritmem** o základu a vzhledem k modulu n.

Nyní je možné zjednodušeně popsat kroky potřebné k volbě klíče, šifrování a dešifrování.

Volba klíče

1) Je nutné zvolit veřejné parametry: základ $a \in \mathbb{N}$ a dostatečně velké prvočíslo $m \in \mathbb{N}$ o délce minimálně 1024 bitů a to tak, aby i číslo $\frac{(m-1)}{2}$ bylo prvočíslem. Tato čísla jsou známa všem účastníkům komunikace.
2) Vygeneruje se náhodné číslo x, které se použije jako soukromý klíč.
3) Poté se spočítá zbytek po dělení $Y = a^x \ mod\ m$.
4) Trojice čísel (a, m, Y) pak tvoří veřejný klíč. Soukromý klíč tvoří exponent x.

Šifrování

1) Před šifrováním otevřeného textu o je nutné zvolit náhodné číslo k, které je nesoudělné s číslem $(m - 1)$.
2) Poté se přistoupí k samotnému šifrování, během kterého vzniknou dvě čísla c, d (odtud dvojnásobná délka šifrového textu):
   a. $c = a^k \ mod\ m$,
   b. $d = (Y^k \cdot o) \ mod\ m$.
3) Čísla (c, d) tvoří šifrový text.

Dešifrování

1) K dešifrování postačí klíč x, pomocí kterého lze vyřešit tuto rovnici:

- $c^x \cdot m = d \bmod p$,

kde m je otevřeným textem. K snadnému vyřešení se dá použít modulární multiplikativní inverze, například pomocí rozšířeného Euklidova algoritmu, která je blíže popsána v další kapitole o RSA.

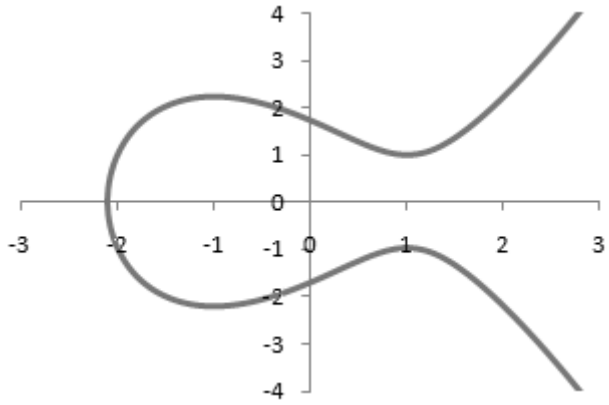
5.4 ECC

Zkratka ECC pochází z anglických slov Elliptic Curve Cryptography a do češtiny se dá přeložit jako „kryptografie založená na teorii eliptických křivek“. Vhodnost použití eliptických křivek pro potřeby asymetrické kryptografie navrhli nezávisle na sobě američtí matematici Neal Koblitz [KOBLITZ, 1987] a Victor Saul Miller [MILLER, 1986] v roce 1985.

Princip

Na začátek je vhodné podívat se na definici eliptické křivky:

Eliptickou křivkou nad tělesem K rozumíme takovou množinu řešení $(x, y) \in K^2$, která vyhovuje rovnici $y^2 = x^3 + ax + b$, kde $a, b \in K$.



Obrázek 17 – Eliptická křivka o rovnici $y^2 = x^3 - 3x + 3$

Kryptografie na principu eliptických křivek stojí a padá, podobně jako algoritmus Elgamal, na problému spočtení diskrétního logaritmu. V tomto případě se však jedná o ještě složitější problém – spočtení diskrétního logaritmu eliptických křivek, konkrétně $Q = dP$, kde Q je bod konečného tělesa a slouží jako veřejný klíč, P je bod konečného tělesa a d je prvočíslo dané bitové délky sloužící jako soukromý klíč. Obdobně, jak tomu u asymetrické kryptografie bývá, spočtení hodnoty soukromého klíče d ze znalosti veřejného klíče Q a veřejně dostupného bodu konečného tělesa P je výpočetně extrémně náročné a neexistují žádné algoritmy, které by tento problém dokázaly vyřešit. Jak poznamenává sám

Neal Koblitz, *„nelze použít ani známé algoritmy řešící problém diskrétního logaritmu u algoritmu ElGamal“* [KOBLITZ, 1987 str. 206].

Stejně jako u ostatních šifrovacích algoritmů sestává šifrovací proces ze tří částí: tvorby klíčového páru, šifrování a dešifrování. Jejich popsání je však nad rámec této práce, zájemci mohou nalézt podrobnější informace například v [KOBLITZ, 1987] nebo [MILLER, 2007], z nichž bylo čerpáno v předcházejících řádcích.

Výhody a nevýhody

ECC nabízejí oproti jiným asymetrickým algoritmům nesporné výhody. Tou největší je bezpochyby **menší velikost klíče** při zachování stejného stupně bezpečnosti, jakou nabízí šifrovací algoritmus RSA s mnohem delším klíčem. S tím jde ruku v ruce **výkon a rychlost prováděných operací**, což se nejvíce projeví například v případě mobilních či embedded zařízení. Další výhodou je to, že rovnice eliptických křivek obsahují značné množství parametrů. Tím pádem mají tvůrci kryptosystémů poměrně volnou ruku při implementaci ECC a mohou velmi přesně navrhnout strukturu tak, aby odpovídala jejich představám.

Naopak nevýhodou této metody je poměrně **složité generování klíčového páru** a taktéž **volba správné eliptické křivky**. Na volbě eliptické křivky záleží celková bezpečnost algoritmu, a tak je nezbytné volit vhodné parametry [PAAR, et al., 2010 str. 250]. Jelikož se jedná o poměrně mladou šifrovací metodu, není ještě dostatečně prověřená časem, a to je v kryptografii jednoduše nevýhoda.

6 RSA

6.1 Stručný pohled na RSA

Již v předchozí kapitole věnované asymetrickému šifrování byla o algoritmu RSA zmínka. Protože tato práce věnuje tomuto algoritmu zvýšenou pozornost, tak budou na tomto místě některé informace zopakovány, aby tak byla tato kapitola úplná.

Informace o RSA lze najít téměř v každé knize o šifrování, jelikož se jedná o nejstarší a nejpopulárnější algoritmus asymetrické kryptografie. Informace v této šesté kapitole byly čerpány z [COBB, 2004], [DENIS, et al., 2007], [PAAR, et al., 2010], [SINGH, 2009], [VAUDENAY, 2006], [WOBST, 2007].

Algoritmus RSA vzniknul v roce 1977, necelé dva roky po uvedení článku [DIFFIE, et al., 1976] trojice Diffie-Hellman-Merkle. Jeho tvůrci jsou američtí matematici Ronald Rivest, Adi Shamir a Leonard Adleman, po nichž získal algoritmus jméno – jedná se o zkratku z prvních písmen jejich příjmení. RSA se stal nejpopulárnějším asymetrickým algoritmem, za což může především to, že ho lze použít jak pro šifrování, tak také pro digitální podepisování dokumentů, a zároveň se pomocí tohoto algoritmu nezvětšuje velikost šifrového textu.

Obrázek 18 – Ronald Rivest, Adi Shamir, Leonard Adleman[10]

[10] Zdroj obrázku: http://www.ulm.ccc.de/old/chaos-seminar/krypto2/rsa.jpg

V praxi se RSA používá především v kombinaci se symetrickými algoritmy a to proto, že symetrické algoritmy jsou mnohonásobně rychlejší. V neprospěch RSA hraje množství výpočtů, které je nutné vykonat, a jejich složitost. RSA proto slouží v procesu šifrování především k bezpečnému přenosu klíčů pro symetrické šifrování, zatímco zpráva jako taková je zašifrovaná právě zvoleným symetrickým klíčem.

Zajímavostí je, že celé čtyři roky před objevem RSA přišel s algoritmem založeným na stejném principu Clifford Cocks z britského GCHQ. O tom se svět dozvěděl až v roce 1997, protože do té doby podléhala práce v GCHQ přísnému utajení [SINGH, 2009].

6.2 Princip algoritmu

6.2.1 Úvod do fungování

Algoritmus RSA se zakládá na jednocestné funkci se zadními vrátky a problému **faktorizace**. Faktorizací se rozumí matematický problém rozložení celého čísla na součin prvočísel. Tento problém je velmi složitý, na čemž se shodují všichni autoři [COBB, 2004], [DENIS, et al., 2007], [MURPHY, et al., 2006], [PAAR, et al., 2010], [SINGH, 2009], [VAUDENAY, 2006], [WOBST, 2007].

Vynásobit dvě prvočísla je velmi triviální úkol. Naopak rozložit (faktorizovat) náhodné číslo na dva činitele, jež by po vynásobení daly ono číslo, je úloha velmi výpočetně náročná. Zvlášť, pokud se jedná o extrémně velká čísla – v řádu několika stovek číslic. RSA v současnosti používá klíč o velikosti minimálně 1024 bitů, což je pro představu číslo v desítkové soustavě o 309 číslicích.

V současnosti navíc neexistuje žádný efektivní algoritmus, který by dokázal rozložit extrémně velká čísla na součin prvočísel v přijatelném čase. A právě na tuto skutečnost spoléhá RSA.

Pro snadnější představu uvedu jednoduchý příklad: Pokud předložím číslo 25957 a řeknu, že je vytvořeno součinem dvou prvočísel, je velmi těžké zjistit, která to jsou. Člověk (či počítačový program) by musel číslo 25957 postupně dělit čísly $2, 3, \ldots \lfloor\sqrt{n}\rfloor$ a sledovat zbytek po dělení, jestli se rovná nule. Po nějaké době by se našel výsledek, první hledané prvočíslo. Druhé se dá snadno dopočítat pomocí dělení, čímž se dostaneme k oběma hledaným prvočíslům – jsou jimi 101 a 257.

Naopak, pokud bych předložil čísla 101 a 257 a chtěl získat jejich součin, je to úloha naprosto triviální.

6.2.2 Bližší pohled na fungování RSA

Než se pustím do podrobnějšího popisu algoritmu RSA, je nutné definovat několik matematických pojmů:

Prvočíslo je takové přirozené číslo $n \in \mathbb{N}$, pro které platí, že je beze zbytku dělitelné pouze dvěma různými přirozenými čísly a to číslem 1 a sebou samým.

Největší společný dělitel (označovaný **NSD**) dvou celých čísel je největší číslo takové, že pro něj platí, že dělí obě dvě čísla beze zbytku:

- $NSD(a,b) = \max\{n \in \mathbb{N}: n \mid a \text{ a zároveň } n \mid b\}$

Modulární multiplikativní inverze celého čísla e modulo n je definována jako číslo d, pro které platí:

- $e \cdot d \equiv 1 \; mod \; n$

Modulární multiplikativní inverze existuje jen a pouze v tom případě, že $NSD(e,n) = 1$, tj. že obě dvě čísla jsou nesoudělná.

Eulerova funkce $\varphi(n)$, kde $n \geq 2$, je definována jako počet čísel nesoudělných s n.

Eulerova věta označuje tvrzení, že pro každé přirozené číslo $n \in \mathbb{N}$ a přirozené číslo $a \in \mathbb{N}$ nesoudělné s n platí: $a^{\varphi(n)} \equiv 1 \; mod \; n$.

Po definování potřebných matematických pojmů se již můžeme blíže podívat na algoritmus RSA, který se skládá ze tří kroků (podobně jako tomu je u jiného asymetrického algoritmu ElGamal):

- vygenerování dvojice klíčů,
- šifrování,
- dešifrování.

Vygenerování dvojice klíčů

Vygenerování dvojice klíčů – privátního a veřejného – je nejdůležitější částí algoritmu RSA, protože má největší a zásadní vliv na jeho správné fungování. Špatná volba klíče, především co se jeho délky týče, má za následek rapidní snížení bezpečnosti šifry a může vést i k jejímu snadnému prolomení násilnou cestou.

1) Prvním krokem je **zvolení dvou velkých prvočísel** p a q přibližně stejné velikosti a to tak, že jejich součin, číslo $n = p \cdot q$, bude mít požadovanou bitovou délku, například 2048 bitů.

2) Dále je nutné **spočítat hodnoty** $n = p \cdot q$ a $\phi(n) = phi = (p-1)(q-1)$, kde phi je počet nesoudělných čísel s n získaný pomocí Eulerovy funkce.

3) Dalším krokem je **volba celého čísla** e, pro které platí:

   - $1 < e < phi$,
   - $NSD(e, phi) = 1$, tedy e a phi jsou nesoudělná čísla.

Volba čísla e má vliv na rychlost šifrování, a tak se často volí jedno z čísel $3, 17, 65537$. Jedná se o Fermatova čísla, která mají ve svém dvojkovém vyjádření nastavené jen dva bity na hodnotu „1“, a tím pádem se snižuje složitost šifrovacího algoritmu z kubické složitosti $O(n^3)$ na kvadratickou $O(n^2)$, což zmiňuje Vaudenay [VAUDENAY, 2006 str. 238].

4) **Spočtení privátního klíče** d, pro který platí:
   - $1 < d < phi$,
   - $e \cdot d \equiv 1 \; mod \; phi$.

Jak je vidět z výše uvedeného, zde lze využít Eulerovy věty k získání modulární multiplikativní inverze.

V počítačové implementaci je neúnosné, aby se hodnota d určovala například zkoušením hodnot od $d = 2$ až do doby, než se nalezne taková vyhovující rovnici výše. Velmi efektivně se dá použít například rozšířený Euklidův algoritmus. Jeho vysvětlení je však nad rámec této práce, proto odkazuji čtenáře například na [The Euclidean Algorithm and the Extended Euclidean Algorithm, 2010] či [MENEZES, et al., 1996].

5) Nyní jsou již vytvořené **oba dva klíče**:
   - veřejný klíč tvoří hodnoty (n, e),
   - soukromý klíč tvoří hodnoty (n, d).

Zašifrování zprávy

Po vygenerování dvojice klíčů již nic nebrání tomu, aby se mohlo přistoupit k samotnému šifrování. Uživatel nyní může zveřejnit svůj veřejný klíč, aby mohl kdokoliv zašifrovat svoji zprávu pro toho uživatele.

1) Prvním krokem v šifrování je tedy pochopitelně **získání veřejného klíče** osoby, které je zpráva určena.
2) Dalším krokem je **převod otevřeného textu** na přirozené číslo $o \in \mathbb{N}: 0 < o < (n - 1)$.[11]
3) Nyní dochází k samotnému šifrování a **vzniku šifrového textu** s a to pomocí vzorce $s = o^e \; mod \; n$.

[11] V praxi je tento proces mnohem složitější, o čemž se zmiňuji v další podkapitole.

Dešifrování zprávy

K dešifrování zprávy je pochopitelně nutné mít příslušný soukromý klíč.

1) Majitel soukromého klíče může šifrový text s převést zpět na otevřený text o provedením operace $o = s^d \ mod \ n$.

6.3 Použití RSA v praxi

Prvním problémem je to, jakým způsobem by měl být otevřený text převeden na číselnou hodnotu. Především u delších otevřených textů či velkých souborů je to velmi problematické a někdy i vyloženě nemožné, a proto se RSA tímto způsobem nepoužívá.

Další nevýhodou RSA je to, že se jedná o výpočetně velice náročný algoritmus. To je dalším důvodem, proč se v praxi běžně neděje, že by se tento algoritmus použil na celý otevřený text (či soubor a tak dále). Místo toho se požadovaný otevřený text zašifruje pomocí některého ze symetrických algoritmů, které jsou mnohem rychlejší (viz následující kapitola).

RSA algoritmus je následně použit až pro zašifrování použitého klíče symetrického algoritmu. Jeho zašifrovaná podoba se poté může bezpečně zaslat příjemci společně se šifrovým textem vzniklým pomocí symetrického šifrování.

6.4 Délka klíče

Délka klíče je jednou ze zásadních vlastností, které přímo ovlivňují bezpečnost RSA šifrování. Od letošního roku 2011 je doporučováno společností RSA Laboratories, která zastřešuje vývoj algoritmu RSA, používat klíč přinejmenším o bitové délce 2048 bitů (jak uvádí Tabulka 19 uvedená níže). Klíč o bitové délce 1024 bitů již není považován za bezpečný, nicméně pro běžnou nebankovní, nevládní a nearmádní výměnu zpráv je jeho délka stále adekvátní.

Zajímavou vlastností klíče je tzv. dosažená bitová bezpečnost, jež určuje, kolik operací je minimálně nutné provést k prolomení RSA algoritmu za použití daného klíče.

Pokud má určitá délka klíče bitovou bezpečnost např. 86 bitů, znamená to, že na prolomení šifry je nutné vykonat $2^{86} \sim 7{,}73 \cdot 10^{25}$ operací.

Tabulka 19 – Doporučená délka klíče RSA [RSA Laboratories, 2003], [DENIS, et al., 2007 str. 389]

	Do roku 2010	***Do roku 2030***	***2031 a dále***
Minimální délka RSA klíče	1024 bitů	2048 bitů	3072 bitů
Dosažená bitová bezpečnost	86	116	138

6.5 Další použití RSA

Kromě šifrování dat se dá algoritmus RSA použít také pro ověření identity pomocí digitálního podpisu či ověření integrity přenášených dat.

6.5.1 Digitální podpis

Pro uplatnění RSA při digitálním podepisování lze využít tzv. hashovací funkce [DENIS, et al., 2007], [WOBST, 2007].

Hashovací funkcí rozumíme takovou matematickou funkci, která ze zadaných vstupních dat libovolné délky vytvoří výstup o pevné, fixní délce. Tento výstup nazýváme *otisk*.

Pro použití v kryptografii však po hashovací funkci požadujeme ještě další vlastnosti navíc:

1) hashovací funkce h musí být jednocestnou funkcí, a tedy pro obraz hashovací funkce y je prakticky nemožné najít takovou hodnotu x, aby platilo $y = h(x)$,

2) hashovací funkce h musí být *odolná kolizím* ([VAUDENAY, 2006] zmiňuje, že tato vlastnost funkce se často mylně označuje jako „*bezkolizní*“, což je ale zavádějící, jelikož kolize existují).

Kolizí se rozumí takový případ, kdy pro dvě rozdílné hodnoty $x \neq z$ vstupních dat vzejdou z hashovací funkce dva stejné otisky $h(x) = h(z)$.

Hashovacích kryptografických algoritmů existuje hned několik, ale zdaleka ne všechny jsou považovány za bezpečné, jak zmiňuje například Wobst [WOBST, 2007].

MD4 a **MD5** (z anglického „*Message Digest*“) jsou hashovací kryptografické algoritmy, za nimiž stojí Ron Rivest a které vznikly na začátku 90. let. Oba však byly shledány jako nedostačující, jelikož „*kolize mohou být nalezeny dokonce za použití tužky a papíru*“ [WOBST, 2007 str. 339].

Do druhé velké rodiny hashovacích kryptografických algoritmů patří **SHA-0**, **SHA-1**, **SHA-224**, **SHA-256**, **SHA-384** a **SHA-512** (z anglického „*Secure Hash Algorithm*“), na jejichž vývoji se podílely americké instituce NIST a NSA [DENIS, et al., 2007]. Už na konci 90. let byly první dvě verze – SHA-0 a SHA-1 – považovány za nevyhovující, jelikož u nich byly objeveny a prokázány kolize. Proto v srpnu 2002 přišel americký úřad pro patenty se zmodernizovanou verzí SHA se čtyřmi možnými délkami výstupu.

Denis [DENIS, et al., 2007 str. 205] zmiňuje, že co se výskytů kolizí týče, tak u posledních čtyř zmiňovaných algoritmů dochází k takzvanému „*narozeninovému paradoxu*“. Z něho vyplývá, že k nalezení kolize je dostačujících $2^{\frac{n}{2}}$ operací, kde n je délka výstupu hashovacího algoritmu. To ukazuje následující tabulka:

Tabulka 20 – Srovnání bezpečnosti algoritmů SHA [DENIS, et al., 2007]

	SHA-224	**SHA-256**	**SHA-384**	**SHA-512**
Délka výstupu v bitech	224	256	384	512
Bitová bezpečnost	112	128	192	256

Postup ověření digitálního podpisu

Odesílatel dat, jenž chce data digitálně podepsat, vygeneruje pomocí hashovací funkce otisk h odesílaného šifrového textu st a zašifruje ho pomocí svého soukromého klíče (d, n): $x = h^d \ mod \ n$. Tento zašifrovaný otisk pošle spolu se zprávou.

Právě odesílatel dat je jedinou osobou, která dokáže zašifrovat otisk tak, aby po použití veřejného klíče vznikl správný otisk. To je dáno tím, že jen odesílatel zná konkrétní soukromý klíč.

Příjemce poté použije veřejný klíč (e, n) odesílatele na zašifrovaný otisk x: $y = x^e \ mod \ n$. Nyní vygeneruje otisk h_2 z přijaté zprávy a porovná ho s hodnotou y:

- pokud $y = h_2$, pak je podpis pravý,
- pokud $y \neq h_2$, pak je podpis falešný.

6.5.2 Integrita dat

Kompletně stejným způsobem lze za pomocí otisku hashovací funkce ověřit i integritu přenášených dat.

7 Porovnání symetrických a asymetrických algoritmů

Jednotlivé přístupy (symetrický versus asymetrický) byly nastíněny v předešlých třech kapitolách a nyní je čas shrnout jednotlivé výhody a nevýhody obou přístupů z hlediska výkonu, paměťové náročnosti, bezpečnosti a dalších aspektů.

7.1 Délka klíče

Abychom dosáhli u obou přístupů k šifrování stejné bezpečnosti, je nutné zvolit klíče rozdílných bitových délek. Klíče používané v asymetrické kryptografii jsou pak nesrovnatelně delší než ty užívané symetrickými algoritmy. Na tomto místě je dobré připomenout, že čím kratší klíč, tím lépe, jelikož zabírá méně místa v paměti, algoritmy mohou pracovat rychleji atd. (viz dále). Zajímavé je srovnání délky klíčů, při které dosahují jednotlivé způsoby stejné bezpečnosti. To ukazuje tato tabulka sestavená na základě dat od amerického úřadu NIST:

Tabulka 21 – Srovnání délky klíčů [BARKER, et al., 2007]

Bitová bezpečnost	Symetrické algoritmy	RSA	Elgamal	ECC
80	80	1024	1024	160
112	112	2048	2048	224
128	128	3072	3072	256
192	192	7680	7680	384
256	256	15360	15360	512

Z tabulky jednoznačně vyplývá výše zmíněné, tedy že k dosažení stejné bezpečnosti je u symetrických algoritmů možné zvolit několikanásobně kratší klíče než u asymetrických algoritmů – až na výjimku šifrování založeného na principu eliptických křivek, jež je jediný způsob asymetrického šifrování schopný konkurovat délkou svých klíčů symetrickému přístupu.

7.2 Výkon

Na výkon jednotlivých algoritmů má velký vliv právě také fakt, zdali se jedná o symetrické či asymetrické algoritmy. Ty patřící do skupiny symetrických algoritmů jsou všeobecně rychlejší, protože využívají například operaci XOR, zatímco asymetrické algoritmy jsou založeny na mnohem složitějších matematických základech. Asymetrické algoritmy často vyžadují výpočet zbytku po dělení, modulární inverzi či například umocňování extrémně vysokých čísel.

Jak zmiňuje Christof Paar, *„je běžné, že asymetrické algoritmy jsou 100-1000krát pomalejší než symetrické algoritmy,“* [PAAR, et al., 2010 str. 156].

Neblahý vliv na výkon asymetrických algoritmů má také to, že používají mnohonásobně delší klíče. Paar toto upřesňuje: *„S rostoucí délkou klíče roste také doba nutná pro šifrování a to s třetí mocninou,“* [PAAR, et al., 2010 str. 157]. To znamená, že pokud místo klíče o délce 512 bitů použijeme klíč dlouhý 2048 bitů, bude libovolný asymetrický algoritmus v případě delšího klíče až $\left(\frac{2048}{512}\right)^3 = 4^3 = 64$krát pomalejší než s oním kratším klíčem.

Ruku v ruce s delšími klíči jde také paměťová náročnost, kdy asymetrické algoritmy potřebují při práci více paměti než algoritmy symetrické.

7.3 Bezpečnost

Pokud se nyní pustím do porovnání symetrických a asymetrických algoritmů z hlediska bezpečnosti, nemám tím na mysli, jak je která skupina algoritmů náchylná k prolomení. V kapitole 1.2.1 se zmiňuji, že existují tři cíle, na něž se kryptografie jako věda soustředí. Těmi cíli jsou *důvěrnost*, *integrita* a *autentifikace*. A právě integrity dat a autentifikace se dá dosáhnout pomocí asymetrické kryptografie, kdežto pomocí symetrické kryptografie nikoliv. Pomocí symetrické kryptografie je možné dosáhnout pouze integrity dat. Důvěrnosti se poté dá pochopitelně dosáhnout u obou přístupů.

8 Budoucnost kryptografie

Odhadovat budoucnost, a především v oboru informačních technologií, je velmi ošemetná záležitost, protože jeden objev může odstartovat celou řadu dalších, v současné době nemyslitelných, možností. Přesto se zraky kryptologů upírají směrem ke **kvantovým počítačům** a kvantové kryptografii, která dle předpokladů a odhadů dokáže vyřešit všechny dosavadní problémy a nedostatky, čímž by vznikla možnost dokonalého, nerozluštitelného a hlavně použitelného šifrování.

8.1 Kvantová kryptografie

8.1.1 Úvod

Kvantová kryptografie však stojí a padá s konstrukcí kvantových počítačů. Jejich vývoj je však ve stádiu výzkumu a testování, na nichž nezávisle na sobě pracují nejlepší vědecké týmy; například z MIT, CALTECH či QUIC. Posledně jmenovaný tým navíc podporuje americká armádní organizace DARPA.

Bohužel princip kvantového počítače je nad rámec této práce. Zájemci o podrobnější informace o jeho fungování je mohou najít například v [KUPČA, 2001].

První protokol kvantové kryptografie spatřil světlo světa již roku 1984. Vymysleli ho Američan Charles H. Bennett a Kanaďan Gilles Brassard, po nichž a roku objevení protokol dostal své jméno: **BB84** [BENNETT, et al., 1984]. Protokol jako takový slouží k bezpečné výměně šifrovacího klíče přes nezabezpečené médium a detailně se mu věnují například [SINGH, 2009] či [WOBST, 2007].

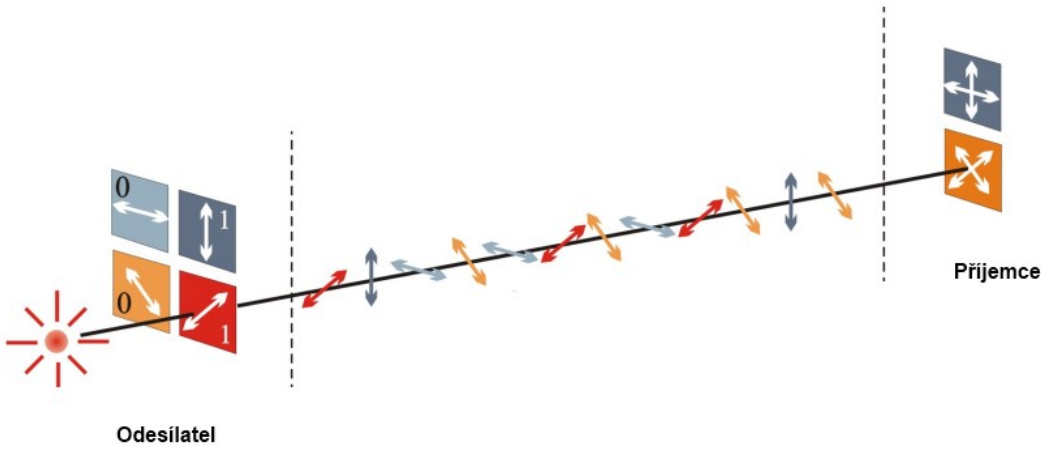
8.1.2 Protokol BB84

Kvantový šifrovací protokol BB84 používá pro přenos informace emitované fotony, které vibrují v jednom ze čtyř možných směrů, tedy jsou v daném směru polarizovány: vertikálně, horizontálně, diagonálně zleva doprava a diagonálně zprava doleva. Každý z těchto směrů, v závislosti na zvolené bázi, reprezentuje jinou bitovou hodnotu, jak ukazuje Tabulka 22 níže:

Tabulka 22 – Volba bitové hodnoty v závislosti na polarizaci fotonu – vytvořeno na základě [SINGH, 2009]

Polarizace	*Báze*	*Bitová hodnota*
↕	+	1
↔	+	0
⤢	×	1
⤡	×	0

K měření slouží dva polarizační filtry, jež kopírují tvar báze: buď +, nebo ×. Z kvantové mechaniky platí, že polarizační filtr typu + dokáže spolehlivě změřit jen fotony orientované horizontálně či vertikálně. Naopak diagonálně orientované fotony tímto filtrem projdou, nicméně změní svoji polarizaci. Obdobně je tomu u filtru typu ×. Ten dokáže spolehlivě změřit jen fotony polarizované diagonálně; horizontálně či vertikálně orientované fotony jím projdou, ale taktéž změní svoji polarizaci.



Obrázek 19 – Schéma komunikace pomocí protokolu BB84[12]

Jak jsem psal výše, pokud se k měření fotonu použije špatný polarizační filtr, dochází k jevu, že daný foton změní směr vibrování. Začne vibrovat v jednom ze směrů v závislosti na tvaru polarizačního filtru. Například při měření fotonu vibrujícího horizontálně (↔) pomocí polarizačního filtru typu × můžou nastat tyto dvě situace:

1) foton filtrem projde a změní se jeho polarizace na ↘,
2) foton filtrem projde a změní se jeho polarizace na ↙.

To má v důsledku vliv na úspěšné určení bitové hodnoty. Pokud použijeme správný polarizační filtr, není co řešit – foton změříme vždy správně, a tím pádem i jeho bitovou hodnotu určíme správně.

[12] Zdroj obrázku: http://swissquantum.idquantique.com/IMG/jpg/bb84.jpg

Pokud však zvolíme špatný filtr, může opět dojít k těmto dvěma situacím:

1) foton filtrem projde a změní polarizaci na typ, který vyjadřuje stejnou bitovou hodnotu, jakou měl původní foton. I přes špatný filtr jsme určili **bitovou hodnotu správně**,

2) foton filtrem projde a změní polarizaci na typ, který nevyjadřuje stejnou bitovou hodnotu, jakou měl původní foton. **Bitovou hodnotu tak určíme špatně**.

8.1.3 BB84 a stanovení šifrovacího klíče

Po stručném vysvětlení, na jakých kvantových principech protokol BB84 stojí, již nic nebrání vysvětlení způsobu stanovení šifrovacího klíče, což je vlastně hlavní úlohou tohoto protokolu:

1) Odesílatel začne vysílat náhodnou sekvenci fotonů, přičemž zcela náhodně střídá báze + a ×. Tato sekvence by měla být dvakrát delší než je požadovaná délka šifrovacího klíče, jelikož je zde jen 50% pravděpodobnost, že příjemce a odesílatel zvolí oba stejnou bázi.

2) Příjemce netuší, která báze byla u konkrétního fotonu zvolena, a tak – taktéž zcela náhodně – báze střídá a zapisuje si bitové hodnoty pro dané fotony.

    a. Pokud příjemce zvolí stejnou bázi jako odesílatel, bude zapsaná bitová hodnota 100% správná.

    b. V případě, že příjemce nezvolí stejnou bázi jako odesílatel, může, ale nemusí, být daná bitová hodnota správná.

3) Po skončení přenosu se příjemce a odesílatel spojí, klidně přes nezabezpečený kanál. Příjemce pro každý odeslaný foton uvede, jakou použil bázi – zdali typ +, či ×. Bitové hodnoty těch fotonů, u nichž i odesílatel použil stejnou bázi, jsou zvoleny jako bity tvořící šifrovací klíč, jelikož je příjemce se 100% jistotou správně určil. Ostatní bity, u nichž byla zvolena špatná báze, nejsou v tuto chvíli důležité (protože 50 % z nich je chybně interpretovaných).

4) Nyní odesílatel zvolí několik desítek bitů, které poslouží k ověření, zdali někdo přenosový kanál neodposlouchával. Jak bylo zmíněno v předchozí podkapitole, jakékoliv měření dat tyto data ovlivňuje, z čehož vyplývá, že pokud někdo přenos odposlouchával, mohl při špatně zvolené bázi změnit polarizaci fotonu. V případě, že se odesílatel a příjemce na několika zvolených desítkách bitů shodnou, je možné s téměř absolutní pravděpodobností prohlásit, že přenos nikdo neodposlouchával. V opačném případě, pokud se liší byť jen jediný bit, došlo během přenosu k odposlechu.

8.2 Výhody a nevýhody kvantové kryptografie

Jednou z výhod kvantové kryptografie je fakt, že kvantovou informaci nelze vzhledem k její povaze jen tak změřit. Při měření totiž dochází k ovlivnění systému a informace samotné. To má za následek, že po přenosu dat stačí udělat několik kontrolních měření a ověření. Pokud se budou kontrolní měření rozcházet, došlo nepochybně během přenosu k odposlechu a je nutné daný klíč znehodnotit, zvolit jiný přenosový kanál a celý proces opakovat. V opačném případě proběhl přenos dat bez zásahu narušitele.

Další nespornou výhodou je, že kvantová mechanika nabízí mnoho možností, jak získat náhodná data. Co je nejdůležitější, v tomto případě se nejedná jen o pseudonáhodná data, která poskytují například generátory pseudonáhodných čísel, a tak se může kvantové šifrování opřít o zcela náhodné klíče vedoucí k dokonalému zabezpečení. V důsledku toho, při použití jednorázové tabulky (viz kapitola 4.3.2), která je sama o sobě jedinou neprolomitelnou šifrovací metodou, se dostáváme k možnosti opravdu dokonale bezpečného šifrování.

Výpočetní výkon kvantového počítače je díky kvantovým stavům natolik enormní, že dokáže provést mnohonásobněkrát více výpočtů než ty nejvýkonnější dnešní superpočítače dohromady. Díky tomu již ani asymetrické šifry, spoléhající na nemožnost faktorizace velkých čísel v dostupném čase, nebudou bezpečné, jelikož z veřejného klíče nebude problém získat klíč privátní. Toto je však vlastnost kvantového počítače jako takového – nejedná se tedy o kryptografickou metodu, nicméně tento fakt má na kryptografii zásadní vliv.

Posledně zmiňovanou skutečnost jsem zařadil do nevýhod, ale záleží čistě na úhlu pohledu, zdali se jedná o nevýhodu, nebo výhodu. Z pohledu kryptografů se jednoznačně jedná o nevýhodu, protože v tom případě není problém prolomit i dosud bezpečné šifry zašifrované pomocí asymetrických algoritmů s extrémně dlouhým klíčem. Naopak kryptoanalytici hledí k výpočetní síle kvantových počítačů s nadějí, jelikož by jim mohly pomoci převážit misky vah v souboji kryptografů a kryptoanalytiků na svou stranu.

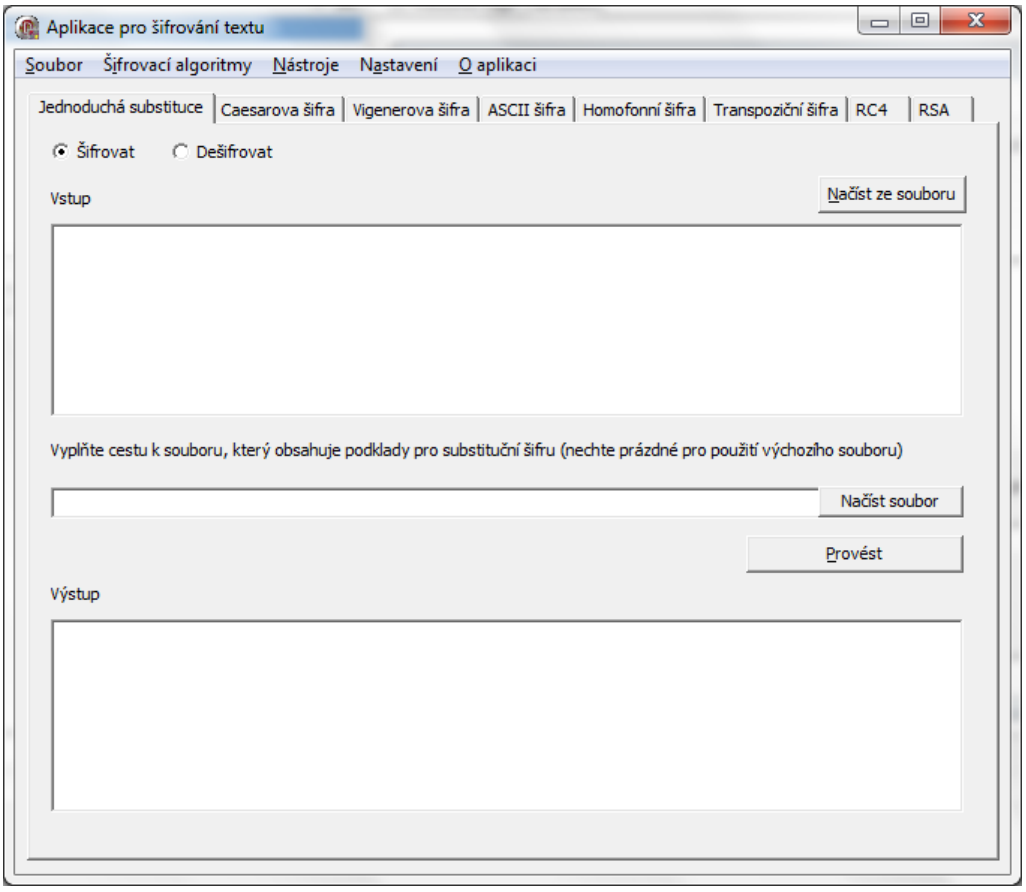
9 Praktická část – aplikace pro šifrování textu

Praktickou částí této bakalářské práce je okenní aplikace navržená ve vývojovém prostředí Delphi 2005, jejímž cílem je demonstrovat vybrané šifrovací algoritmy. Tato aplikace dokáže šifrovat a dešifrovat textový vstup.

Přítomné je jednoduché nastavení, v němž se dá zvolit, jaká abeceda se bude používat – zdali jen malá písmena, velká písmena nebo jejich kombinace. Podobně se dá nastavit, jestli se bude používat diakritika, či nikoliv.

Další volbou v nastavení je to, zdali se neznámé znaky mají při šifrování/dešifrování zachovávat, nebo jestli se mají nahradit nějakým zástupným „chybovým“ znakem. Je taktéž možné zvolit, aby se z neznámých znaků zachovávaly jen speciální znaky (mezera, čárka, tečka, vykřičník, otazník a další). U některých šifer lze toto chování vypnout.

Nastavení je uloženo v souboru **`nastaveni.dat`**, jehož strukturu lze nalézt v Příloze C. Pokud daný soubor není nalezen, je nově vytvořen při ukončení aplikace.



Obrázek 20 – Aplikace pro šifrování textu

Jedná se o grafickou okenní aplikaci, jež dokáže text šifrovat pomocí těchto algoritmů:

- jednoduchá substituční šifra,
- Caesarova šifra,
- Vigenèrova šifra,
- ASCII šifra,
- homofonní šifra,
- sloupcová transpoziční šifra,
- RC4,
- RSA.

9.1 Vzhled aplikace

V následujících dvou podkapitolách je aplikace stručně popsána po stránce designu a funkčnosti.

9.1.1 Menu

Menu aplikace obsahuje pět položek:

- *Soubor* – slouží k načtení vstupního textu, uložení výstupního textu či k ukončení aplikace,
- *Šifrovací algoritmy* – seznam všech šifrovacích algoritmů s možností přechodu na daný šifrovací algoritmus,
- *Nástroje* – názorná ukázka Caesarovy šifry a průvodce generováním klíčů pro algoritmus RSA,
- *Nastavení* – možnost měnit chování aplikace,
- *O aplikaci* – stručné informace o aplikaci.

9.1.2 Pracovní prostor

Pracovní prostor aplikace je rozdělen do šesti pomyslných částí:

- *lišta záložek* – v horní části se nachází lišta záložek, jež obsahuje jednotlivé šifrovací algoritmy,
- *dvě přepínací tlačítka* – pod lištou záložek se nachází tlačítka, která slouží pro rozlišení, zdali se bude šifrovat, či dešifrovat,
- *editační pole pro vstup a tlačítko „Načíst ze souboru“* – editační pole, do něhož se vpisuje otevřený či šifrový text (popřípadě se může načíst ze souboru),
- *nápověda ke klíči* – krátká textová nápověda, která pomáhá správně zadat šifrovací klíč,
- *editační linka pro klíč a potvrzovací tlačítko* – vstupní pole slouží k zadání šifrovacího klíče, zatímco potvrzovací tlačítko spouští proces šifrování/dešifrování,
- *editační pole pro výstup* – ve spodní části se nachází editační pole, které zobrazuje výsledek šifrovacího či dešifrovacího procesu.

9.2 Implementace algoritmů

Každý z algoritmů, jež je v této aplikaci použit, je implementován jako třída v samostatném modulu. Jednotlivé moduly jsou pak připojeny k projektu. Jedná se o tyto moduly:

- *Sifra_ASCII,*
- *Sifra_Caesar,*
- *Sifra_Homofonni,*
- *Sifra_Jednoducha_Substituce,*
- *Sifra_RC4,*
- *Sifra_RSA,*
- *Sifra_Transpozicni,*
- *Sifra_Vigenere.*

9.2.1 ASCII šifra (modul Sifra_ASCII)

ASCII šifra je jednoduchou variací Caesarovy šifry, která používá ke svému fungování místo abecedy ASCII tabulku. Výhodou je, že si třída nemusí nikde uchovávat abecedu, se kterou pracuje, a také je možné šifrovat a dešifrovat i speciální znaky. I přes to se však ASCII šifra řídí dle zvoleného typu abecedy. Toto chování lze vypnout v nastavení zaškrtnutím volby „*Nevynucovat typ abecedy u ASCII šifry*".

Otevřený text je převeden do šifrového textu, který tvoří čísla o třech číslicích z intervalu $\langle 0{,}255 \rangle$ oddělená čárkou bez mezery. Tato hodnota je číslem daného znaku v ASCII tabulce. Zvolil jsem právě tuto variantu (namísto znaků), protože prvních 31 znaků ASCII tabulky je řídících a nelze je zobrazit, tudíž při určitém posunu by mohl být otevřený text zašifrován do podoby těchto znaků, a byl by tak nečitelný.

Funkce provádějící šifrování vypadá následujícím způsobem:

```
01:  function TSifraASCII.sifrujZnak(znak: char; posun: integer): string;
02:  begin
03:    Result:=chr((ord(znak)+posun)mod 256);
04:  end;
```

Na řádku číslo **03** lze vidět, že výstupem funkce je znak ASCII tabulky posunutý o **posun** pozic od hodnoty šifrovaného znaku **znak**.

9.2.2 Caesarova šifra (modul Sifra_Caesar)

K realizaci Caesarovy šifry lze s úspěchem použít modulární aritmetiku, která velmi usnadňuje šifrování i dešifrování, což doporučuje i [MURPHY, et al., 2006]. Jednotlivé znaky abecedy jsou v tomto případě uloženy v jednorozměrném poli, jelikož aplikace může používat nejrůznější druhy abeced.

Mějme kladné celé číslo N, kde N je počet znaků zvolené abecedy, a celé číslo P, které představuje posun při použití Caesarovy šifry. Každému písmenu abecedy odpovídá jedno číslo z intervalu $\langle 0; N-1 \rangle$ a to tak, že $0 = A, 1 = B, \dots, N-1 = \check{Z}$.

Pokud chceme zašifrovat písmeno na pozici ot s posunem P, získáme pomocí modulární aritmetiky pozici zašifrovaného písmena st z těchto vztahů:

- pokud $P \geq 0$, pak: $st = (ot + P)\ mod\ N$,
- pokud $P < 0$, pak: $st = \big(ot + N + (P\ mod\ N)\big)\ mod\ N$.

Pokud chceme dešifrovat písmeno na pozici st s posunem P, získáme pozici dešifrovaného písmena ot z těchto vztahů:

- pokud $P \geq 0$, pak $ot = \Big(st + N + \big((-P)\ mod\ N\big)\Big)\ mod\ N$,
- pokud $P < 0$, pak $ot = (st - P)\ mod\ N$.

Na tomto principu je založena šifrovací funkce **sifrujZnak()** třídy **TSifraCaesar**:

```
01:  function TSifraCaesar.sifrujZnak(znak: char; posun: integer): char;
02:  var pozice: word;
03:  begin
04:    if znamZnak(znak,pozice) then
05:      if znak in spec_znaky then
06:        Result:=znak
07:      else
08:        begin
09:          if (posun>=0) then
10:            Result:=znaky[(pozice-1+posun)mod pocet]
11:          else
12:            Result:=znaky[(pozice-1+(pocet+(posun mod pocet)))mod
13:  pocet];
14:        end
15:    else
16:      Result:=ZNAK_CHYBA;;
17:  end;
```

Funkce nejprve ověřuje, zdali zná šifrovaný znak **znak** (tedy zdali daný znak patří do nastavené abecedy) a to pomocí funkce **znamZnak()** (řádek číslo **04**). Pokud šifrovaný znak patří do množiny **spec_znaky** (která obsahuje znaky **#10** a **#13** představující zalomení řádku), vrací šifrovací funkce daný znak bez aplikování šifrování (řádky **05** a **06**). Na řádcích číslo **10** a **12** dochází k samotnému šifrování znaku pomocí hodnoty **posun**.

9.2.3 Homofonní šifra (modul Sifra_Homofonni)

Při použití homofonní šifry je nezbytné nadefinovat množiny zástupných symbolů pro jednotlivé znaky. Tyto množiny se načítají ze souboru, který je možné uživatelsky vytvořit. Strukturu souboru pro homofonní šifru lze najít v Příloze D. Pro ukázku této šifry je připraven soubor **homofonni01.txt**, který obsahuje vstupní data pro abecedu malých písmen bez diakritiky.

Aplikace před samotným šifrováním ověřuje soubor, zdali existuje a následně také jestli odpovídá formát stanoveným požadavkům. Pokud vše proběhne v pořádku, dochází ke startu šifrování. V opačném případě je uživatel upozorněn pomocí dialogového okna na skutečnost, že došlo k chybě, případně i na jakém řádku souboru.

Jednotlivé znaky abecedy a množiny jejich reprezentací jsou načítány do dvourozměrného pole právě ze vstupního souboru. Při šifrování konkrétního znaku se náhodně vybere jedna z možných reprezentací.

Výjimečná situace nastává v případě, že se v otevřeném textu objeví některý neznámý znak. V tom případě dojde – v závislosti na volbě nastaveného filtrování – k jedné z těchto tří situací:

- volba „**nefiltrovat**“, pak bude znak převeden do podoby skupiny *k* stejných znaků,
- volba „**filtrovat**“, pak bude znak nahrazen skupinou *k* znaků „**#**“,
- volba „**nefiltrovatInterpunkci**“, pak bude znak převeden do:
  - skupiny *k* stejných znaků, pokud se jedná o interpunkční znak,
  - skupiny *k* znaků „**#**“, pokud se jedná o libovolný jiný znak.

Hodnota *k* je v těchto případech rovna počtu symbolů použitých pro jednu zástupnou reprezentaci.

Šifrovací funkce **sifrujZnak()** třídy **TSifraHomofonni** vypadá následujícím způsobem:

```
01:  function TSifraHomofonni.sifrujZnak(znak: char): string;
02:  var pozice,j,maxZnaku: byte;
03:  begin
04:    Randomize;
05:    if najdiZnak(znak,pozice) then
06:      begin
07:        maxZnaku:=strtoint(znaky[pozice,1]);
08:        j:=Random(maxZnaku)+2;
09:        Result:=znaky[pozice,j]+' ';
10:        exit;
11:      end
12:    else
13:      begin
14:        Result:=pocetZnaku(znakuKod, ZNAK_CHYBA)+',';
15:        exit;
16:      end;
17:  end;
```

Šifrovací funkce nejprve ověřuje, zdali daný znak **znak** patří do nastavené abecedy a to pomocí funkce **najdiZnak()**, jak lze vidět na řádku číslo **05**. Na řádku číslo **08** je náhodně zvolena jedna ze zástupných reprezentací daného znaku. Pokud šifrovaný znak nepatří do nastavené abecedy, vrací šifrovací funkce **znakuKod**-krát chybový znak, což generuje funkce **pocetZnaku()**, kde **znakuKod** je počet symbolů tvořících jednu zástupnou reprezentaci (řádek číslo **14**).

9.2.4 Jednoduchá substituční šifra (modul Sifra_Jednoducha_Substituce)

Jednoduchá substituční šifra vyžaduje ke svému fungování substituční tabulku, která určuje, kterým znakem bude dané písmeno (či znak) nahrazeno. Substituční tabulka se v tomto případě opět načítá ze souboru.

Uživatel aplikace si může vytvářet své vlastní soubory se substituční tabulkou. Jedinou nutností je dodržet strukturu souboru, která je popsána v Příloze E. Pro ukázku této šifry je připraven soubor **substitucni01.txt**, který obsahuje podklad pro abecedu malých písmen bez diakritiky.

Třída ukládá písmena abecedy a jejich nahrazující znaky do dvourozměrného pole o dvou řádcích.

Šifrovací funkce **sifrujZnak()** třídy **TSifraJednoduchaSubstituce** vypadá následovně:

```
01:  function TSifraJednoduchaSubstituce.sifrujZnak(znak: char): char;
02:  var pozice: word;
03:  begin
04:    if znamZnak(znak,pozice) then
05:      begin
06:        if znak in spec_znaky then
07:          Result:=znak
08:        else
09:          Result:=znaky[pozice-1,1];
10:      end
11:    else
12:      Result:=ZNAK_CHYBA;
13:  end;
```

Šifrovací funkce si nejprve ověří pomocí funkce **znamZnak()**, zdali skutečně šifrovaný znak **znak** patří do nastavené abecedy (řádek číslo **04**). Pokud ano, vrací funkce jeho nahrazující znak z pole **znaky** (řádek číslo **09**). V opačném případě vrací chybový znak **ZNAK_CHYBA** (řádek číslo **12**).

9.2.5 RC4 (modul Sifra_RC4)

Implementace algoritmu RC4, coby zástupce proudových šifrovacích algoritmů, zahrnovala dvě nezbytné fáze:

- inicializace klíčovacího toku,
- samotné šifrování/dešifrování.

K udržování stavu používá třída implementující tento algoritmus dvě stavové proměnné. Jelikož vývojové prostředí Borland Delphi 2005 vyžaduje pro návrh for cyklů použití lokálních proměnných (a tedy ne proměnných třídy), byl jsem při návrhu inicializační části nucen nahradit *for cykly* poněkud těžkopádnými *cykly while-do*, jak ukazuje následující zdrojový kód.

```
01:  constructor TSifraRC4.Create(newKlic: string);
02:  begin
03:    klic:=newKlic;
04:
05:    i:=0;
06:    j:=0;
07:
08:    while (true) do
09:      begin
10:        S[i]:=i;
11:        if (i=255) then break;
12:        inc(i);
13:      end;
14:
15:    i:=0;
16:    while (true) do
17:      begin
18:        j:=(j+S[i]+ord(klic[(i mod length(klic))+1])) mod 256;
19:        temp:=S[i];
20:        S[i]:=S[j];
21:        S[j]:=temp;
22:        if (i=255) then break;
23:        inc(i);
24:      end;
25:
26:    i:=0;
27:    j:=0;
28:  end;
```

Na řádcích číslo **08** až **13** dojde k inicializaci prvků pole **S** hodnotami proměnné **i**. Poté na řádcích číslo **16** až **24** dochází k vytvoření klíčovacího toku pomocí šifrovacího klíče **klic**.

```
01:  function TSifraRC4.sifrujZnak(znak: char): byte;
02:  var vyslXOR: byte;
03:  begin
04:    i:=(i+1) mod 256;
05:    j:=(j+S[i]) mod 256;
06:    temp:=S[i];
07:    S[i]:=S[j];
08:    S[j]:=temp;
09:    vyslXOR:=S[(S[i]+S[j]) mod 256]xor(ord(znak));
10:
11:    Result:=vyslXOR;
12:  end;
```

Funkce **sifrujZnak()** nejprve zaktualizuje hodnotu klíčovacího toku (řádky číslo **04** až **08**). Na řádku číslo **09** je do proměnné **vyslXOR** přiřazena hodnota klíče vzniklá z klíčovacího toku, k níž je pomocí operace XOR připojen šifrovaný znak **znak**.

Abych zabránil přetečení datového typu *byte*, který může obsahovat jen čísla z intervalu $\langle 0, 255 \rangle$, používám jak v incializační části, tak při šifrování a dešifrování operaci modulo.

Navíc je šifrový text tvořen právě čísly z intervalu $\langle 0, 255 \rangle$, jelikož pokud by byly v této fázi převedeny pomocí ASCII tabulky, hrozil by stejný problém jako u ASCII šifry. Tedy, že znaky zašifrované do podoby čísla z intervalu $\langle 0, 31 \rangle$ by nebyly korektně zobrazeny.

Aplikace si také hlídá, zdali jsou zadaná čísla právě ze zmiňovaného intervalu $\langle 0, 255 \rangle$. Pokud tomu tak není, tak aplikace vyvolá dialogové okno upozorňující na chybný formát vstupních dat.

Ačkoliv i šifra RC4 dokáže šifrovat text složený z libovolných znaků, řídí se podle nastavení filtrování a zvolené abecedy. Toto chování je opět možné vypnout v nastavení a to zaškrtnutím volby *„Nevynucovat typ abecedy u šifry RC4“*.

9.2.6 RSA (modul Sifra_RSA)

Při implementaci algoritmu RSA jsem narazil na tři problémy, které jsem však dokázal nakonec eliminovat:

1) pomalost při generování klíčů,
2) problém příliš velkého exponentu při šifrování a dešifrování (exponent v řádu stovek milionů až několika miliard),
3) přetékání i těch největších celočíselných typů.

Pomalost při generování klíčů spočívala v tom, že je nutné najít hodnotu soukromého klíče d a to tak, aby platilo $e \cdot d \equiv 1 \, mod \, n$, kde e je jedna část veřejného klíče a n je část veřejného i soukromého klíče.

Pro d dále platí, že se jedná o celé číslo z intervalu $(1, phi)$, kde $phi = (P - 1) \cdot (Q - 1)$. P a Q jsou prvočísla. Proto bylo mým prvním řešením použití *cyklu for*:

```
01:  for i:=2 to phi-1 do
02:    begin
03:      if (((e*i)mod n)=1) then
04:        begin
05:          d:=i;
06:          break;
07:        end;
08:    end;
```

V rámci for cyklu se testují hodnoty $2, 3, \ldots, (phi - 1)$, zdali splňují rovnici $e \cdot i = 1 \, mod \, n$ (řádek číslo `03`), kde e je jedna část veřejného klíče a n je druhá část veřejného klíče.

Tento způsob je však silně neefektivní, což se projeví už u hodnoty $n = 4 \cdot 10^9$, což je přibližně největší hodnota, kterou lze uložit do celočíselného typu *longword*. Proto jsem algoritmus upravil za pomoci takzvané **rozšířené verze Euklidova algoritmu**, která

se dá vhodně použít právě ke spočtení multiplikativní inverze [The Euclidean Algorithm and the Extended Euclidean Algorithm, 2010], [MENEZES, et al., 1996].

Tabulka 23 – Porovnání průměrné doby výpočtu při použití *for cyklu* a *rozšířeného Euklidova algoritmu* (průměr z 10 měření)

Hodnota n	For cyklus	Rozšířený Euklidův algoritmus
n = 4149678499	28 sekund	< 1 sekunda

Druhým problémem bylo, že se při šifrování postupuje podle vztahu

$$st = (ot^e)\ mod\ n$$

kde st je výsledná podoba šifrového textu, ot je otevřený text reprezentovaný číslem, e je jedna část veřejného klíče a n je druhá část veřejného klíče. Jelikož e může taktéž jako d nabývat hodnot v rozmezí $(1, phi)$, může v mé aplikaci dosáhnout hodnoty až $e = 3034907956$. I když této hodnoty ve většině případů nedosáhne, téměř vždy se pohybuje hodnota e nad hranicí několika tisíc, což je stále příliš vysoké číslo.

Z toho důvodu jsem se dostal k takzvané **Montgomeryho redukci**, která je extrémně efektivní, jelikož její složitost je v nejhorším případě $\log_2 n$, kde n je hodnota exponentu [MENEZES, et al., 1996]. Její princip spočívá v tom, že se exponent převede do binární podoby a základ se umocňuje na druhou, přičemž se zároveň provádí operace modulo a počítají se mezivýsledky:

```
01:  function TSifraRSA.exponentModulo(zaklad, exponent, modulo:
02:  int64): int64;
03:  var exp: string;
04:      vysl: longword;
05:      i: byte;
06:  begin
07:    vysledek:=1;
08:
09:    cisloToBitsBackwards(exp, exponent);
10:
11:    for i:=1 to length(exp) do
12:      begin
13:        if exp[i]='1' then
14:          vysledek:=(vysledek*zaklad) mod modulo;
15:
16:        zaklad:=(zaklad*zaklad) mod modulo;
17:      end;
18:
19:    Result:=vysledek;
20:  end;
```

Výše uvedená část zdrojového kódu slouží ke spočtení hodnoty

$$vysledek = zaklad^{exponent}\ mod\ modulo.$$

Nejprve je na řádku číslo **09** zavolána procedura **CisloToBitsBackwards(exp, exponent)**, která převede číselnou hodnotu **exponent** na řetězec znaků **exp**, který představuje hodnotu exponentu ve dvojkovém zápisu. Tato binární hodnota je ale **zapsána pozpátku** (jelikož by po přepsání do správné podoby byla stejně procházena odzadu). Poté se pro každý znak tohoto binárního čísla prochází cyklem, kdy se nejprve testuje, zdali je daný znak exponentu „1“, či „0“ (řádek číslo **13**). V případě, že je znakem „1“, aktualizuje se hodnota výsledku **vysledek**. V obou případech se aktualizuje také hodnota základu **zaklad** (řádek číslo **16**).

Tabulka 24 – počet potřebných pomocných výpočtů při použití Montgomeryho redukce

Hodnota exponentu	Pomocných výpočtů
e = 10	4
e = 1000	10
e = 100000	17
e = 10000000	24
e = 1000000000	30

K šifrování pak slouží funkce **sifrujZnak()** třídy **TSifraRSA**:

```
01:  function TSifraRSA.sifrujZnak(znak: char; var ok: boolean):
02:  longword;
03:  begin
04:    Result:=0;
05:
06:    if e>1 then
07:      begin
08:        Result:=exponentModulo(ord(znak), e, n);
09:        ok:=true;
10:      end
11:    else
12:      ok:=false;
13:  end;
```

Šifrovací funkce nejprve testuje, zdali je zvolená hodnota **e** větší než **1** (řádek číslo **06**). Pokud ano, využije se funkce **exponentModulo()**, vysvětlené na předchozí stránce, k získání zašifrovaného znaku a nastaví se příznak **ok** na hodnotu **true**, čímž funkce dává najevo, že vše proběhlo v pořádku (řádky číslo **08 a 09**). V opačném případě se příznak **ok** nastaví na hodnotu **false** (řádek číslo **12**).

RSA šifra není nikterak omezena, co se typu šifrovaných znaků týče. Přesto se jako všechny ostatní šifry chová v závislosti na nastavené abecedě. Toto chování lze vypnout v nastavení zaškrtnutím volby *„Nevynucovat typ abecedy u šifry RSA“*.

Posledním problémem bylo nedostatečné rozpětí i toho největšího celočíselného typu, kterým je v Delphi 2005 **Int64** a do něhož se dá uložit celé číslo z intervalu $\langle -2^{63}, 2^{63} - 1 \rangle$. I tak ale docházelo k přetečení a to právě při počítání Montgomeryho

redukce. Proto jsem byl nucen snížit hodnotu maximálního možného vstupu na 55103. K této hodnotě jsem přišel z následujících rovnic:

- Uvažujme prvočísla p a q, pro něž platí $p \sim q$, tudíž hodnota části veřejného klíče $n = (p \cdot q) \cong p^2$.
- Při Montgomeryho redukci se počítá vztah $zaklad = zaklad^2 \, mod \, n$. Pokud dosadíme z předchozí rovnice za n, dostáváme vztah:

$$zaklad = zaklad^2 \, mod \, p^2.$$

- Jelikož při operaci zbytku po dělení nabývá $zaklad$ maximálně hodnoty p^2, která je při výpočtu dále umocněna na druhou, musí být toto číslo menší než je maximální hodnota datového typu int64:

$$(p^2)^2 < \max(int64)$$

- Pokud nyní dosadíme za $\max(int64)$ hodnotu $2^{63} - 1$, dostáváme nerovnici:

$$p^4 < 2^{63} - 1$$

$$p < \sqrt[4]{2^{63} - 1}$$

$$p \leq 55108$$

Největší prvočíslo menší než hodnota p je právě výše uvedených 55103.

Speciálně pro šifrování pomocí algoritmu RSA byl vytvořen pomocník pro tvorbu šifrovacích klíčů. Ten pracuje ve dvou módech:

1) ruční,
2) automatický.

V prvně zmiňovaném může uživatel zadat dvě prvočísla, na jejichž základě se vygenerují oba klíčové páry. Pochopitelně jsou ošetřeny nejrůznější situace, kdy některé z čísel není prvočíslo, nepatří do požadovaného intervalu a tak dále.

Automatický mód generuje klíčové páry zcela automaticky na základě volby přepínače určujícího velikost prvočísel:

- malá,
- střední,
- velká.

9.2.7 Transpoziční sloupcová šifra (modul Sifra_Transpozicni)

Transpozičních šifer existuje hned několik, jak bylo ukázáno v kapitole 4.2. Tato aplikace používá sloupcovou transpozici bez doplnění neúplných sloupců.

Šifrovací funkce **`sifruj()`** třídy **`TSifraTranspozicni`** vypadá následovně:

```
01:  function TSifraTranspozicni.sifruj(otevrenyText: string; var
02:  sifrovyText: string): boolean;
03:  var i: integer;
04:      j: integer;
05:      help: string;
06:  begin
07:    help:='';
08:    for i:=1 to length(otevrenyText) do
09:      if otevrenyText[i]<>#10 then
10:        help:=help+otevrenyText[i];
11:
12:    sifrovyText:='';
13:    if (klic>1)and(klic<length(help)) then
14:      begin
15:        for i:=1 to klic do
16:          begin
17:            j:=i;
18:
19:            repeat
20:              sifrovyText:=sifrovyText+help[j];
21:              j:=j+klic;
22:            until (j>length(help));
23:          end;
24:
25:        Result:=true;
26:      end
27:    else
28:      Result:=false;
29:  end;
```

Šifrovací funkce nejprve vytvoří z otevřeného textu **`otevrenyText`** pomocný text **`help`**, přičemž z otevřeného textu odstraní všechny znaky **`#10`** způsobující zalomení stávajícího řádku (řádky číslo **`08`** až **`10`**). Poté v případě, že je klíč **`klic`** delší než jeden znak a zároveň kratší než pomocný otevřený text, přichází na řadu samotné šifrování (řádky číslo **`14`** až **`26`**).

Sloupcová transpoziční šifra nepotřebuje pro svou práci žádnou datovou strukturu, ve které by uchovávala abecedu. I přes to, že tedy není tato šifra závislá na nastaveném typu abecedy, chová se tak. Tuto možnost lze opět vypnout v nastavení zaškrtnutím volby „*Nevynucovat typ abecedy u transpoziční šifry*“.

9.2.8 Vigenèrova šifra (modul Sifra_Vigenere)

Při implementaci Vigenèrovy šifry jsem vhodně využil již představené Caesarovy šifry. Třída **`TSifraVigenere`** totiž obsahuje jako atribut instanci třídy **`TSifraCaesar`**.

Pomocí této instance a zadaného posunu lze snadno a rychle vytvořit libovolný řádek Vigenèrova čtverce.

Šifrovací funkce **sifrujZnak()** třídy **TSifraVigenere** vypadá takto:

```
01:  function TSifraVigenere.sifrujZnak(znak: char; kl: char): char;
02:  var poziceKlice: word;
03:  begin
04:    if pomoc.znamZnak(kl, poziceKlice) then
05:      begin
06:        Result:=pomoc.sifrujZnak(znak,poziceKlice-1);
07:      end
08:    else
09:      Result:=znak;
10:  end;
```

Privátní proměnná **pomoc** je instancí Caesarovy šifry (třída **TSifraCaesar**), která slouží k vytvoření libovolného řádku Vigenèrova čtverce. Šifrovací funkce nejprve ověří pomocí proměnné **pomoc** a funkce **znamZnak()**, zdali znak klíče **kl**, podle něhož se bude šifrovat, patří do nastavené abecedy (řádek číslo **04**). Pokud ano, tak dochází k samotnému šifrování pomocí proměnné **pomoc** a funkce **sifrujZnak()** (řádek číslo **06**).

10 Závěr

Kryptografie je bezesporu zajímavou vědeckou disciplínou, se kterou v dnešní době lidé běžně přicházejí do styku každý den, aniž by si toho byli vůbec vědomi – nemám nyní na mysli jen komunikaci po internetu, ale i spoustu offline aktivit a dennodenních situací.

Cílem této práce bylo přinést ucelený pohled na kryptografii, ať už se jedná o její historii, dobové postupy, cíle šifrování, ale také o samotné „*vnitřnosti*“, které odvádí během šifrování většinu práce – samy šifrovací algoritmy.

Šifrovací algoritmy byly probrány od těch nejtriviálnějších – ač se v dnešní době a kontextu mohou zdát poněkud úsměvné a beze smyslu – až po ty v současnosti používané nejmodernější algoritmy jako například AES, RC4 či RSA. Jejich tvorba a návrh jsou extrémně komplexní, a proto musely být některé kapitoly zjednodušeny. I tak je ale tato práce popisuje tak, aby nebyly žádné části vynechány a aby měl čtenář ucelenou představu o jejich fungování.

Taktéž byla nastíněna možná budoucnost kryptografie v podobě kvantové kryptografie, jež ale patří stále spíše do teoretické roviny a ve výsledku může budoucnost vypadat úplně jinak.

Nedílnou součástí této práce je také praktická část v podobě okenní aplikace sloužící k šifrová/dešifrování textu, k čemuž používá osm vybraných algoritmů, v nichž jsou zastoupeny i tři moderní algoritmy. Jedním z cílů praktické části byla implementace algoritmu RSA a tento cíl byl splněn, ačkoliv právě při programování tohoto algoritmu vzniklo nejvíc otazníků. Na druhou stranu si však těchto situací velmi cením, jelikož jsem díky nim nakoukl pod pokličku některých velmi zajímavých matematických postupů (Montgomeryho redukce, rozšířený Euklidův algoritmus). Na vlastní kůži jsem si navíc vyzkoušel, jak zásadní význam má na rychlost aplikace matematická složitost řešeného problému.

Jelikož se tématu kryptografie a šifrovacích algoritmů hodlám věnovat i nadále, beru tuto práci jako takový úvod do problematiky. Díky ní už nyní například vím, že při programování moderních šifrovacích algoritmů bude nezbytné nespoléhat na vestavěné datové typy, nýbrž vytvořit si svoje vlastní včetně operací nad nimi a tak dále.

K této práci je přiložený CD-ROM obsahující jak tuto textovou část v elektronické podobě, tak zdrojové kódy vzorové aplikace včetně její spustitelné verze.

Bibliografie

ANDERSON, Ross – BIHAM, Eli – KNUDSEN, Lars. *Serpent home page* [online]. [1998?] [citováno 13. března 2011].
Dostupné z WWW: <http://www.cl.cam.ac.uk/~rja14/Papers/serpent.tar.gz>.

BARKER, Elaine, et al. *Recommendation for Key Management - Part 1: General (Revised)* [online] 8th March of 2007 [citováno 11. lenda 2011].
Dosutpné z WWW: <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf>. Dokument lze nalézt pod identifikátorem NIST SP 800-57.

BENNETT, Charles H – BRASSARD, Gilles. IEEE International Conference on Computers, Systems, and Signal Processing. *Quantum Cryptography: Public Key Distribution and Coin Tossing* [online]. Bangalore, 1984 [citováno 14. ledna 2011].
Dostupné z WWW:
<http://www.research.ibm.com/people/b/bennetc/bennettc198469790513.pdf>.

Block Cipher. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 24. září 2006, last revision 8th of January 2011 [citováno 15. března 2011]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Template:Crypto_block>.

CHRISTENSEN, Chris. *Transposition Ciphers* [online]. 2006 [citováno 23. února 2011]. Dostupné z WWW: <www.nku.edu/~christensen/section%209%20transposition.pdf>.

COBB, Chey. *Cryptography for Dummies.* 1st edition. Indianapolis (Indiana): Wiley, 2004. 336 s. ISBN 0764541889.

DENIS, Tom St – JOHNSON, Simon. *Cryptography for Developers.* 1st edition. Rockland (Massachusetts): Syngress Publishing, 2007. 423 s. ISBN 1-59749-104-7.

DIFFIE, William – HELLMAN, Martin E. IEEE Transactions on Information Theory. *New Directions in Cryptography* [online]. 1976 [citováno 14. ledna 2011]. Dostupné z WWW:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.494&rep=rep1&type=pdf>.

ELGAMAL, Taher. IEEE Transactions on Information Theory. *A public-key cryptosystem and a signature scheme based on discrete logarithms* [online]. 1985 [citováno 15. ledna 2011]. Dostupné z WWW:
<http://hereford.homeip.net/ElGamal/ElGamal%20-%20A%20Public%20Key%20Cryptosystem%20and%20a%20Signature%20Scheme%20Based%20on%20Discrete%20Logarithms%20ElGamal.pdf>.

GAINES, Helen Fouché. *Cryptanalysis: A Study of Ciphers and Their Solution.* 1st edition. New York: Dover Publications, 1956. 256 s. ISBN 0486200973.

KALA, Vít. *Kongruence a teorie čísel* [online]. 2004 [citováno 24. února 2011]. Dostupné z WWW:
<http://mks.mff.cuni.cz/library/KongruenceATeorieCiselVK/KongruenceATeorieCiselVK.pdf>.

KÁLDY, Robert. *Velká prvočísla* [online]. 2000 [citováno 25. února 2011]. Dostupné z WWW: <http://mks.mff.cuni.cz/library/VelkaPrvocislaRK/VelkaPrvocislaRK.pdf>.

KOBLITZ, Neal. Mathematics of Computation. *Elliptic Curve Cryptosystems* [online]. 1987 [citováno 27. února 2011]. Dostupné z WWW: <http://www.ams.org/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf>.

KRÁLÍK, Jan. *The Czech Language On WWW* [online]. 2001 [citováno 17. února 2011]. Dostupné z WWW: <http://www.czech-language.cz/alphabet/alph-prehled.html>.

KUPČA, Vojtěch. *Teorie a perspektiva kvantových počítačů.* Praha, 2001. Diplomová práce na Fakultě elektrotechnické Českého vysokého učení technického v Praze. Vedoucí diplomové práce Ing. Tomáš Rosa.

MENEZES, Alfred – OORSCHOT, Paul – VANSTONE, Scott. *Handbook of Applied Cryptography.* 1st edition. Ottawa: CRC Press, 1996. 780 s.
ISBN 9780849385230. Dostupné z WWW: <http://www.cacr.math.uwaterloo.ca/hac/>.

MILLER, Victor Saul. Advances in Cryptology – CRYPTO '85. *Use of elliptic curves in cryptography*. 1986 [citováno 27. února 2011].

MILLER, Victor Saul. *Elliptic Curves and their use in Cryptography* [online]. 21st of March 2007 [citováno 27. února 2011].
Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.1785&rep=rep1&type=pdf>.

MURPHY, Sean – PIPER, Fred. *Kryptografie: Průvodce pro každého.* 1. vydání.
Praha: Dokořán, 2006. 158 s. ISBN 80-7363-074-5.

NECHVATAL, J. Computer Security Resource Center of National Institute of Standards and Technology. *Report on the Development of the Advanced Encryption Standard (AES)* [online]. 2000 [citováno 13. března 2011]. Dostupné z WWW:
<http://csrc.nist.gov/archive/aes/round2/r2report.pdf>.

ONDRÁČKOVÁ, Eva. *Lineární algebra* [online]. 2002 [citováno 24. února 2011]. Dostupné z WWW:
<http://mks.mff.cuni.cz/library/LinearniAlgebraEO/LinearniAlgebraEO.pdf>.

PAAR, Christof – PELZL, Jan. *Understanding Cryptography.* 1st edition.
Heidelberg: Springer, 2010. 372 s. ISBN 978-3-642-04100-6.

RITTER, Terry. *Dynamic Transposition Revisited Again (long)* [online]. 6th of March 2001 [citováno 23. února 2011]. Dostupné z WWW: <http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM>.

RIVEST, Ronald L. *Ronald L. Rivest: FAQ* [online]. [199-?] [citováno 9. března 2011]. Dostupné z WWW: <http://people.csail.mit.edu/rivest/faq.html>.

ROOS, Andrew. <andrewr@vironix.co.za**>** *Weak Keys in RC4* [Online] 22nd of September 1995. <sci.crypt.research> [citováno 9. března 2011]. Dostupné z WWW: <http://groups.google.com/group/sci.crypt.research/msg/078aa9249d76eacc>.

RSA Algorithm. In *Cryptography Code* [online]. Sydney: DI Management, c2002-2011, last revision 22nd of February 2011 [citováno 24. března 2011]. Dostupné z WWW: <http://www.di-mgt.com.au/rsa_alg.html>.

RSA Laboratories. *TWIRL and RSA Key Size* [online]. 6th of March 2003, last revision 6th of May 2003 [citováno 31. prosince 2010.] Dostupné z WWW: <http://www.rsa.com/rsalabs/node.asp?id=2004#table1>.

SCHNEIER, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* 2nd edition. New York: Wiley, 1995. 784 s. ISBN 0471128457.

Contemporary Cryptology: The Science of Information Integrity. Edited by **SIMMONS, Gustavus. J.** 1st edition. New York: Institute of Electrical & Electronics Enginee, 1991. 640 s. ISBN 0879422777.

SINGH, Simon. *Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii.* 2. vydání. Praha: Dokořán, 2009. 382 s. ISBN 978-80-7363-268-7.

Stream Cipher. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 24th of October 2006, last revision 29th of January 2011 [citováno 15. března 2011]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Template:Crypto_stream>.

The Euclidean Algorithm and the Extended Euclidean Algorithm. In *Cryptography Code* [online]. Sydney: DI Management, 14th of August 2010, last revision 22nd of October 2010 [citováno 24. března 2011]. Dostupné z WWW: <http://www.di-mgt.com.au/euclidean.html>.

VAUDENAY, Serge. *A Classical Introduction To Cryptography: Applications for Communications Security.* 1st edition. New York: Springer, 2006. 335 s. ISBN 0-387-25464-1.

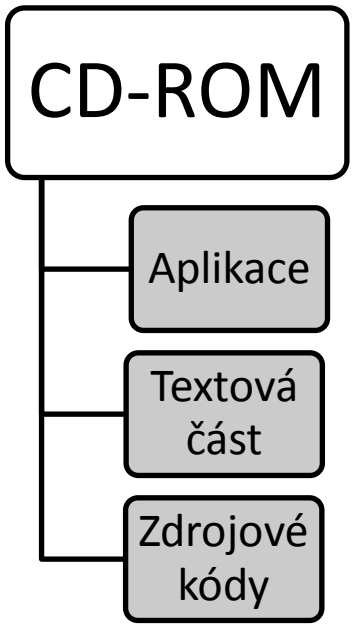
WIDHIASTRA, I Made Krisna. *Rail-Fence Cipher* [online]. 20th of December 2010 [citováno 24. února 2011.] Dostupné z WWW: <http://times.imkrisna.com/2010/12/rail-fence-cipher/>.

WOBST, Reinhard. *Cryptology Unlocked.* 1st edition. Chichester: Wiley, 2007. 540 s. ISBN 0470060646.

Příloha A – Adresářová struktura přiloženého CD-ROM

Přiložený CD-ROM obsahuje zdrojové kódy praktické části, zkompilovaný program s potřebnými soubory a samozřejmě také tuto práci ve formátu PDF.

Adresářová struktura je následující:



Příloha B – S-boxy použité v algoritmu DES [PAAR, et al., 2010]

S-Box číslo 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box číslo 2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box číslo 3:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box číslo 4:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box číslo 5:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box číslo 6:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box číslo 7:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box číslo 8:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Příloha C – Struktura souboru „nastaveni.dat“

Soubor **`nastaveni.dat`** uchovává nastavené chování celé aplikace. Tato nastavení se načítají při startu aplikace. Pokud při spuštění aplikace soubor chybí nebo není v požadovaném formátu, aplikace o tom informuje uživatele, použije výchozí nastavení a při ukončení soubor s nastavením vytvoří.

Tento soubor má svoji strukturu, kterou je nutné dodržet:

1. typ abecedy,
2. typ filtrování,
3. hodnotu určující, zdali nevynucovat typ abecedy u ASCII šifry,
4. hodnotu určující, zdali nevynucovat typ abecedy u šifry RC4,
5. hodnotu určující, zdali nevynucovat typ abecedy u šifry RSA,
6. hodnotu určující, zdali nevynucovat typ abecedy u transpoziční šifry.

Typ abecedy

Typ abecedy udává, s jakým typem abecedy bude program pracovat:

- **`male`**,
- **`velke`**,
- **`maleVelke`**,
- **`maleEN`**,
- **`velkeEN`**,
- **`maleVelkeEN`**.

Typ filtrování

Typ filtrování udává, zdali bude aplikace zachovávat při šifrování neznámé znaky, zdali je bude nahrazovat chybovým znakem „#“ či zdali bude zachovávat alespoň interpunkční znaky. Tyto tři typy filtrování jsou reprezentovány čísly následovně:

- 1 – nefiltrovat znaky,
- 2 – filtrovat znaky,
- 3 – nefiltrovat interpunkční znaky.

Hodnoty určující, zdali se u některých šifer nemá vynucovat typ abecedy

Tyto hodnoty určují, zdali program u vybraných šifer má, či nemá vynucovat nastavenou abecedu. Jedná se o šifry, které umí šifrovat znaky nezávisle na zvoleném typu abecedy:

- ASCII šifra,
- RC4,
- RSA,
- transpoziční sloupcová šifra.

Hodnoty jsou reprezentovány buď číslem „0“, nebo „1“:

- „0“ – vynucovat typ zvolené abecedy,
- „1“ – nevynucovat typ zvolené abecedy.

Ukázka souboru „nastaveni.dat“

Obsah souboru	Poznámka k významu daného řádku
velke	Aplikace používá velká písmena.
3	Nebudou se filtrovat interpunkční znaky.
1	Nevynucovat typ abecedy u ASCII šifry.
0	Vynucovat typ abecedy u šifry RC4.
0	Vynucovat typ abecedy u šifry RSA.
0	Vynucovat typ abecedy u transpoziční šifry.

Příloha D – Struktura souboru pro homofonní šifru

Soubor obsahující zástupné reprezentace při použití homofonní šifry má pevně danou strukturu, která se skládá z těchto částí:

1. typ abecedy,
2. počet znaků jedné zástupné reprezentace,
3. výčet jednotlivých množin reprezentujících písmena,
4. výčet množin reprezentujících speciální znaky.

Jedná se o obyčejný textový soubor s příponou ***.txt**.

Typ abecedy

Typ abecedy udává, zdali se bude jednat o českou, či anglickou abecedu, a zdali budou použita jen malá písmena, velká písmena, či jejich kombinace. Typ abecedy nabývá jedné z těchto možností:

- **male**,
- **velke**,
- **maleVelke**,
- **maleEN**,
- **velkeEN**,
- **maleVelkeEN**.

První tři možnosti reprezentují českou abecedu, zbylé pak abecedu anglickou.

Počet znaků jedné zástupné reprezentace

Jelikož reprezentace určitého písmena či znaku může být složená z několika symbolů, udává tato číselná hodnota, z kolika symbolů se skládá právě jedna reprezentace.

Výčet jednotlivých množin reprezentujících písmena

Následuje výčet množin reprezentací jednotlivých písmen abecedy. Je vhodné upozornit, že písmena jsou uspořádána dle abecedy od „a“ do „z“ případně „ž“. Pokud se jedná o abecedu obsahující malá i velká písmena, následuje velké písmeno bezprostředně za malým.

Jednotlivé reprezentace se píší na řádek a oddělují se čárkou bez mezery. Každé písmeno abecedy má vyhrazený jeden řádek.

Výčet množin reprezentujících speciální znaky

Nyní následují tři řádky obsahující reprezentace znaků „mezera“, „tečka“ a „čárka“ v tomto pořadí.

Ukázka vzorového souboru „homofonni01.txt“

Obsah souboru	Poznámka k významu daného řádku
maleEN	Jedná se o malá písmena anglické abecedy.
2	Reprezentace se skládá ze dvou symbolů.
12,33,53,67,78,92	Množina zastupující písmeno „a“.
48,81	Množina zastupující písmeno „b“.
13,41,62	
01,03,45,79	
24,44,46,55,57,64,82,87	
10,31	
06,25	
23,56,65,68	
32,70,73,83,88,93	
15	
04	
26,37,51,84	…
22,27	
18,58,59,66,71,91	
00,05,07,54,72,90,99	
38,95	
94	
29,35,40,42	
11,19,36,76,86	
17,20,30,49,59,75,85	
08,61,63	
34	
60,89	
28	
21,52	
02	Množina zastupující písmeno „z“.
98,80,47,16,50,14,39,74,43,09	Množina zastupující znak „mezera“.
97,77	Množina zastupující znak „tečka“.
96	Množina zastupující znak „čárka“.

Příloha E – Struktura souboru pro jednoduchou substituční šifru

Soubor obsahující substituční tabulku pro použití při šifrování a dešifrování pomocí jednoduché substituce je obyčejný textový soubor s příponou **`*.txt`**. Jeho struktura je pevně daná a skládá se z těchto částí:

1. typ abecedy,
2. seznam nahrazujících znaků.

Typ abecedy

Typ abecedy udává, zdali se bude jednat o českou, či anglickou abecedu, a zdali budou použita jen malá písmena, velká písmena, či jejich kombinace. Typ abecedy nabývá jedné z těchto možností:

- **`male`**,
- **`velke`**,
- **`maleVelke`**,
- **`maleEN`**,
- **`velkeEN`**,
- **`maleVelkeEN`**.

První tři možnosti reprezentují českou abecedu, zbylé pak abecedu anglickou.

Seznam nahrazujících znaků

Následuje seznam nahrazujících znaků. Každý z nich se píše na samostatný řádek. Znak na prvním řádku zastupuje písmeno „a“, na druhém písmeno „b“ a tak dále.

Ukázka vzorového souboru „substitucni01.txt“

Obsah souboru	Poznámka k významu daného řádku
maleEN	Jedná se o malá písmena anglické abecedy.
o	Znak „a“ je nahrazen znakem „o“.
c	Znak „b“ je nahrazen znakem „c“.
v	
x	
a	
k	
m	
y	
s	
t	
z	
e	…
h	
f	
w	
i	
p	
b	
q	
l	
n	
d	
g	
j	
r	
u	Znak „z“ je nahrazen znakem „u“.