Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Užití teorie grup v informatice

2016

Kamil Hanus

Vedoucí práce: RNDr. Miroslav Kolařík, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Kamil Hanus
Název práce:	Užití teorie grup v informatice
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2016
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	48
Přílohy:	1 CD
Jazyk práce:	český

Bibliograhic info

Author:	Kamil Hanus
Title:	Usage of group theory in computer science
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2016
Study field:	Computer Science, full-time form
Supervisor:	RNDr. Miroslav Kolařík, Ph.D.
Page count:	48
Supplements:	1 CD
Thesis language:	Czech

Anotace

Uplatnění různých algebraických struktur, včetně teorie grup, lze nalézt v mnohých oblastech informatiky, i když to nemusí být na první pohled patrné. Práce se zaměřuje na popis jejich užití se zaměřením na aplikaci v kódování a kryptografii. Pojednává o kryptosystémech DES, AES, RSA a ElGamal. Dále také o variantách Diffie-Hellmanovy výměny klíčů. Pro lepší pochopení problematiky jsou u většiny popisovaných pojmů uvedeny příklady. Na závěr je představen program pro tvorbu šifrovaného souborového systému.

Synopsis

The usage of various algebraic structures, including group theory, can be found in many areas of computer science, although it is not apparent at first sight. The aim of this work is to describe its usage focusing on application at coding and cryptography. It describes cryptosystems like DES, AES, RSA and ElGamal. Also variants of Diffie-Hellman key exchange are mentioned. Most of the described definitions are accompanied with examples for better understanding. In the last part the program for encrypting the file system is introduced.

Klíčová slova: algebraické struktury; teorie grup; kryptografie; šifrovaný souborový systém; Python, Filecrypter

Keywords: algebraic structure; group theory, cryptography; encrypted file system; Python, Filecrypter

Děkuji RNDr. Miroslavu Kolaříkovi, Ph.D., za jeho rady a vedení, které významně přispělo ke kvalitě této práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Bakalářská práce pojednává o oblastech informatiky, ve kterých se lze setkat s užitím různých algebraických struktur, zejména teorie grup. Vytvořením sumarizovaného přehledu, byť se omezuje pouze na vybrané části informatiky, může poskytnout zájemci o problematiku prvotní motivaci pro další studium.

Oblastí, na kterou se práce zaměřuje nejvíce, je kryptografie. Poznamenejme, že se jedná o vědu která zkoumá metody utajování informací (zpráv) tak, že ze zašifrované podoby se k původní zprávě můžeme dostat pomocí specifických znalostí – klíčů. Samozřejmě existují různé metody útoků, které si kladou za cíl rozšifrovat získanou zašifrovanou zprávu bez znalosti původního klíče. Na ně se však práce nezaměřuje a jestliže jsou u vybraných šifrovacích metod některé zmíněny, je jejich popis strohý.

Text práce je rozdělen do čtyř částí. První část seznamuje čtenáře s problematikou algebraických struktur a grup. U příslušných oblastí jsou uvedeny příklady, ze kterých jim lze lépe porozumět. Dále jsou popsány i některé další algebraické pojmy užívané v kryptografii.

Další kapitola se zabývá užitím teorie grup v informatice. Čtenáře stručně uvede například do problematiky grafových automorfismů, samoopravných kódů nebo počítání s velkými čísly. U všech témat zmiňuje oblasti užití v informatice a k vybraným pojmům uvádí příklady.

Ve třetí části se práce zaměřuje na souvislost teorie grup a kryptografie. Po uvedení základních kryptografických pojmů jsou popsány vybrané metody šifrování – AES, DES, ElGamal, šifrování s pomocí kvazigrup a RSA. V neposlední řadě se práce zabývá variantami Diffie-Hellmanovy výměny klíčů. Je zde popsán průběh činnosti předchozích metod a vybrané sekce jsou doplněny jednoduchými příklady. Okrajově jsou zmíněny i jejich slabiny.

Závěr je věnován programu, který aplikuje poznatky z kryptografie. V rámci bakalářské práce byl vytvořen program nazvaný `Filecrypter` pro tvorbu šifrovaného virtuálního souborového systému, který znemožňuje čtení dat bez znalosti potřebného klíče. Je určen pro operační systémy na bázi Linuxu a naprogramovaný v jazyce Python. Pro pokrytí potřeb různých skupin uživatelů byla vytvořena konzolová i grafická verze.

2 Algebraické struktury

V této kapitole se práce zaměřuje na popis vybraných algebraických pojmů, jejichž znalost je nezbytná pro následné pojednání o souvislosti teorie grup a moderní kryptografie. Představení základních definic si klade za cíl přiblížit bakalářskou práci i lidem, kteří neabsolvovali kurzy algebry a učinit pro ně čtení textu srozumitelnější. Definice jsou proloženy příklady, které zastávají zmíněný cíl.

Definice 1 (uspořádaná n-tice)
Zápisem $\langle a_1, a_2, ..., a_n\rangle$ se reprezentuje struktura nazývaná uspořádaná n-tice. Prvek a_1 se nazývá první složkou, a_2 druhou, atd. Jestliže je $n = 2$, hovoříme o uspořádané dvojici, pro $n = 3$ o uspořádané trojici, atd. Na rozdíl od množiny u uspořádané n-tice záleží na pořadí prvků a také se v ní stejný prvek může vyskytovat vícekrát.

Definice 2 (n-tá kartézská mocnina, n-ární relace)
Nechť $A \neq \emptyset$, potom n-tou kartézskou mocninou množiny A rozumíme množinu všech uspořádaných n-tic $\langle a_1, a_2, ..., a_n\rangle$ takových, že $a_i \in A$ pro $1 \leq i \leq n$. Libovolnou podmnožinu n-té kartézské mocniny nazýváme n-ární relací na množině A.

Od hodnoty n se také odvíjí název relace. Pro $n = 0$ nazveme relaci nulární, pro $n = 1$ unární, pro $n = 2$ binární a tak dále.

Příklad 3 (binární relace)
Binární relací na množině prvků $A = \{1, 2, 3, 4, 5\}$ je například množina uspořádaných dvojic $\alpha = \{\langle 1, 1\rangle, \langle 2, 1\rangle, \langle 2, 3\rangle\}$.
Množiny $\beta = \{\langle 1, 1, 4\rangle, \langle 2, 1\rangle, \langle 2, 3\rangle\}$ a $\gamma = \{\langle 1, 1\rangle, \langle 2, 1\rangle, \langle 2, 6\rangle\}$ binárními relacemi na A nejsou. V případě množiny β není prvek $\langle 1, 1, 4\rangle$ uspořádanou dvojicí na A a u prvku $\langle 2, 6\rangle$ z množiny γ platí, že $6 \notin A$.

Definice 4 (n-ární operace)
Nechť $A \neq \emptyset$, potom n-ární operací na množině A nazveme takové zobrazení f, pro které platí $f : A^n \to A$. U binárních operací se často místo značení $f(a, b)$ používá zápis afb.

Příklad 5 (binární operace)
Uvažujme zobrazení $f : \mathbb{Z}^2 \to \mathbb{Z}$, kde $\mathbb{Z}$ je množina celých čísel, dané předpisem $f(a, b) = a \cdot b$. Zobrazení f je uzavřené na $\mathbb{Z}$, jelikož součin dvou celých čísel je celé číslo. Tedy není možné najít prvky j, k, l takové, že $j, k \in \mathbb{Z}$, $l \notin \mathbb{Z}$ a $f(j, k) = l$. Zobrazení f je proto binární operací na množině celých čísel. Oproti tomu však $g(a, b) = a/b$ binární operací není. Pokud zvolíme $a = 1$, $b = 2$, potom $g(1, 2) = 1/2 \notin \mathbb{Z}$.

2.1 Algebraická struktura

Definice 6

Algebraickou strukturou rozumíme obecně množinu prvků A spolu s n-árními operacemi $\sigma_1, \sigma_2, \sigma_3, \ldots$, které jsou na A definovány. Strukturu zapisujeme ve tvaru $\mathbb{A} = \langle A, \sigma_1, \sigma_2, \sigma_3, \ldots \rangle$, kde A je tzv. nosná množina a symboly σ_i reprezentují n-ární operace na množině A.

Jestliže je σ binární operace na A, potom se algebraická struktura $\langle A, \sigma \rangle$ vzhledem k σ nazývá:

- *asociativní* – jestliže pro $\forall a, b, c \in A$ platí $a\ \sigma\ (b\ \sigma\ c) = (a\ \sigma\ b)\ \sigma\ c$,
- *komutativní* – jestliže pro $\forall a, b \in A$ platí $a\ \sigma\ b\ =\ b\ \sigma\ a$.

Prvek $e \in A$ se vzhledem k operaci σ nazývá *neutrální*, pokud pro $\forall a \in A$ platí $a\ \sigma\ e = a = e\ \sigma\ a$.

Inverzním prvkem k prvku $a \in A$ značíme prvek a^{-1} takový, že $a\ \sigma\ a^{-1} = e = a^{-1}\ \sigma\ a$.

Příklad 7

Uvažujme pro demonstraci algebraické struktury nosnou množinu celých čísel $\mathbb{Z}$ a jako binární operaci zvolme násobení celých čísel značené symbolem $\cdot$. Vznikne struktura $\mathbb{A} = \langle \mathbb{Z}, \cdot \rangle$, která reprezentuje násobení celých čísel. $\mathbb{A}$ je asociativní, komutativní, vzhledem k operaci $\cdot$ existuje neutrální prvek $e = 1$, avšak neplatí, že pro každý prvek $a \in \mathbb{Z}$ existuje $a^{-1} \in \mathbb{Z}$. Přítomnost inverzních prvků bychom získali záměnou nosné množiny $\mathbb{Z}$ za $\mathbb{Q} \setminus \{0\}$, kde $\mathbb{Q}$ značí množinu všech racionálních čísel, popř. $\mathbb{Q}^+$.

Další příklad uspořádané dvojice množiny a binární operace, která však algebraickou strukturou není. Nechť $\mathbb{B} = \langle \mathbb{Z}, / \rangle$, kde operace $/$ je klasické dělení. $\mathbb{B}$ není uzavřená vzhledem k $/$, nemůže tedy být ani algebraickou strukturou.

Algebraickou strukturu $\mathbb{A} = \langle A, \circ \rangle$ nazýváme:

- *grupoid*, jestliže je $\circ$ binární operací na A,
- *pologrupa*, jestliže $\mathbb{A}$ je grupoid a zároveň $\circ$ je asociativní,
- *monoid*, jestliže $\mathbb{A}$ je pologrupa a obsahuje neutrální prvek,
- *grupa*, jestliže $\mathbb{A}$ je monoid a pro $\forall a \in A$ existuje prvek $a^{-1} \in A$.

2.2 Grupy

Základní vlastnosti grupy byly napsány výše, nyní si je popišme detailněji. Jedná se o algebraickou strukturu s jednou binární operací, která je asociativní, obsahuje neutrální prvek vzhledem k užívané binární operaci a ke každému prvku

z nosné množiny existuje prvek inverzní, který náleží do stejné množiny. Kromě značení $\mathbb{G} = \langle G, \circ \rangle$ se lze setkat i se zápisem $\mathbb{G} = \langle G, \circ, ^{-1}, e \rangle$, kde G i $\circ$ mají stejný význam (nosná množina, resp. binární operace na množině), $^{-1}$ je unární operace přiřazující prvku z G prvek inverzní a e značí neutrální prvek. Jestliže je navíc grupa komutativní, tedy platí

$$\forall a, b \in G : a \circ b = b \circ a,$$

nazývá se grupa $\mathbb{G}$ abelovskou.

Definice 8

Grupa $\mathbb{G}$ se nazývá konečná, jestliže existuje takové číslo $n \in \mathbb{N}$, že počet prvků G je roven n. Potom se n nazývá řádem grupy $\mathbb{G}$. V opačném případě se hovoří o nekonečné grupě nebo také o grupě nekonečného řádu.

Grupy konečného řádu je možné zapsat do tzv. Cayleyho tabulky. Pro grupu řádu n má tabulka n sloupců a n řádků plus záhlaví, ve kterém jsou vypsány veškeré prvky z nosné množiny. Uvažujme grupu s binární operací $\circ$, prvek a_i v záhlaví i-tého řádku a prvek a_j v záhlaví j-tého sloupce. Na průniku řádku i a sloupce j je prvek odpovídající $a_i \circ a_j$. Reprezentaci grupy Cayleyho tabulkou je vhodné volit pro grupy s nízkým řádem, jelikož s rostoucím počtem prvků je tato reprezentace vzhledem k zvětšování velikosti tabulky nevhodná.

Příklad 9

Ukázkou komutativní grupy nekonečného řádu je struktura $\langle \mathbb{R}, + \rangle$ – množina reálných čísel s operací sčítání. Neutrálním prvkem je 0, inverzním prvkem k číslu $a \in \mathbb{R}$ je číslo opačné, tedy $-a$. Asociativita, resp. komutativita sčítání reálných čísel je dána axiomaticky.

Nechť $\mathbb{A} = \langle \{0, 1, 2, 3, 4, 5, 6\}, \oplus \rangle$ a $\oplus$ značí operaci sčítání modulo 7. Z Cayleyho tabulky 1 lze vidět, že neutrálním prvek je 0. V libovolném řádku tabulky lze nalézt všechny prvky z G, což zaručuje existenci inverzních prvků. Sčítání modulo 7 je asociativní a komutativní, tj. $\oplus$ je asociativní a komutativní operace. Struktura $\mathbb{A}$ je příkladem abelovské grupy řádu 7.

Příkladem struktury, která netvoří grupu je $\mathbb{B} = \langle \{1, 2, 3\}, \otimes \rangle$, kde $a \otimes b = a \cdot b \pmod 4$. V Cayleyho tabulce 2 je ihned vidět, že $2 \otimes 2 = 0$, přičemž $0 \notin \{1, 2, 3\}$.

Tabulka 1: Cayleyho tabulka pro $\mathbb{A}$

$\oplus$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Tabulka 2: Cayleyho tabulka pro $\mathbb{B}$

$\otimes$	1	2	3
1	1	2	3
2	2	0	2
3	3	2	1

Definice 10 (mocnina prvku)
Uvažujme grupu $\mathbb{G} = \langle G, \cdot \rangle$. Pak n-tou mocninu prvku $g \in \mathbb{G}$ spočítáme následovně:

$$\text{pro } n \in \mathbb{Z}_0^+$$

$$g^n = \begin{cases} 1 & \text{pokud } n = 0 \\ g & \text{pokud } n = 1 \\ g \cdot g^{n-1} & \text{jinak,} \end{cases}$$

$$\text{a pro } n \in \mathbb{Z}^-$$

$$g^n = \begin{cases} 1/g & \text{pokud } n = -1 \\ g^{-1} \cdot g^{n+1} & \text{jinak.} \end{cases}$$

Dalším potřebným termínem v následující kapitole je pojem cyklická grupa. Cyklickou grupou nazveme takovou grupu $\mathbb{G} = \langle G, \circ \rangle$, ve které existuje prvek $g \in G$ takový, že $\{g^i, i \in \mathbb{Z}\} = G$. Tedy všechny prvky grupy $\mathbb{G}$ lze vyjádřit jako mocniny prvku g. Prvek g se přitom nazývá generátorem grupy $\mathbb{G}$.

V textu práce se lze setkat také s pojmy kvazigrupa, těleso a polynom nad tělesem. Proto je vhodné seznámit se i s jejich definicemi.

Definice 11 (Kvazigrupa)
Struktura $\mathbb{G} = \langle G, \circ \rangle$ se nazývá kvazigrupa, jestliže pro $\forall a, b \in G \; \exists c, d \in G$ takové, že $a \circ c = b$ a $d \circ a = b$. Hodnotu prvků c, d lze zapsat jako $c = b \backslash a$, resp. $d = a/b$. Obsahuje-li kvazigrupa neutrální prvek, nazývá se lupou.

Na rozdíl od grupy nemusí být operace $\circ$ asociativní. Kvazigrupa je tedy zobecněním pojmu grupa.

Je-li $\mathbb{G}$ konečná, potom Cayleyho tabulka kvazigrupy odpovídá tzv. latinskému čtverci. V každém řádku a každém sloupci se prvek $a \in G$ vyskytuje právě jednou.

Nyní se dostáváme k algebraické struktuře, na které jsou (oproti grupě) zavedeny dvě binární operace.

Definice 12 (Těleso)
Struktura $\mathbb{T} = \langle T, +, \cdot \rangle$ se nazývá těleso, pokud $\langle T, + \rangle$ je abelovská grupa s neutrálním prvkem 0 (nazvaným nulový), $\langle T \setminus \{0\}, \cdot \rangle$ je grupa s neutrálním prvkem 1 (nazvaným jednotkový) a zároveň platí dva distributivní zákony tak, že pro $\forall a, b, c \in T : a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ a $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$. Je-li navíc $\langle T \setminus \{0\}, \cdot \rangle$ abelovskou grupou, hovoříme o komutativním tělese.

Speciálním případem je Galoisovo těleso značené $GF(p^k)$, kde p je prvočíslo, $k \in \mathbb{N}$, nosná množina je $\mathbb{Z}_{p^k}$ a binární operace sčítání a násobení se uvažují modulo p^k.

Definice 13 (Polynom nad tělesem)
Polynom nad tělesem $\mathbb{T}$ neurčité x lze označit $p(x) = a_n x^n + \cdots + a_1 x^1 + a_0$, kde $a_i \in \mathbb{T}$ pro $i \in \mathbb{N}_0$. Stupněm polynomu nazveme takové největší číslo i, pro které platí $a_i \neq 0$. Těleso polynomů neurčité x nad $\mathbb{T}$ značíme $\mathbb{T}[x]$.

2.3 Diskrétní logaritmus

Nechť $a, b, c, m \in \mathbb{Z}$ a $c \equiv a^b \pmod{m}$. Každé číslo b řešící uvedenou rovnici nazveme diskrétním logaritmem o základu a z c. Je snadné dopočítat hodnotu c, jestliže známe a, b, m. Avšak určit b na základě hodnot a, c, m je velmi obtížné a toho využívají některé kryptosystémy popsané v následujících kapitolách.

Příklad 14
Zvolme $a = 7$, $b = 3$, $m = 31$. Pak $c = 2$, jelikož $a^b \pmod{m} \equiv 7^3 \pmod{31} \equiv 2$. Nyní uvažujme pouze znalost prvků $a = 7$, $c = 2$ a $m = 31$, tedy $7^b \pmod{31} \equiv 2$. Tento diskrétní logaritmus řeší všechny $b = 3 + 15n$, kde $n \in \mathbb{N}_0$, jelikož $7^{3+15n} \pmod{31} \equiv 7^3 \cdot (7^{15})^n \pmod{31} \equiv 2 \cdot 1^n \pmod{31} \equiv 2$. Úvahy při výpočtu s většími čísly by byly podobné.

2.4 Eliptická křivka

Definice 15 (Eliptická křivka)
Singulární eliptickou křivkou se rozumí bod v nekonečnu O spolu s množinou

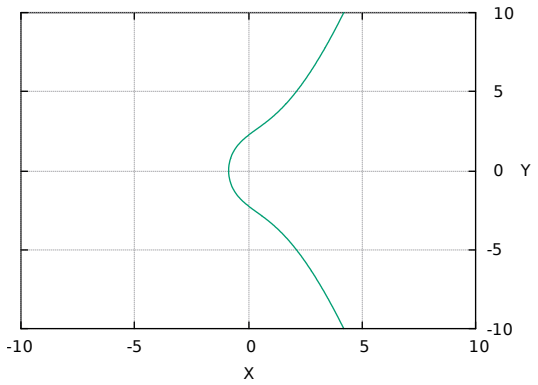
bodů (x, y), které splňují rovnici

$$y^2 = x^3 + ax + b, \tag{1}$$

kde koeficienty a, b jsou předem zvolené prvky příslušného číselného tělesa. Nutnou podmínkou pro to, abychom strukturu mohli nazvat nesingulární eliptickou křivkou je splnění následující nerovnice:

$$4a^3 + 27b^2 \neq 0. \tag{2}$$

Tato podmínka zaručuje, že má rovnice (1) tři různé kořeny. Následující obrázek zobrazuje eliptickou křivku $y^2 = x^3 + 5x + 5$, kde $x, y \in \mathbb{R}$.



Obrázek 1: Graf eliptické křivky $y^2 = x^3 + 5x + 5$

V geometrii prvky a, b, x, y patří do množiny reálných čísel $\mathbb{R}$, avšak při počítačovém zpracování dochází k zaokrouhlování hodnot (zpravidla u iracionálních čísel) a bylo by velmi pravděpodobné, že původní či nově získané body (x, y) již na eliptické křivce ležet nebudou. Proto se volí $a, b, x, y \in GF(p)$. V tomto případě je navíc nutné, aby byla splněna nerovnice (2) vzhledem k příslušnému modulu.

Nad eliptickými křivkami lze provádět operaci sčítání bodů v rovině. Uvažujme body s, t, které jsou prvky eliptické křivky. Jejich součet nalezneme tak, že nejprve vytvoříme přímku m procházející body a a b. Označme $-u$ nový bod, ve kterém přímka m protne eliptickou křivku. Bod u, osově souměrný podle osy x s $-u$ je hledaným součtem bodů s a t. Komplikace nastává, chceme-li sčítat body opačné. Intuitivně $s + (-s) = 0$. Z grafického znázornění eliptické křivky je však patrné, že zkonstruovaná přímka m, na které leží body s a $-s$, v žádném dalším bodě eliptickou křivku neprotíná. Proto se k eliptické křivce přidává nulový bod O interpretovaný jako bod v nekonečnu, který protne přímka procházející opačnými body. Přitom se dodefinuje sčítání opačných bodů $s + (-s) = O$ a nulových bodů $s + O = s$, resp. $O + O = O$.

Příklad 16

Vraťme se k výpočtům s eliptickými křivkami nad $GF(p)$. Zvolme $p = 17$ a

$a = b = 5$. Ověříme podmínku nesingularity: $4 \cdot 5^3 + 27 \cdot 5^2 \pmod{17} \equiv 500 + 675 \pmod{17} \equiv 1175 \pmod{17} = 2$. Podmínka je splněna a zbývá najít body řešící rovnici (1) vzhledem ke zvolenému modulu p. Ty jsou reprezentovány množinou

$$P = \{O, (7,3), (6,9), (6,8), (15,2), (5,11), (5,6), (10,16), (4,15), (12,12), (7,14), (16,4), (8,8), (3,8), (3,9), (10,1), (12,5), (4,2), (8,9), (16,13), (15,15)\}.$$

3 Grupy a informatika

Následující kapitola obsahuje popis několika oblastí informatiky, ve kterých můžeme najít uplatnění grup (a nejen jich). Ukáže jejich roli například u grafového automorfismu nebo samoopravných kódů. Ačkoliv nejsou zdaleka pokryty všechny významné oblasti, z níže uvedeného si můžeme udělat obrázek o tom, jak spolu informatika a algebraické struktury úzce souvisí. Své místo mají také v kryptografii, čemuž se v práci věnuje samostatná kapitola.

3.1 Grafový automorfismus

Poznatky z teorie grafů jsou v informatice často využívány a grafy mají široké uplatnění. Své místo nacházejí v případě stromových struktur, vyhledávání optimální či nejkratší cesty, nebo grafické reprezentaci binárních relací či konečných automatů. Lze nalézt i komplikovanější oblasti informatiky, kde teorie grafů nachází uplatnění. Cílem této podkapitoly je uvedení do základní problematiky teorie grafů a ukázání souvislosti grafového automorfismu, grup a užití v informatice.

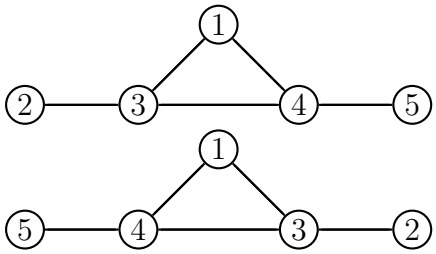
Definice 17 (Neorientovaný graf)
Uspořádaná dvojice $G = \langle V, E\rangle$ se nazývá neorientovaný graf, kde množina V značí neprázdnou množinu vrcholů grafu a E je soubor dvouprvkových množin reprezentující hrany grafu. Pro $\forall\{u, v\} \in E$ platí, že $u, v \in V$.

Orientovaný graf představuje zúžení pojmu neorientovaný graf, kde záleží na směru každé hrany.

Definice 18 (Orientovaný graf)
Uspořádaná dvojice $G = \langle V, E'\rangle$ se nazývá orientovaný graf, pokud V je neprázdná množina vrcholů grafu a $E' \subseteq V \times V$. Pro dvojici $\langle u, v\rangle \in E'$ lze vrchol u chápat jako počátek hrany (resp. šipky při grafickém znázornění), v jako její konec.

Definice 19 (Grafový isomorfismus)
Nechť $G = \langle V, E\rangle$ a $G' = \langle V', E'\rangle$ jsou orientované grafy. Grafovým isomorfismem se rozumí bijektivní zobrazení $f : V \to V'$ takové, že $\langle u, v\rangle \in E \iff \langle f(u), f(v)\rangle \in E'$. Existuje-li takové zobrazení f, grafy G a G' se nazývají isomorfní. Tuto vlastnost značíme $G \cong G'$.

Grafový isomorfismus představuje aparát pro popis shodnosti grafů. Všimněme si, že definice platí pouze pro orientované grafy. Triviálně ji lze zobecnit i na grafy neorientované. Všechny neorientované hrany $\{u, v\}$ stačí nahradit dvojicí orientovaných hran $\langle u, v\rangle$, $\langle v, u\rangle$. Speciálním případem grafového isomorfismu je grafový automorfismus.



Obrázek 2: Automorfismy neorientovaného grafu

Definice 20 (Grafový automorfismus)
Nechť $G = \langle V, E \rangle$ je orientovaný graf. Grafovým automorfismem se rozumí bijektivní zobrazení $g : V \to V$ takové, že $\langle u, v \rangle \in E \iff \langle f(u), f(v) \rangle \in E$.

Výše uvedená definice popisuje symetrii orientovaného grafu, přičemž ji lze použít také u grafu neorientovaného. V horní polovině obrázku 2 je zobrazen graf, který má pouze dva automorfismy – identické zobrazení vrcholu sama na sebe (horní polovina obrázku) a zobrazení představující zrcadlení podle vertikální osy procházející vrcholem 1 (dolní polovina obrázku).

Jelikož je kompozice dvou bijektivních zobrazení opět bijekcí, výsledkem kompozice dvou automorfismů je také automorfismus. Označme $g_1, \ldots, g_n$ všechny automorfismy grafu G. Uspořádaná dvojice $A = \langle \{g_1, \ldots, g_n\}, \circ \rangle$ tvoří grupu všech automorfismů grafu G vzhledem k operaci skládání zobrazení.

Automorfismu se využívá například při dataminingu nebo při optimalizaci kreslení grafu. Po zjištění všech automorfismů daného grafu (např. pomocí `nauty` – programu pro výpočet grup automorfismů zadaného grafu) je možné vygenerovat vizuálně přehledný graf, ze kterého je patrná symetrie. Dále nachází automorfismy využití během redukování velikosti velkých grafů nebo při řešení SAT problému [2], tedy při určování splnitelnosti logických formulí. Jedná se o problém, kdy chceme pro formuli φ na vstupu získat odpověď `ANO` nebo `NE` v závislosti na existenci ohodnocení e, pro které je formule pravdivá. Některé algoritmy řešící problém splnitelnosti formulí vyžadují během své činnosti získání symetrií vstupní formule. Postupně se formule v CNF tvaru (konjunktivní normální formě, neboli konjunkci klauzulí) převede na graf, ze kterého jsou (opět pomocí `nauty`) získány jeho symetrie. Ty se na závěr převedou na symetrie formule v CNF. Detailnější seznámení s pojmy SAT problému je však nad rámec této práce. V neposlední řadě lze nalézt využití symetrií grafu u redukování paměti potřebné pro indexaci nejkratších cest v grafu [28].

3.2 Formální jazyky

Oblast formálních jazyků je v teoretické informatice intenzivně studovanou a užitečnou oblastí. Faktem je, že některé užívané formalismy lze popsat také algebraicky, čehož lze využít při hlubším zkoumání problému a získávání nových poznatků.

Pro připomenutí dodejme, že formálním jazykem L nad abecedou Σ se rozumí libovolná podmnožina Σ^*, přičemž abeceda Σ je jakákoliv neprázdná konečná množina symbolů a množina Σ^* obsahuje všechny různé konečné posloupnosti znaků ze Σ spolu s prázdným řetězcem ϵ. Prvky L se nazývají řetězce, které je možné skládat pomocí operace zřetězení (konkatenace) značené $\circ$. Tedy pro $a, b \in L$: $a \circ b = ab$. Výsledkem konkatenace však nemusí být prvek jazyka L. Triviálním příkladem tohoto tvrzení je výsledek konkatenace $ab \circ ba = abba \notin L$ pro $L = \{ab, ba\}$. Dále platí, že $\epsilon \circ a = a = a \circ \epsilon$. Tyto poznatky stačí k popisu tzv. absolutně volného monoidu nad Σ, kterým je uspořádaná trojice $\langle \Sigma^*, \circ, \epsilon \rangle$.
Při práci s jazyky můžeme definovat další pojmy:
Syntaktické kvaziuspořádání $\leq_L$: $a \leq_L b$ $(a, b \in L)$ právě když pro $\forall u, v \in \Sigma^*$ platí $ubv \in L \implies uav \in L$.
Syntaktická kongruence $\sim_L$: $a \sim_L b$ právě když $a \leq_L b$ a $b \leq_L a$.

Definice 21 (Syntaktický monoid)
Monoid $\mathbb{M} = \langle M, \cdot \rangle$ rozpoznává jazyk L, existuje-li homomorfismus $\varphi : \Sigma^* \to M$ a podmnožina $N \subseteq M$ taková, že $\varphi^{-1}(N) = L$. Je-li $\mathbb{M}$ nejmenší takový, nazývá se syntaktický monoid a lze jej zapsat jako $\mathbb{M} = \langle \Sigma^* / \sim_L, \leq_L / \sim_L \rangle$. Příslušným homomorfismem je $\varphi(a) = a/\sim_L$.

Příklad 22
Nechť $\Sigma = \{0, 1\}$ a $L = \{w \mid w \in \Sigma^* \text{ a } w \text{ je sudé délky}\}$. Nosná množina syntaktického monoidu M obsahuje dva prvky: L a $\Sigma^* \setminus L$. Jedná se o jazyk a jeho doplněk – množinu slov liché délky. Tyto množiny jsou třídy syntaktické kongruence a homomorfismus φ poté zobrazí slovo w do příslušné třídy kongruence podle délky slova w.

Je-li syntaktický monoid příslušný nějakému jazyku konečný, pak je daný jazyk regulární. Touto problematikou se zabývá text [11], který popisuje souvislost mezi regulárními jazyky a velikostí příslušných syntaktických monoidů.

3.3 Samoopravné a samodetekující kódy

Během komunikace mezi zařízeními je očekávané, že během přenosu může nastat nějaká chyba. Může dojít k elektromagnetickému rušení, interferenci signálu a podobně. Proto je velmi důležité mít možnost odhalit, že došlo k chybě a ideálně ji také opravit. S tímto účelem vzniklo hned několik samodetekujících, resp. samoopravných kódů. Některé z nich budou představeny.

Pro porozumění nadcházejícímu textu uveďme několik neformálních definic. Blokový kód m, n je způsob kódování, který data délky m nad vstupní abecedou kóduje na data délky n (kódové slovo) nad abecedou výstupní. Lineární kód je takový blokový kód, kde lineární kombinací několika kódových slov získáme opět kódové slovo. Kódová slova můžeme chápat jako vektory, se kterými provádíme lineární kombinace: $a_1a_2a_3 = (a_1, a_2, a_3)$. Lineární kód nazveme cyklickým kó-

dem, jsou-li jeho kódová slova uzavřena na cyklický posun. Tedy pokud $(a_1, \ldots, a_{n-1}, a_n)$ je kódovým slovem, potom jím je také $(a_n, a_1, \ldots, a_{n-1})$. Uvažujeme-li a_i pouze ze $\mathbb{Z}_2 = \{0, 1\}$, jde o binární kód.

Binární blokový kód $\mathbb{C}$ délky n má kontrolní matici K, jestliže pro $\forall w \in \mathbb{C}$ platí $K \cdot w = \mathbf{0^n}$, kde prvek $\mathbf{0^n}$ značí nulový sloupcový vektor délky n. Důležitým pojmem v teorii kódování je také Hammingova vzdálenost δ. Pro $u, v \in \mathbb{C}$ je $\delta(u, v) = i$, kde i značí počet pozic, na kterých jsou slova u a v různá. Minimální vzdálenost kódu $\mathbb{C}$ je rovna $min(\{\delta(u, v) \mid u, v \in \mathbb{C} \text{ a } u \neq v\})$.

Příklad 23

Příkladem blokového kódu $1, 8$ je rozšířená ASCII tabulka. Každý znak je zakódován na posloupnost bitů délky 8. Například symbol a je roven posloupnosti bitů 01100001.

Pro abecedu $A = \{a, b, c, d\}$ zkonstruujme blokový kód $1, 2$, který je lineární. Kombinací libovolných dvou slov z kódu získáme opět slovo do něj patřící.

Tabulka 3: Blokový kód $1, 2$

znak	kódové slovo
a	00
b	01
c	10
d	11

Shodou okolností je kód v tabulce 3 cyklický.

3.3.1 CRC kód

CRC kód, nebo také cyklický redundantní součet, je druh samodetekujícího kódu, který se v informatice často využívá k ověření konzistence dat. Data délky m bitů, pro která chceme CRC kód délky n spočítat, reprezentujeme jako polynom g řádu $m-1$: např. pro 1011 je $g = x^3 + x + 1$. V závislosti na hodnotě n je nutné zvolit tzv. řídící polynom h řádu n. Dodejme, že polynomy g a h jsou z tělesa $GF(2)[x]$. Výsledný CRC kód je roven posloupnosti koeficientů polynomu c, který vznikne jako zbytek po dělení $x^n \cdot g$ polynomem h. Jelikož koeficienty polynomu c nemusí být z množiny $\{0, 1\}$, provede se s nimi operace modulo 2.

Příklad 24

Spočítejme CRC16 (kontrolní součet délky 16 bitů) pro vstupní data 0x2000A, kde prefix 0x značí číslo v šestnáctkové soustavě. Pro CRC16 se typicky volí $h = x^{16} + x^{15} + x^2 + 1$. Data 0x2000A představují polynom $g = x^{17} + x^3 + x$. Dělíme tedy $x^{33} + x^{19} + x^{17} = x^{16} \cdot (x^{17} + x^3 + x)$ polynomem h. Zbytek po dělení je polynom $c = -5x^{15} + 2x^{14} - 2x^{13} + 2x^{12} - 2x^{11} + 2x^{10} - 2x^9 + 2x^8 - 2x^7 + 2x^6 - 2x^5 + x^4 - 3x^2 + x - 3 \equiv x^{15} + x^4 + x^2 + x + 1 \pmod 2$. Převedeme-li koeficienty polynomu c na posloupnost nul a jedniček, výsledný CRC kód je 0x8017.

3.3.2 Hammingův kód

Jedná se o skupinu binárních lineárních kódů, které jsou schopny detekovat dvě chyby a jednu chybu opravit. Mají délku $n = 2^m - 1$, kde m značí počet paritních (kontrolních) bitů. V kódu délky $n = 7$ (kde $m = 3$) jsou čtyři bity informační a tři paritní – jedná se o $(7, 4)$ kód a hodnota paritních bitů je odvozena z kódovaných dat. Kontrolní matice K má vždy velikost $m \times (2^m - 1)$, přičemž platí že neobsahuje nulové sloupce. Korektnost obdrženého slova w zjistíme vynásobením spolu s kontrolní maticí (uvažujme slova jako vektory) a získáme sloupcový vektor, tzv. syndrom: $K \cdot \mathbf{w} = \mathbf{0^m}$. Není-li syndrom nulový, během přenosu nastala chyba a syndrom odpovídá i-tému sloupci kontrolní matice, který binárně vyjadřuje pozici chyby. Generující matice velikosti $(n - m) \times n$ obsahuje bázi kódu a platí, že vstupní data reprezentována vektorem $\mathbf{a}$ zakódujeme vynásobením s generující maticí: $\mathbf{a} \cdot G = \mathbf{w}$. Hammingovy kódy mají minimální vzdálenost tři. Uplatnění Hammingových kódů lze nalézt například u počítačových pamětí, které implementují detekci chyb či v oblasti telekomunikací.

Příklad 25

Sestavme zmiňovaný Hammingův $(7, 4)$ kód. Postupujeme-li podle obecného algoritmu na tvorbu Hammingova kódu [26], pak jsou pozice kódového slova odpovídající mocnině dvou brány jako paritní bity. Nechť $w = w_1w_2w_3w_4w_5w_6w_7$ je kódovým slovem Hammingova $(7, 4)$ kódu, pak prvky w_1, w_2 a w_4 představují paritní bity. Dle zmíněného algoritmu mají generující a kontrolní matice následující podobu:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \; K = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Uvažujme zdrojové slovo dané vektorem $\mathbf{a} = (1, 0, 1, 0)$. V našem Hammingově kódu je $\mathbf{b} = \mathbf{a} \cdot G = (1, 0, 1, 1, 0, 1, 0)$. Dále předpokládejme, že během přenosu došlo k chybě a příjemce obdržel $\bar{\mathbf{b}} = (1, 1, 1, 1, 0, 1, 0)$. Změna tedy proběhla na druhé pozici slova $\mathbf{b}$. Spočítáme $K \cdot \bar{\mathbf{b}}^T = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. V tomto případě je chyba na druhé pozici slova $\bar{\mathbf{b}}$ a příslušný bit invertujeme. To odpovídá původně odeslanému slovu. Je vhodné dodat, že právě díky vhodně zvolenému uspořádání kontrolní matice stačí pouze spočítat syndrom a hodnotu jeho sloupce rovnou převést do desítkové soustavy.

3.3.3 BCH kód

Skupina BCH kódů se řadí mezi cyklické samoopravné kódy. Při jejich konstrukci lze takřka libovolně zvolit počet opravitelných chyb. Nejen díky těmto vlastnostem se kód využívá při ukládání dat na pevné disky, v multimediálních

přehrávačích nebo při satelitní komunikaci. BCH kód je určen několika parametry – délkou n, abecedou p a vzdáleností d (Hammingova vzdálenost je $\geq d$). BCH kód délky n nad $A = \mathbb{Z}_p$ (p je nesoudělné s n) s plánovanou vzdáleností d ($d \leq n$) je cyklický kód s generujícími kořeny $\beta^b, \beta^{b+1}, \ldots, \beta^{b+d-2}$, kde β je prvek řádu n v nějakém tělese $B = GF(p^k)$, $b \geq 1$. O takovém kódu řekneme, že je schopen detekovat $d-1$ chyb a opravit $\lfloor \frac{d-1}{2} \rfloor$ chyb. Možnou variantou je tzv. binární BCH kód nad $\mathbb{Z}_2$, kde $b = 1, d = 2t+1$ a $\beta \in GF(2^k)$. Takový kód opravuje alespoň t chyb. Jestliže je $A = B$, hovoříme o Reed-Solomonovu kódu – samoopravném kódu využívaném při kódování CD/DVD, QR kódů, DSL či u technologie RAID6. Část textu o BCH kódech čerpá z [5], kde lze nalézt více informací.

3.4 Vektorové prostory

Vektorové prostory jsou v matematice a informatice velmi užívanou oblastí. Nacházejí uplatnění při řešení matematických problémů či v diferenciální geometrii. Jako vektory se uvažují některé prvky v teorii kódování a v neposlední řadě mají své místo v počítačové grafice. Nalezení algebraických struktur ve vektorových prostorech je snadné.

Mějme dánu neprázdnou množinu V, jejíž prvky nazýváme vektory, která spolu s operací $\oplus$ (sčítání vektorů) tvoří komutativní grupu, kde nulový vektor $\bar{\mathbf{o}}$ je neutrální prvek. Dále potřebujeme číselné těleso T a levou vnější operaci $\odot : T \times V \to V$ představující násobení vektoru skalárem. Čtveřici $(V, \oplus, T, \odot)$ nazveme vektorovým prostorem, splní-li pro libovolné prvky $a, b \in T$, $\mathbf{x}, \mathbf{y} \in V$ následující čtyři axiomy: $a \odot (b \odot \mathbf{x}) = (a \cdot b) \odot \mathbf{x}$, $a \odot (\mathbf{x} \oplus \mathbf{y}) = a \odot \mathbf{x} \oplus a \odot \mathbf{y}$, $(a + b) \odot \mathbf{x} = a \odot \mathbf{x} \oplus b \odot \mathbf{x}$, $1 \odot \mathbf{x} = \mathbf{x}$.

Vektorové prostory mají v informatice široké uplatnění – lze je nalézt v teorii kódování, své místo nachází při řešení některých problémů z teorie grafů, nebo třeba při konstrukci textových vyhledávačů (např. Apache Lucene). V neposlední řadě lze nalézt aplikace vektorových prostorů v počítačové grafice (např. při zpracování obrazu a videa, modelování pomocí OpenGL, ...).

3.5 Počítání s velkými čísly

Je běžné, že v informatice potřebujeme spočítat součin dvou velkých čísel, případně mocninu nějakého čísla s velkým exponentem. Počítáme-li modulo nějaké n, je možné výpočty významně urychlit užitím Čínské věty o zbytcích.

Věta 26 (Čínská věta o zbytcích)
Nechť $m_1, \ldots, m_n$ jsou po dvou nesoudělná přirozená čísla, označme $m = m_1 \cdot \cdots \cdot m_n$. Pak pro libovolná celá čísla $a_1, \ldots, a_n$ existuje právě jedno $b \in \{0, \ldots, m-1\}$, které řeší soustavu kongruencí

$$b \equiv a_1 \pmod{m_1}, \ \ldots, \ b \equiv a_n \pmod{m_n}.$$

Čínská věta o zbytcích umožňuje zjistit konkrétní hodnotu nějakého čísla, víme-li pouze do jaké třídy kongruence patří vzhledem k danému modulu. Dále uveďme tzv. Malou Fermatovu větu:

Věta 27 (Malá Fermatova věta)
Pokud je m prvočíslo, potom pro každé přirozené číslo a, nesoudělné s m, platí

$$a^{m-1} \pmod{m} \equiv 1.$$

V nadcházejícím příkladu bude ještě užitečný algoritmus pro řešení soustavy kongruencí [8]:

1. Označme $m = m_1 m_2 \dots m_n$ a $M_i = \frac{m}{m_i}$ pro $1 \leq i \leq n$.
2. Pro každé i nalezněme inverzní prvek k M_i vzhledem k příslušnému násobení modulo m_i.
3. Nechť $b = \sum_{i=1}^{n} a_i M_i x_i$. Množina všech řešení soustavy je $\{b + km; k \in \mathbb{Z}\}$.

Příklad 28
Chtějme spočítat $12^{3141} \pmod{85}$. Jelikož číslo 85 jde rozdělit na součin čísel 17 a 5, můžeme vytvořit dvě následující kongruenční rovnice:

$$12^{3141} \pmod{5} \equiv 12^{4 \cdot 785 + 1} \pmod{5} \equiv 12 \pmod{5} \equiv 2 \pmod{5},$$
$$12^{3141} \pmod{17} \equiv 12^{16 \cdot 196 + 5} \pmod{17} \equiv 12^5 \pmod{17} \equiv 3 \pmod{17}.$$

Užitím Malé Fermatovy věty jsme výrazně snížili hodnoty exponentů, jelikož $12^{4 \cdot 785} \pmod{5} \equiv 1$ a $12^{16 \cdot 196} \pmod{17} \equiv 1$. Označme $x = 12^{3141}$ a spočítejme kongruenční rovnice

$x \equiv 2 \pmod{5}$	$x \equiv 3 \pmod{17}$	
$M_1 = 17$	$M_2 = 5$	Zapíšeme příslušné M_i.
$x_1 \equiv 17^{-1} \pmod{5}$	$x_2 \equiv 5^{-1} \pmod{17}$	Nalezneme inverzní prvky pro M_i.
$17x_1 \equiv 1 \pmod{5}$	$5x_2 \equiv 1 \pmod{17}$	
$2x_1 \equiv 1 \pmod{5}$	$5x_2 \equiv 1 \pmod{17}$	
$x_1 = 3$	$x_2 = 7$	Určíme nejmenší kladné řešení.

Nyní můžeme dopočítat $b = \sum_{i=1}^{n} a_i M_i x_i$. Rozepišme $b = 2 \cdot 17 \cdot 3 + 3 \cdot 5 \cdot 5 = 207$, což lze ještě zjednodušit vzhledem k příslušnému modulu a dostáváme, že $b = 37$.

Platí, že $12^{3141} \equiv 37 + k \cdot 85 \pmod{85}$ pro $k \in \mathbb{Z}$. Takový postup pro výpočet zbytku po dělení velké mocniny nějakého čísla využívá například kryptosystém RSA.

4 Grupy a kryptografie

Jak bylo zmíněno v úvodu předchozí kapitoly, užití grup lze velmi často nalézt také v kryptografii. Převážně tomu tak je kvůli počítání modulo nebo využití eliptických křivek, okrajově se lze setkat s kryptografií pomocí kvazigrup. Jejich konkrétní aplikaci se věnuje následující podkapitola. Dále se seznámíme s populárními kryptosystémy DES, AES a RSA. V neposlední řadě si popíšeme Diffie-Hellmanův protokol pro výměnu klíčů, kde má své místo také diskrétní logaritmus.

Definice 29 (Symetrická šifra)
Symetrickou šifrou se nazývá taková šifra, která k zašifrování i dešifrování používá pouze jeden stejný klíč.

Je tedy nutné, aby obě strany komunikačního kanálu předem znaly použitý klíč, případně si jej nějak sdělily. Což je sice oproti asymetrickým šifrám značná nevýhoda, avšak výpočetní výkon potřebný k provedení šifrování (dešifrování) je mnohonásobně menší. Následuje obecná definice asymetrické šifry.

Definice 30 (Asymetrická šifra)
Metoda šifrování, která pro šifrování a dešifrování využívá dva odlišné klíče, se nazývá asymetrická šifra.

Často se lze setkat s užitím tzv. veřejného a soukromého klíče, například u šifrované emailové komunikace, kde zúčastněné osoby drží svůj privátní klíč v tajnosti a veřejný je dostupný ostatním.

Příklad 31
Uvažme situaci, kdy chce Alice zaslat šifrovaně Bobovi email. Vytvořenou zprávu zašifruje pomocí Bobova veřejného klíče a odešle ji. I když by byl komunikační kanál narušený Chuckem a ten odposlechl odeslanou zprávu, bez Bobova privátního klíče je pro něho velmi obtížné dostat se k původní zprávě.

Jak již bylo zmíněno výše, v moderní kryptografii lze grupy nalézt téměř u každé šifrovací metody, která využívá operaci sčítání modulo 2. Množina $\{0, 1\}$ totiž s operací sčítání modulo 2 tvoří grupu, což se dokáže stejně jako v příkladu 9. Pro úplnost následuje Cayleyho tabulka 4. Z pohledu logických operací se jedná o funkci XOR, neboli exkluzivní disjunkci.

Tabulka 4: Cayleyho tabulka pro $\langle \{0, 1\}, \oplus \pmod 2 \rangle$

$\oplus$	0	1
0	0	1
1	1	0

4.1 Šifrování pomocí kvazigrup

Jednou z méně užívaných oblastí kryptografie je šifrování pomocí kvazigrup. Jeho praktickým uplatněním se zabývají například texty [9], [17], [22].

Nejprve předpokládejme vstupní abecedu A spolu s operacemi $\cdot$ a $\backslash$ (jedná se o násobení a levé dělení), které dohromady tvoří kvazigrupu $\mathbb{A} = (A, \cdot, \backslash)$. Tzv. dílčí šifrování je definované jako zobrazení $e_v(a_1a_2 \dots a_n) = b_1b_2 \dots b_n$, kde $a_i, b_i, v \in A$ a prvek v je odvozen z tajného klíče. Pro každé b_i platí, že

$$b_i = \begin{cases} v \cdot a_1 & \text{pokud } i = 1, \\ b_{i-1} \cdot a_i & \text{jinak.} \end{cases}$$

Podobně je definováno i dílčí dešifrování $d_v(b_1b_2 \dots b_n) = a_1a_2 \dots a_n$, kde

$$a_i = \begin{cases} v \backslash b_1 & \text{pokud } i = 1, \\ b_{i-1} \backslash b_i & \text{jinak.} \end{cases}$$

Výsledná šifrovací funkce E_k je odvozena jako kompozice dílčích šifrování s využitím tajného klíče $k = k_1k_2 \dots k_m$ ($k_i \in A$), jelikož samotné dílčí šifrování neposkytuje dostatečné zabezpečení. Píšeme $E_k = e_{k_1} \circ e_{k_2} \circ \dots \circ e_{k_m}$. V případě dešifrovací funkce D_k je kompozice skládána v opačném pořadí: $D_k = d_{k_m} \circ d_{k_{m-1}} \circ \dots \circ d_{k_1}$. Dále platí, že $D_k(E_k(w)) = E_k(D_k(w)) = w$.

Tabulka 5: Cayleyho tabulka pro $\mathbb{A} = \langle A, \cdot \rangle$

$\cdot$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	f	2	e	3	d	4	c	5	b	6	a	7	9	8
1	1	2	0	3	f	4	e	5	d	6	c	7	b	8	a	9
2	2	3	1	4	0	5	f	6	e	7	d	8	c	9	b	a
3	3	4	2	5	1	6	0	7	f	8	e	9	d	a	c	b
4	4	5	3	6	2	7	1	8	0	9	f	a	e	b	d	c
5	5	6	4	7	3	8	2	9	1	a	0	b	f	c	e	d
6	6	7	5	8	4	9	3	a	2	b	1	c	0	d	f	e
7	7	8	6	9	5	a	4	b	3	c	2	d	1	e	0	f
8	8	9	7	a	6	b	5	c	4	d	3	e	2	f	1	0
9	9	a	8	b	7	c	6	d	5	e	4	f	3	0	2	1
a	a	b	9	c	8	d	7	e	6	f	5	0	4	1	3	2
b	b	c	a	d	9	e	8	f	7	0	6	1	5	2	4	3
c	c	d	b	e	a	f	9	0	8	1	7	2	6	3	5	4
d	d	e	c	f	b	0	a	1	9	2	8	3	7	4	6	5
e	e	f	d	0	c	1	b	2	a	3	9	4	8	5	7	6
f	f	0	e	1	d	2	c	3	b	4	a	5	9	6	8	7

Příklad 32

Předpokládejme, že chceme šifrovat všechny znaky z ASCII tabulky, celkem

256 znaků, kde lze každý z nich vyjádřit jako dvojici číslic v šestnáctkové soustavě. Této skutečnosti využijeme – jako množinu A zvolme znaky šestnáctkové soustavy. Pro multiplikativní operaci $\cdot$ uveďme náhodně vygenerovanou (pomocí skriptu pro generování latinských čtverců [24]) Cayleyho tabulku 5. Námi šifrované slovo *Ahoj* odpovídá posloupnosti $w = 41686f6a$. Úlohu zjednodušíme tím, že jako klíč zvolíme krátké slovo xy, potom $k = 7879$. Pro E_k (D_k) platí $E_k = e_7 \circ e_8 \circ e_7 \circ e_9$ ($D_k = d_9 \circ d_7 \circ d_8 \circ d_7$), tedy

$$\begin{aligned} E_k(41686f6a) &= e_7(e_8(e_7(e_9(41686f6a)))) = e_7(e_8(e_7(7851e63e))) = \\ &= e_7(e_8(c8bc524d)) = e_7(1d3d0fd4) = 8f188075, \\ D_k(8f188075) &= d_9(d_7(d_8(d_7(8f188075)))) = d_9(d_7(d_8(1d3d0fd4))) = \\ &= d_9(d_7(c8bc524d)) = d_9(7851e63e) = 41686f6a. \end{aligned}$$

4.2 Data Encryption Standard

Data Encryption Standard, zkráceně DES, je symetrická šifra vyvinutá v USA v 70. letech minulého století. Tato metoda utajování informací se stala na dlouhou dobu nejrozšířenější pro šifrování dat.

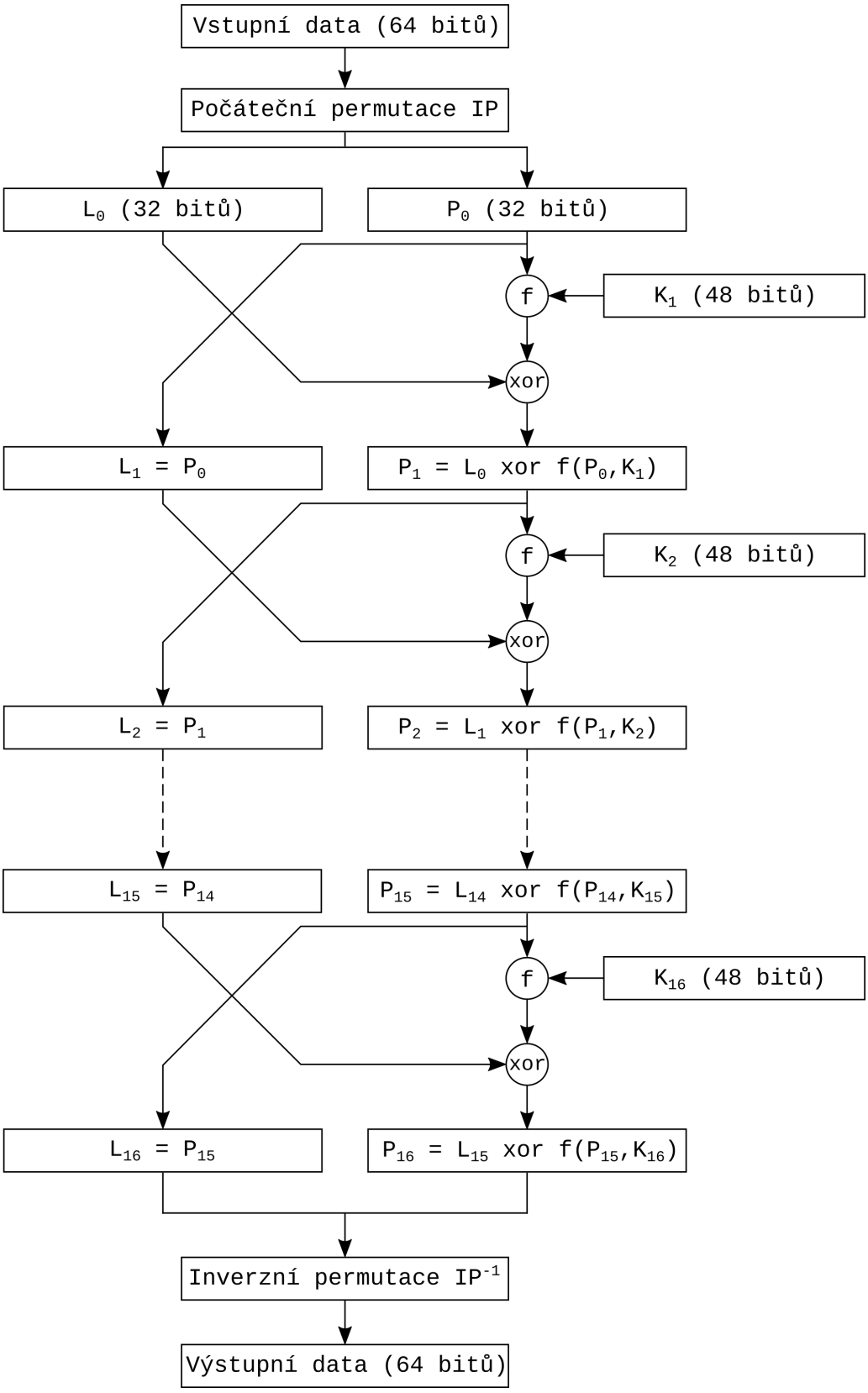
K zašifrování zprávy je kromě dat samotných potřeba ještě šifrovací klíč $\mathbb{K}$ o délce 56 bitů. Klíč je posléze rozšířen na 64 bitů, kde jsou na pozicích 8, 16, ..., 64 umístěny paritní bity. Během průběhu šifrovacího algoritmu se využívá pevně daných permutací a substitucí. Ty však nejsou v textu všechny uvedeny, lze je případně nalézt v literatuře [25]. Zjednodušeně, při výpočtu šifry pracuje DES následovně:

- Ze vstupního klíče $\mathbb{K}$ se vygeneruje šestnáct klíčů $\mathbb{K}_1,\ldots,\mathbb{K}_{16}$ o délce 48 bitů. Tento postup generování dalších klíčů se nazývá expanze klíčů.
- Vstupní data se rozdělí na bloky o délce 64 bitů. Poslední blok se na potřebnou délku doplní.
- Každý blok dat se šestnáctkrát iterativně zašifruje s využitím klíčů $\mathbb{K}_1,\ldots,\mathbb{K}_{16}$.

Následující odstavce popisují činnosti kryptosystému DES podrobněji.

4.2.1 Expanze klíčů

Nejprve se ze vstupního klíče $\mathbb{K}$ rozšířeného o paritní bity získá pomocí permutace $PC-1$ původní 56-bitový klíč. Zbylé je možno použít pro kontrolu parity. Tento 56-bitový klíč je rozdělen na levou a pravou polovinu (LP, PP), každá má tedy velikost 28 bitů. Na základě toho, o kolikátou generaci subklíče se jedná je levá i pravá polovina, nezávisle na sobě, bitově posunuta doleva o jeden nebo dva bity. Poté je pomocí permutace $PC-2$ vytvořen subklíč o velikosti 48 bitů, který využívá 24 bitů z LP a 24 bitů z PP. Nakonec je LP i PP opět posunuta a proces tvorby nového subklíče se opakuje.



Obrázek 3: Průběh šifrování jednoho bloku dat u DES

4.2.2 Šifrování

Nyní je připraveno šestnáct 48-bitových klíčů a data potřebná k zašifrování jsou rozdělena do bloků po 64 bitech. Každý blok dat se zašifruje dle popisu níže.

Na vstupní blok dat se aplikuje počáteční permutace IP, kterou lze rozdělit na levou (L_0) a pravou (P_0) polovinu. L_0 i P_0 mají tedy délku 32 bitů. Požadovaným výstupem je provedení inverzní permutace IP^{-1} na zřetězení P_{16} a L_{16} do jednoho 64 bitového bloku dat. K němu se dospěje pomocí iterací (tzv. rund) následovně:

$$L_i = P_{i-1}$$
$$P_i = L_{i-1} \oplus f(P_{i-1}, K_i).$$

V každé iteraci se z pravé poloviny stane nová levá a nová pravá polovina vznikne provedením operace XOR na předchozí levou polovinu s výsledkem Feistelovy funkce f aplikované na předchozí pravou polovinu a subklíč příslušný dané rundě.

4.2.3 Feistelova funkce f

Parametry funkce f jsou 32 bitový blok dat a 48 bitový subklíč příslušný dané rundě. Poté proběhne výpočet rozdělený do čtyř kroků. Nejprve se pomocí permutace E expanduje blok dat na délku 48 bitů. Ta je nyní shodná s délkou subklíče. Tento výsledek označme jako M_0. Ve druhém kroku se provede operace XOR s M_0 a příslušným subklíčem, vzniká M_1. Těchto 48 bitů (M_1) se opět rozdělí, nyní do osmi bloků po šesti bitech. Ty jsou vstupy pro tzv. *S-boxy* ($S_1, \ldots, S_8$), které každým šesti bitům na vstupu přidělují čtyřbitový výstup v závislosti na hraničních a vnitřních bitech. Například pro vstup 101001 jsou hraničními bity 11 a vnitřními bity 0100. S-boxy jsou stejně jako všechny permutace ve standardu DES pevně definovány. Závěrem se výsledky $S_1, \ldots, S_8$ spojí a provede se na nich permutace P.

4.2.4 Dešifrování

Dešifrování probíhá u DES totožně jako proces šifrování, avšak pořadí použití subklíčů je inverzní, tedy $K_{16}, \ldots, K_1$. Implementačně se jedná o triviální záležitost a není nutné programovat dešifrovací funkci samostatně.

4.2.5 Bezpečnost DES

Ačkoliv je implementace šifry DES snadná a nějaký čas byl takový způsob šifrování dat dostatečný, je v dnešní době již překonaný. V roce 1998 byla šifra prolomena za 56 hodin, o rok později za méně než polovinu předchozího času s použitím DES crackeru sdružení EFF. Vzhledem k ohrožení útoky hrubou silou (anglicky brute-force – postupně se zkouší všechny možné kombinace hesla) vznikla varianta zvaná Triple-DES, která využívá trojí aplikace DES spolu s třemi šifrovacími klíči.

4.3 Advanced Encryption Standard

Algoritmus pojmenovaný Advanced Encryption Standard, zkráceně AES, vznikl jako následník DES šifry vzhledem k úspěšným pokusům o její prolomení. Opět se jedná o blokovou symetrickou šifru. Na vhodnou podobu algoritmu byla v roce 1997 vyhlášena veřejná soutěž, která požadovala velikost bloku šifrovaných dat 128 bitů a tři možné délky klíče: 128, 192 a 256 bitů. Po čtyřech letech byl vybrán algoritmus Rijndael (*„rejndál“*), jehož název vychází ze jmen tvůrců – Daemen a Rijmen. Podobně jako u DES, i zde probíhá šifrování vstupních dat iterativně v $\mathbb{I}$ iteracích (10, 12, resp. 14 pro klíč délky 128, 192, resp. 256 bitů). AES při svém průběhu využívá několik klíčových operací – `AddRoundKey`, `MixColumns`, `ShiftRows` a `SubBytes`. Vstupní blok dat je reprezentován jako matice bytů velikosti 4×4, v textu pojmenovaná $\mathbb{M}$.

4.3.1 Šifrování

Ve stručnosti lze činnost algoritmu, pro délku klíče 128 bitů, popsat následovně:

- Provede se expanze klíčů.
- Ke vstupnímu bloku dat se pomocí `AddRoundKey` přičte příslušný šifrovací klíč. Výsledek se uloží zpět do $\mathbb{M}$.
- Pro iteraci $1, \ldots, \mathbb{I} - 1$ se na $\mathbb{M}$ aplikují postupně funkce `SubBytes`, `MixColumns` a `AddRoundKey`. Výsledky operací se uloží do $\mathbb{M}$.
- V poslední iteraci se aplikují pouze operace `SubBytes` a `AddRoundKey`. Matice $\mathbb{M}$ nyní obsahuje zašifrovaná data.

4.3.2 Užívané operace a matice

$$\begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix} \qquad \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

(a) Počáteční stav matice $\mathbb{M}$.

(b) Matice $\mathbb{B}$.

Obrázek 4: Použité matice

AddRoundKey
Na byte v i-tém řádku a j-tém sloupci v $\mathbb{M}$ se provede operace XOR s bytem na téže pozici z matice subklíče dané rundy.
MixColumns
V této operaci se matice $\mathbb{M}$ vynásobí maticí $\mathbb{B}$, která obsahuje prvky z tělesa $GF(2^8)$. Jednotlivé byty během násobení představují polynomy, kde nejvyšší bit

přísluší nejvyššímu koeficientu polynomu. Např. 0xF0 reprezentuje polynom $x_7\alpha^7 + x_6\alpha^6 + x_5\alpha^5 + x_4\alpha^4$. Násobení probíhá modulo polynom $m(\alpha) = \alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$, operaci sčítání vyjadřuje XOR. Koeficienty výsledného polynomu se uloží zpět na příslušné místo v $\mathbb{M}$.

ShiftRows

Jedná se o operaci posunující řádky matice s uloženými daty. Pro $i = 0, \ldots, 3$ se řádek i posune o i sloupců doprava.

SubBytes

Operace se stará o záměnu bytů pomocí S-boxu, podobně jako u DES. Rozdíl je v tom, že AES využívá jeden veliký S-box velikosti 16×16. Byte XY se nahradí bytem na průniku řádku X se sloupcem Y z S-boxu.

4.3.3 Expanze klíčů

Pro činnost algoritmu je zapotřebí mít 11 klíčů pro každou iteraci, každý o délce 16 bytů. S klíči se opět pracuje jako s maticemi velikosti 4×4. Sloupce původního klíče označme jako W_0, W_1, W_2 a W_3. Pro $i = 4, \ldots, 43$ se postupuje následovně:

$$W_i = \begin{cases} W_{i-4} \oplus W_{i-1} & \text{pokud } i \pmod 4 \neq 0 \\ W_{i-4} \oplus V & \text{jinak.} \end{cases}$$

$V = (v_0, v_1, v_2, v_3)$ je sloupec vzniklý z $W_{i-1} = (w_0, w_1, w_2, w_3)$. Platí, že $v_j = \texttt{SubBytes}(w_{j+1 \pmod 4})$ pro $j \in \{1, 2, 3\}$. Dále $v_0 = \texttt{SubBytes}(w_1) \oplus 2^{r-1}$, kde $r = i/4$ značí číslo odvozovaného klíče. Poznamenejme, že pomocný sloupec V se počítá pouze pro $i \in \{4 \cdot j \mid 1 \leq j \leq 10\}$. Matice získané spojením $W_{4k}, W_{4k+1}, W_{4k+2}$ a W_{4k+3} pro $0 \leq k \leq 10$ představují nových 11 klíčů.

4.3.4 Dešifrování

Obdobně jako u DES se při dešifrování provádějí operace v opačném pořadí. Navíc je potřeba operace `AddRoundKey`, `ShiftRows` a `SubBytes` nahradit jejich inverzí. `MixColumns` je inverzní sama k sobě.

4.3.5 Bezpečnost AES

Oproti svému předchůdci, DES, je tato šifra považována za neprolomitelnou, jestliže je korektně implementována. Existují různé teoretické verze útoků na šifru, ty však počítají s nižším počtem iterací příslušných k délce klíče. Zároveň jejich časová i paměťová složitost je natolik veliká, že lze s dnešním hardwarem považovat AES za bezpečnou. Šifra je však do jisté míry náchylná k útoku postranním kanálem. Při něm se útočník nesnaží získat heslo nedostatky v návrhu kryptosystému, ale využívá analýzy fyzikálního stavu zařízení – např. sledování změny vydávaného elektromagnetického pole počítače, kolísání napětí nebo sledování doby potřebné k vykonání určitých operací. Využití tohoto útoku na AES-128 bylo publikováno v roce 2010 a k úspěchu vyžadovalo spuštění vlastního kódu na zařízení, které se staralo o šifrování [7].

4.4 Diffie-Hellmanova výměna klíčů

Diffie-Hellmanova výměna klíčů (DH) je kryptografický protokol sloužící k vytvoření tajného, sdíleného klíče při navazování šifrovaného spojení. Tato metoda byla objevena Whitfieldem Diffiem a Martinem Hellmanem v roce 1976. Jedná se o populární metodu, která je implementována v rámci protokolů SSH, SSL/TLS, IPsec nebo se používá pro výměnu klíčů před následným šifrováním, například pomocí AES.

4.5 Činnost DH

Typickým příkladem pro popis průběhu DH je situace, kdy se dva lidé, Alice a Bob, chtějí dohodnout na šifrovacím klíči po veřejném kanálu, díky kterému utají následnou komunikaci.

1. Účastníci se dohodnou na prvočíslu p, resp. grupě $\langle \mathbb{Z}_p, \cdot \rangle$ a jejím generátoru g.
2. Alice vybere privátní klíč $a \in \mathbb{Z}_p$, Bobovi zašle veřejný klíč $A = g^a \pmod p$.
3. Bob zvolí privátní klíč $b \in \mathbb{Z}_p$, $B = g^b \pmod p$ předá Alici.
4. Číslo $s = A^b \pmod p = (g^a)^b \pmod p = g^{ab} \pmod p = (g^b)^a \pmod p = B^a \pmod p = s'$. Alice a Bob nyní disponují údaji, ze kterých mohou odvodit sdílený klíč, kterým je s, resp. s'.

Příklad 33

Alice se s Bobem dohodla na grupě $\langle \mathbb{Z}_{314159}, \cdot \rangle$ a číslu 23, které je jejím generátorem. Jako privátní klíč pseudonáhodně zvolila $a = 85516$. Veřejný klíč je tedy $A = 214488$ a může být zaslán Bobovi. Mezitím si Bob také zvolil privátní klíč $b = 126106$, z něho vyplývající veřejný klíč $B = 261408$ také předal Alici. Alice nyní ze znalosti svého privátního klíče a Bobova veřejného klíče spočítá sdílený klíč: $261408^{85516} \pmod{314159} = 3540$. Podobně postupuje i Bob: $214488^{126106} \pmod{314159} = 3540$. Nově získaný klíč mohou použít pro zabezpečenou komunikaci pomocí nějaké symetrické šifry.

4.5.1 Bezpečnost DH

DH se spoléhá na obtížnost vyřešení problému diskrétního logaritmu v rozumném čase za předpokladu, že se zvolí prvočíselná grupa $\langle \mathbb{Z}_p, \cdot \rangle$ s dostatečně velkým prvočíslem p. Protokol je však náchylný na man-in-the-middle (MIM) útoky, tedy situaci, kdy se útočník dostane mezi klienta komunikujícího se serverem a je schopný odposlechnout jejich komunikaci, byť zabezpečenou. Těm se lze však bránit například přidáním autentizačního mechanizmu. Nedávno byly objeveny další slabiny – Logjam [1] a používání stejných prvočísel na straně serveru. Doporučením je přejít ke kryptografii nad eliptickými křivkami, popř. využívat $p \geq 2048$ bitů.

Útočník, který praktikuje Logjam, také využívá MIM. Při navazování komunikace DH protokolu vynutí použití slabší šifry o velikosti 512b, jestliže bude požadovat metodu nazývanou `DHE_EXPORT`. Jedná se o pozůstatek z 90. let, avšak spousta serverů `DHE_EXPORT` stále podporuje a jsou tak náchylné k útoku. Šifra s velikostí 512b jde s vhodným hardwarem prolomit rychle a tak je možné takřka v reálném čase číst zabezpečenou komunikaci.

4.6 Diffie-Hellmanova výměna klíčů s využitím eliptických křivek

Jedná se o variantu klasického Diffie-Hellmanova protokolu s tím rozdílem, že se namísto prvočíselné grupy $\langle \mathbb{Z}_p, \cdot \rangle$ operace provádí nad eliptickou křivkou. Z toho plyne i mírně odlišná činnost. Ve zbytku textu bude použita zkratka názvu – ECDH.

Dodejme, že eliptickou křivku lze vyjádřit parametricky jako (p, a, b, G, n, h), kde je p číslo definující $GF(p)$, a, b jsou koeficienty z rovnice (1), respektive nerovnice (2). Dále G označuje generátor $GF(p)$ a n je takové kladné číslo, pro které platí $nG = O$. Kofaktor $h = \frac{1}{n}|E(GF(p))|$, kde $E(GF(p))$ je podgrupa $GF(p)$. Z Lagrangeovy věty plyne, že kofaktor bude vždy celé číslo.

4.6.1 Průběh ECDH

Předpokládejme, že komunikujícími stranami jsou opět Alice a Bob. Začátek komunikace bude vypadat takto:

1. Strany se dohodnou na parametrech eliptické křivky, tedy (p, a, b, G, n, h).
2. Alice vybere privátní klíč $a \in [1, n-1]$, Bobovi zašle veřejný klíč $A = aG$.
3. Bob zvolí privátní klíč $b \in [1, n-1]$, $B = bG$ předá Alici.
4. Platí, že $s = bA = b(aG) = baG = a(bG) = aB = s'$. Bod křivky, který může dopočítat Alice je tedy shodný s tím, který zjistí Bob a naopak.

4.6.2 Bezpečnost ECDH

Použití eliptických křivek odstraňuje možnost použít Logjam útok na Diffie-Hellmanův protokol, slabiny v případě man-in-the-middle útoku jsou však zachovány.

4.7 RSA

Kryptosystém RSA, pojmenovaný podle svých tvůrců (Rivest, Shamir, Adleman), představuje jeden z nejpoužívanějších kryptosystémů s veřejným klíčem v dnešní době [4]. Díky svému návrhu může být použit nejen k bezpečné komunikaci, ale také k digitálnímu podpisu dat. Základní činnost kryptosystému

je možné rozdělit do tří částí – vytvoření klíčů, šifrování a dešifrování dat. RSA spoléhá na obtížnost rozkladu součinu dvou velkých prvočísel na jeho prvočinitele – tzv. faktorizaci.

4.7.1 Tvorba klíčů

Nejprve se zvolí dvě rozdílná velká prvočísla p, q. Délka p a q dnes bývá v rozmezí 1024 až 4096 bitů. Poté je spočítán jejich součin $n = pq$ a $\phi = \varphi(n) = (p-1)(q-1)$. Náhodně se vybere číslo e, pro které platí $1 < e < \phi$ a zároveň je nesoudělné s ϕ. Pomocí Eukleidova algoritmu je vypočteno nové číslo d takové, že $1 < d < \phi$ a $ed \equiv 1 \pmod{\phi}$. Veřejným klíčem se stává dvojice (n, e), soukromým klíčem pouze číslo d. Číslo e, resp. d se nazývá šifrovací, resp. dešifrovací exponent. Modulem je n a veškeré výpočty probíhají v $\mathbb{Z}_n$.

4.7.2 Šifrování

Opět předpokládejme, že Alice chce zaslat Bobovi tajnou zprávu a již obdržela jeho veřejný klíč (n_B, e_B).

1. Zprávu, nebo její část, reprezentuje jako číslo m z intervalu $\langle 0, n-1 \rangle$.
2. Spočítá $c = m^{e_B} \pmod{n}$, přičemž c představuje zašifrovaný text nebo jeho část.

4.7.3 Dešifrování

Získání původní zprávy je snadné:

1. Bob spočítá $c^d \pmod{n} = (m^{e_B})^{d_B} \pmod{n} = m^{e_B d_B} \pmod{n} = m$.

Příklad 34

Bob chce vygenerovat své klíče a veřejný umístit na dostupném místě. Zvolí $p = 31$, $q = 41$. Poté $n_B = 1271$, $\phi = 1200$ a zvolí $e_B = 59$. Soukromým klíčem je číslo $d_B = 1139$, veřejným dvojice $(1271, 59)$. Alice chce Bobovi zaslat zprávu reprezentovanou číslem 333. Spočítá $c = 333^{59} \pmod{1271} = 771$ a odešle po veřejném kanálu Bobovi. Ten spočítá $c^{d_B} \pmod{1271} \equiv 771^{1139} \pmod{1271} = 333$.

4.7.4 Digitální podpis

V případě digitálního podpisu lze využít obdobných postupů jako při (de)šifrování. Chtějme zaslat zprávu, u které je důležité ověřit důvěryhodnost odesílatele. Spolu se zprávou zašleme tedy i její digitální podpis. Pro zprávu spočteme její kontrolní součet h. Digitálním podpisem je $s = h^d \pmod{n}$, kde d je privátní klíč a n modul. Příjemce může z našeho veřejného klíče získat hash původní zprávy $h = s^e \pmod{n}$, kde e je šifrovací exponent z veřejného klíče, a porovnat s hashem zprávy obdržené. Jestliže se shodují, je zaručena autenticita obdržené

zprávy. Operace jsou prováděny za předpokladu, že $h < n$, jinak je nutné rozdělit hash na části, stejně jako v případě šifrování zprávy.

4.7.5 Bezpečnost

RSA se spoléhá, jak již bylo zmíněno, na obtížnou řešitelnost faktorizace. Pro dnešní počítače není znám žádný efektivní algoritmus, který by otázku faktorizace řešil. Nicméně v blízké budoucnosti, s nástupem kvantových počítačů, bude možné využít tzv. *Shorův algoritmus* [27], který pracuje v polynomickém čase. Roku 1991 zveřejnila společnost RSA Laboratories výzvu nazvanou *RSA Factoring Challenge*, ve které jsou uvedena tzv. RSA čísla – součiny dvou velkých, náhodně zvolených prvočísel, jejichž faktorizace není do *prolomení* známa. V současné době je největším faktorizovaným číslem z výzvy `RSA-768` a lze tedy předpokládat, že `RSA-1024` a vyšší jsou prozatím bezpečná. Známe-li faktorizaci čísla n z veřejného klíče (n, e), je možné dopočítat dešifrovací exponent d a dešifrovat komunikaci. V neposlední řadě se lze setkat i s útoky postranními kanály, jejichž princip byl popsán na konci části 4.3.5.

4.8 ElGamal

Na závěr této kapitoly zmiňme asymetrický kryptosystém ElGamal, který byl zveřejněn Taherem ElGamalem v roce 1985. ElGamal vychází z Diffie-Hellmanovy výměny klíčů a stejně jako RSA může sloužit jak k šifrování komunikace, tak k podepisování dat pomocí soukromého/veřejného klíče. Činnost kryptosystému lze v případě šifrování rozdělit na části generování klíčů, šifrování a dešifrování dat.

4.8.1 Tvorba klíčů

ElGamal se spoléhá na problém diskrétního logaritmu. Proto, stejně jako v případě DH, pracuje s prvky grupy $\langle \mathbb{Z}_p, \cdot \rangle$ a jejím generátorem g. Jako privátní klíč a se zvolí prvek $\mathbb{Z}$ takový, že $1 < a < p-1$. Poté se veřejným klíčem stává trojice $A = (p, g, g^a)$.

4.8.2 Šifrování

Šifrování opět demonstrujme na případu, kdy chce Alice zaslat Bobovi tajně zprávu. Alicin soukromý, resp. veřejný klíč označme a, resp. A. Bobův b, resp. B.

- Alice získá Bobův veřejný klíč $B = (p, g, g^b)$.
- Svou tajnou zprávu m, resp. její část, reprezentuje jako prvek $m' \in \mathbb{Z}_p$.
- Zvolí náhodně celé číslo k takové, že $1 \leq k \leq p-2$.

- Spočítá $\gamma = g^k \pmod{p}$ a $\delta = m' \cdot (g^b)^k \pmod{p}$. Dvojice $s = (\gamma, \delta)$ značí zašifrovaný text.

4.8.3 Dešifrování

V porovnání se symetrickými šiframi DES či AES nelze provést body šifrování pozpátku a získat původní zprávu. Po obdržení zprávy s musí Bob vypočítat:

- $t = \gamma^{p-1-b} \pmod{p} = \gamma^{-b} \pmod{p} = g^{-bk} \pmod{p}$.
- $u = t\delta \pmod{p} = g^{-bk} m' g^{bk} \pmod{p} = m'$. Bob tedy získal Alicinu tajnou zprávu.

4.8.4 Bezpečnost

Kryptosystém ElGamal je závislý na neprolomitelnosti diskrétního logaritmu v rozumném čase, stejně jako Diffie-Hellman. I zde platí, že zvolené prvočíslo p má být dlouhé 2048 bitů a více, požadujeme-li bezpečnou komunikaci.

5 Šifrovaný souborový systém

Jako poslední část této bakalářské práce byla vytvořena aplikace nesoucí název Filecrypter, která umožňuje vytvořit tzv. šifrovaný virtuální souborový systém (dále také pouze VFS). Přístup k souborům umožňuje pouze oprávněným uživatelům (resp. uživatelům se znalostí korektního hesla), pro ostatní jsou data zašifrována. Aplikace je určena pro operační systém Linux, přičemž existuje ve verzi pro příkazový řádek a také s grafickým rozhraním.

5.1 Virtuální souborový systém

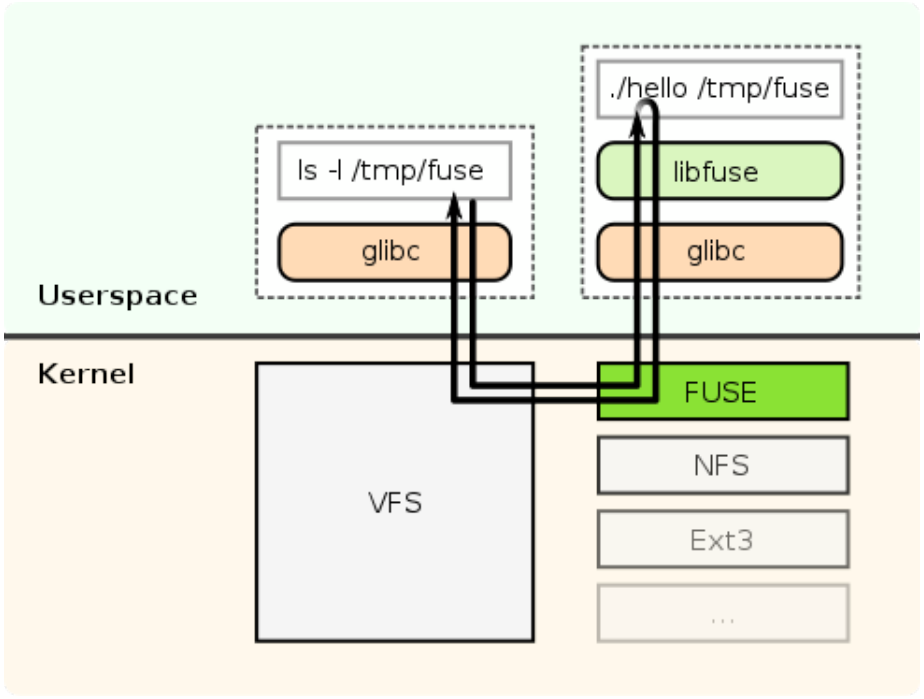
Pro realizaci VFS byl zvolen software zvaný FUSE (Filesystem in USErspace). Skládá se ze dvou složek – jaderného[1] modulu `fuse` a knihovny `libfuse` běžící v uživatelském prostoru. Zmíněná knihovna poskytuje rozhraní pro komunikaci s jaderným modulem. Výhod takového řešení je několik. Zaprvé odpadá nutnost psát nový modul pro vlastní souborový systém a také jej přizpůsobovat změnám v nových verzích linuxového jádra. Další výhodou je možnost připojit (vytvořit) nový souborový systém pro libovolného uživatele – je-li v systému dostupný FUSE, není nutné instalovat podporu libovolného nového souborového systému využívajícího `libfuse` a mít tak administrátorská práva. Lze říci, že FUSE je abstrakcí nad již existujícím souborovým systémem. Když systémová knihovna `glibc` dostane požadavek na čtení souboru/adresáře z FUSE souborového systému, předá jej jadernému modulu `fuse`, ten dále knihovně `libfuse` a teprve potom požadavek obdrží aplikace implementující FUSE souborový systém. Tento princip je výstižněji zobrazen na obrázku 5. Pro jazyk Python komunikaci s `libfuse` poskytuje modul `fusepy` [12]. Samotný FUSE souborový systém poté může být definován jako jeden skript či uživatelská aplikace. Tak tomu je v případě Filecrypteru.

5.2 Uživatelská dokumentace

Jak již bylo zmíněno v úvodu kapitoly, aplikace Filecrypter existuje ve dvou variantách – konzolové a grafické verzi. Systémové požadavky jsou následující:

- operační systém Linux,
- interpret jazyka Python verze 3 a vyšší,
- jaderný modul `FUSE` a knihovna `libfuse`,
- modul `fusepy`,
- kryptografická knihovna `PyCrypto`.

[1] Jádrem je zde myšlený linuxový kernel.



Obrázek 5: Ilustrace zpracování požadavku na výpis adresáře pomocí FUSE. *zdroj: https://commons.wikimedia.org/wiki/File:FUSE_structure.svg*

Grafická verze navíc vyžaduje knihovnu GTK verze 3 a vyšší pro tvorbu uživatelského rozhraní.

Jelikož je aplikace napsaná v interpretovaném jazyce a data se musí navíc při každém požadavku na čtení a zápis (de)šifrovat, je nutné počítat se snížením rychlosti operací čtení a zápisu dat na disk v porovnání s prací s hostitelským souborovým systémem (FS). Uveďme konkrétní příklad. Během testování aplikace probíhal zápis 700MB souboru pomocí Filecrypteru rychlostí 4MB/s, přičemž stejný soubor se na hostitelský FS bez použití Filecrypteru zapisoval rychlostí 75MB/s.

Dále popišme společné předpoklady obou aplikací. Filecrypter vyžaduje k připojení (vytvoření) šifrovaného VFS zadání patřičného hesla a dvou cest – cesty ke zdrojovému adresáři se zašifrovanými daty a cesty k cílovému prázdnému adresáři, do kterého se VFS připojí. U obou adresářů se předpokládá oprávnění čtení a zápisu. Spouštíme-li aplikaci poprvé, je nutné aby byl prázdný i zdrojový adresář. Je tomu tak kvůli absenci ověřovacího mechanismu, který by řekl, zda je zadané heslo správné nebo není. Aplikace se pokusí zdrojový adresář rozšifrovat s libovolným heslem, byť to může vést k nesprávnému výsledku. Volba takového řešení je popsaná v části technické dokumentace. VFS se tedy nijak speciálně neinicializuje, zdrojový adresář se připojí „tak, jak je“. Na základě hodnoty systémové proměnné `$LANG` je aplikace lokalizována do českého nebo anglického jazyka.

5.2.1 Konzolová verze

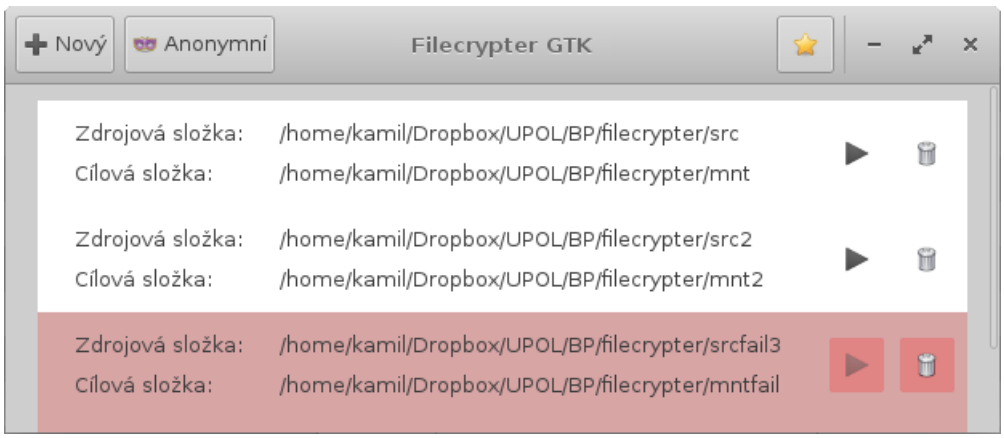
Konzolová verze je univerzálnější a méně náročná na systémové prostředky, jelikož nevyžaduje specifické grafické knihovny. Práce s ní však nemusí být komfortní pro všechny uživatele, i když je velmi snadná.

Spuštění
Po otevření terminálu stačí spustit příkaz
`python3 /cesta/k/filecrypter_console.py zdroj cil`,
kde je druhým argumentem cesta ke zdrojovému adresáři a třetím argumentem cesta k cílovému adresáři. Jinak dojde k výpisu jednoduché nápovědy. Ta se spolu s chybou zobrazí také poté, co zadané cesty neprojdou validací.

Odpojení
Předpokládejme, že máme otevřený terminál. Zadáním následujícího příkazu odpojíme náš VFS.
`fusermount -u /cilovy/adresar`
Program `fusermount` je do systému nainstalován spolu s jaderným modulem `FUSE` a knihovnou `libfuse`. Parametr `-u` říká, že má dojít k odpojení následujícího adresáře. Použijeme-li namísto něj `-uz`, odpojení vynutíme například v případě, že kopírujeme data a program čeká na dokončení operace. Je zřejmé, že potom budou některé soubory nekonzistentní.

5.2.2 Grafická verze

Ke spuštění grafické verze aplikace lze přistupovat dvěma způsoby. Má-li soubor `filecrypter-gtk.py` nastavený příznak spouštění, je možné aplikaci spustit dvojklikem na soubor (v závislosti na použitém správci souborů a jeho nastavení). V opačném případě je nutné daný příznak nastavit a nebo spustit aplikaci z terminálu příkazem níže.
`python3 /cesta/k/filecrypter-gtk.py`



Obrázek 6: Spuštěná aplikace Filecrypter GTK.

Zobrazí se jednoduchá aplikace (obrázek 6), která spravuje „svazky“ – dvojici zdrojová složka a cílová složka. Nejsou-li žádné svazky uloženy, je možné je přidat pomocí tlačítka `Nový`. Tyto svazky jsou reprezentovány graficky seznamem, přičemž validní svazky jsou zobrazeny nejdříve. Validitou se rozumí takový stav svazku, ve kterém jeho složky splňují předpoklady ze začátku této podkapitoly. Není-li svazek validní, není jej ani možné připojit dokud se chyby neopraví. Tento stav je indikován zbarvením příslušného řádku seznamu do červené barvy. Jakmile se nad ním ocitne kurzor, jsou zobrazeny nalezené chyby. Záznamy o svazcích se ukládají do souboru `.filecrypter-entries.xml` v domovském adresáři uživatele.

Připojení svazku proběhne po kliknutí na připojovací tlačítko (šedý trojúhelník) a následném zadání hesla. Pro lepší orientaci v aplikaci je při najetí kurzorem nad většinu prvků okna zobrazena stručná nápověda. Jakmile se svazek připojí, změní se ikona připojovacího tlačítka a opětovným kliknutím na něj se svazek odpojí.

Editace již existujícího záznamu je možná dvojklikem na položku seznamu, přičemž svazek nesmí být připojen. Totéž platí pro odstranění záznamu pomocí posledního tlačítka v příslušném řádku.

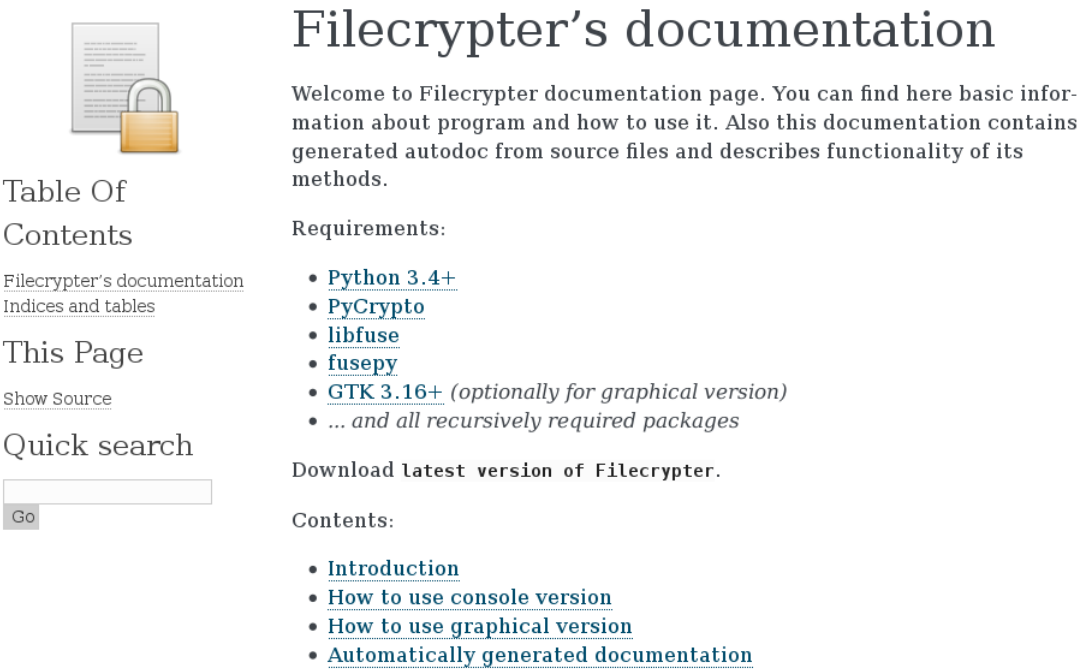
V některých situacích může být žádoucí připojit svazek pouze dočasně a neukládat o něm informace. Pro takové případy slouží tlačítko Anonymní připojení. Po kliknutí na něj se zobrazí takřka totožný dialog jako v případě vytváření nového záznamu, avšak informace o svazku se neukládají do XML souboru a anonymní svazek je v seznamu svazků umístěný jako první.

Aplikace je velmi spartánská a za cíl si především klade poskytnout jednoduché a přehledné rozhraní pro práci se šifrovanými VFS. Graficky se snaží zachovat zásady designu aplikací pro desktopové prostředí Gnome 3.

5.3 Technická dokumentace

Stěžejní částí aplikace je ta, která obstarává tvorbu VFS a šifrování. Proto jsou zde popsány pouze soubory `filecrypterfs.py` a `encryptor.py`. První obsahuje třídu `Filecrypter` dědící z `fuse.Operations`, která reimplementuje souborové operace tak, aby se na hostitelský souborový systém ukládala data zašifrovaná. Ve druhém je definována třída `Encryptor`, která se využívá k šifrování bloků dat. Dokumentace ke zbytku obou aplikací je v anglickém jazyce pro zájemce uvedena ve zdrojových kódech a v adresáři `doc/filecrypter/` na přiloženém CD jako HTML soubor `doc/filecrypter/index.html`. Hlavní stránku webové dokumentace zobrazuje obrázek 7.

Před popisem zmíněných tříd je namístě vysvětlit, proč aplikace připojí zdrojovou složku bez ověření hesla. Jestliže by se do každé zdrojové složky umístil soubor, který by se nejprve aplikace pokusila rozšifrovat a podle výsledných dat rozhodla o korektnosti hesla, významně by tím usnadnila útok hrubou silou. Takový soubor by vykazoval v každém šifrovaném VFS stejnou velikost a umístění. Útočník by tedy věděl, na co se má zaměřit. Současné řešení tedy připojí



Obrázek 7: Dokumentace programu Filecrypter

složku kdykoliv. V závislosti na zadaném hesle může během dešifrování v operaci `readdir` dojít k chybě. Chyba je ošetřena jednoduše tak, že se ignoruje. Proto se adresář může se špatným heslem jevit jako prázdný, nebo může vykazovat méně souborů než ve skutečnosti má obsahovat.

5.3.1 Třída Encryptor

Pro šifrování dat byl zvolen algoritmus AES-256 v CTR (CounTeR) režimu. Režim CTR mění blokovou AES šifru na proudovou, což odstraňuje bezpečnostní nedostatky, díky kterým lze ze šifrovaných dat zjistit četnost výskytu znaků v nešifrovaných datech. Využívá přitom čítače (counteru), jehož počáteční hodnota musí být pro každý blok dat unikátní a jehož velikost koresponduje s velikostí bloku dat (128 bitů). Nejčastěji se využívá čítač inkrementující svou hodnotu přičtením jedničky. V případě třídy `Encryptor` je výchozí hodnota counteru odvozena od odsazení (offsetu) právě šifrovaných dat uvnitř souboru a tím je umožněn náhodný přístup. Třída `Encryptor` využívá kryptografickou sadu nástrojů `PyCrypto`, která je napsaná v jazyce C a poskytuje rychlé vykonávání kryptografických operací v porovnání s tím, když by byly napsané pouze v jazyce Python, tedy interpretovaném jazyku. Třída obsahuje následující funkce:

- `__init(`*`password`*`)__`: Konstruktor třídy. Jako argument vyžaduje heslo. To je zahashováno pomocí funkce `convert_password` a uloženo ve třídě jako klíč k (de)šifrování. Inicializuje `counter` s výchozí hodnotou 0 a objekt `aes` třídy `Crypto.Cipher.AES` s argumenty `password` a

counter.

- convert_password (*password*): Z argumentu password je pomocí algoritmu PBKDF2 odvozeno nové heslo o délce 128 bitů. PBKDF2 obecně slouží ke zpomalení útoků na odhalení hesla, jelikož doba jeho vykonávání je závislá na počtu iterací. V tomto případě je však využit k odvození nového hesla s požadovanou délkou 128 bitů, které je návratovou hodnotou funkce.

- add_padding (*data*): Argument data, jehož délka v bitech není násobkem 128, se na konci vyplní dle PCKS7 standardu. Ten se chová následovně – chybí-li do korektního zarovnání 0xAB bytů, 0xAB-krát se na konec dat přidá byte 0xAB. Návratovou hodnotou jsou vyplněná data.

- remove_padding (*data*): Z argumentu data odebere výplň dle standardu PCKS7 schématu, je-li to možné. Návratovou hodnotou je argument bez výplně.

- encrypt (*plain*, *offset*, *size*): Funkce slouží pro zašifrování dat z argumentu plain. Funce zkontroluje délku plain, eventuálně přidá výplň a uloží do stejné proměnné, a nastaví counter na hodnotu offset. Návratovou hodnotou je výsledek zavolání funkce aes s proměnnou plain.

- decrypt (*enc*, *offset*): Funkce nastaví hodnotu counteru na hodnotu argumentu offset. Data se dešifrují zavoláním aes s argumentem enc a uloží se do proměnné decrypted. Funkce vrátí výsledek volání funkce remove_padding s argumentem decrypted.

Aby názvy adresářů obsahovaly pouze validní symboly, je nutné zašifrovaná data interpretovat v omezené znakové sadě. O to se starají následující dvě funkce:

- encrypt_base64 (*plain*): Vstup plain zašifruje pomocí uvedené funkce encrypt a aplikuje na něj funkci base64.urlsafe_b64encode, jejíž výsledek je návratovou hodnotou. Přičemž base64 je způsob kódování, které vstupní binární data převede na tisknutelné znaky.

- decrypt_base64 (*enc*): Zašifrovaný text enc nejprve převede z kódování base64 zavoláním funkce base64.urlsafe_b64decode do původní podoby a navrátí výsledek volání funkce decrypt spolu s nově dekódovaným textem.

5.3.2 Třída Filecrypter

Aby bylo možné definovat vlastní FUSE souborový systém, je nutné vytvořit třídu reimplementující potřebné souborové operace. Konkrétně v případě užití fusepy je nutné předat třídu dědící z fuse.Operations. V případě Filecrypteru jí je třída nesoucí stejný název v souboru filecrypter-fs.py. Jsou

zde uvedeny pouze ty operace, které mají souvislost s šifrováním bloků dat. Detailní dokumentaci ke všem ostatním metodám lze nalézt na přiloženém CD.

- `__init__(src,password)`: Konstruktor třídy Filecrypter. Je nutné předat cestu se zdrojovým adresářem, aby mohlo dojít k překladu relativní cesty v rámci VFS na adresu v hostitelském FS. Argument `password` slouží pro inicializaci třídy `Encryptor`, která je navázána na proměnnou `aes`.

- `transform_path(relative)`: Pomocná metoda, která obdrženou relativní cestu zašifruje a převede na cestu v hostitelském FS.

- `getattr(fd)`: Metoda navrací atributy cesty, na kterou odkazuje deskriptor `fd`. Odkazuje-li deskriptor na soubor, je nutné opravit atribut `st_size` na konkrétní hodnotu, jelikož je délka šifrovaných dat v hostitelském FS násobkem velikosti 128 bitů. Skutečná velikost se zjistí tak, že se od velikosti zašifrovaného souboru odečte velikost odsazení v posledním bloku zašifrovaných dat.

- `read(path, size, offset, fh)`: Metoda čte data délky `size` ze souboru `path` daného ukazatelem `fh` od indexu obdrženého v argumentu `offset`. Ve zdrojovém kódu 1 je uveden způsob vypořádání se s problémem čtení dat, které leží mimo hranice šifrovaného bloku.

```
1 def read(self, path, size, offset, fh):
2   block_offset = offset - (offset % self.bs) # self.bs = 16
3   block_end = offset + size + self.bs - ((offset + size) % self.bs)
4   data_length = block_end - block_offset
5   os.lseek(fh, block_offset, os.SEEK_SET)
6   blocks = os.read(fh, data_length)
7   decrypted = self.aes.decrypt(blocks, int(block_offset / self.bs))
8   return decrypted[block_offset-offset:size]
```

Zdrojový kód 1: Přečtení zašifrovaných dat

- `write(path, buf, offset, fh)`: Metoda, která obsah bufferu `buf` zapíše do souboru daného cestou `path`, resp. ukazatelem `fh` od pozice `offset`. Opět je nutné se vypořádat s možným nezarovnáním dat na šifrované bloky. Více lze vyčíst ze zdrojového kódu 2.

- `truncate(path, length)`: Metoda zkrátí soubor daný cestou `path` na požadovanou délku `length`. Nastává podobný problém se zarovnáním dat na násobek velikosti šifrovaného bloku. Detaily zobrazuje zdrojový kód 3.

```
1  def write(self, path, buf, offset, fh):
2    buf_size = len(buf)
3    old_size = buf_size
4    file_size = os.fstat(fh).st_size
5    block_buf = b''
6    block_offset = offset - (offset % self.bs)
7    if (offset % self.bs) != 0:
8      os.lseek(fh, block_offset, os.SEEK_SET)
9      enc_block = os.read(fh, self.bs)
10     block_buf += self.aes.decrypt(enc_block, int(block_offset / self.
           bs))[:(offset % self.bs)]
11     buf_size += (offset % self.bs)
12   block_buf += buf
13   if (((buf_size + offset) % self.bs != 0) &
14     ((buf_size + offset) < file_size)):
15     last_block_start = (offset + buf_size) - ((offset + buf_size) %
                                self.bs)
16     os.lseek(fh, last_block_start, os.SEEK_SET)
17     last_block = os.read(fh, self.bs)
18     block_buf += self.aes.decrypt(last_block, int(last_block_start /
           self.bs))[(offset+buf_size) % self.bs:]
19     buf_size += self.bs - ((offset+buf_size) % self.bs)
20   enc_data = self.aes.encrypt(block_buf, buf_size, int(block_offset
         / self.bs))
21   os.lseek(fh, block_offset, os.SEEK_SET)
22   os.write(fh, enc_data)
23   return len(buf)
```

Zdrojový kód 2: Zapsání nových dat

```
1  def truncate(self, path, length):
2    full_path = self.transform_path(path, self.path_encrypt)
3    with open(full_path, 'r+b') as f:
4      if(length > 0):
5        block_pos = length % self.bs
6        f.seek(length - block_pos, os.SEEK_SET)
7        last_block = f.read(self.bs)
8        dec_last_block = self.aes.decrypt(last_block, int((length -
             block_pos) / self.bs))
9        last_block = dec_last_block[:block_pos]
10       enc_last_block = self.aes.encrypt(last_block, len(last_block),
             int((length - block_pos) / self.bs))
11       f.seek(length - block_pos, os.SEEK_SET)
12       f.write(enc_last_block)
13       f.truncate(length + (self.bs - block_pos))
14     else:
15       f.truncate(0)
16     f.close()
```

Zdrojový kód 3: Zkrácení souboru

Závěr

Jedním z přínosů této práce je text pojednávající o souvislosti teorie grup a informatiky. Zájemcům o danou problematiku může posloužit jako prvotní materiál při hledání informací. Po zavedení potřebných pojmů je popsána souvislost teorie grup a grafových automorfismů, formálních jazyků, samoopravných kódů a také užití grup při počítání s velkými čísly. Vybrané pojmy jsou demonstrovány na příkladech. Zejména se však práce zaměřuje na kryptografii. Příslušná část se zabývá kryptosystémy AES, DES, ElGamal či RSA. Je zde popsána činnost kryptosystémů a okrajově jejich bezpečnost. Zmíněny jsou také varianty Diffie-Hellmanovy výměny klíčů, která se využívá například v souvislosti s AES či DES.

Praktickým přínosem je také program `Filecrypter`, který poskytuje uživateli nástroj pro ukládání dat na disk tak, aby k nim bez znalosti hesla neměl přístup nikdo jiný. Je vytvořený v jazyce Python a dostupný pro operační systémy na bázi Linuxu. S použitím softwaru `FUSE` vytváří šifrovaný souborový systém. Konzolovou verzi je možné využít na větším množství linuxových distribucí, jelikož vyžaduje méně závislostí. Grafická verze cílí na uživatele hledající jednoduché uživatelské rozhraní, kteří mají dostupné potřebné grafické knihovny. Další vývoj aplikace by se mohl zabývat zejména optimalizací práce s diskem a jejím zrychlením. To by znamenalo pohodlnější práci s velkými soubory, například multimédii.

Conclusions

One result of this work is a text which discusses the connection between group theory and computer science. It might be helpful for people interested in this problem as essential material during the initial research. After familiarization with fundamental definitions, this work describes the connection between group theory and graph automorphisms, formal languages, self-correcting codes and calculating with big numbers. Selected subjects are demonstrated using examples. However, this work is mainly aimed on cryptography. A designated part describes the cryptosystems like AES, DES, ElGamal or RSA. The variants of Diffie-Hellman key exchange, which are used together with AES or DES, are also mentioned.

Further, application called `Filecrypter` was created. It provides the functionality of storing data on disk securely so that nobody else is able to access the data without entering the password. The application is created in Python language and available for Linux based operating systems. Together with `FUSE` software it creates encrypted virtual file system. The console version can be used on various linux distributions due to less dependencies. The graphical version aims on users looking for simple user interface, who have access to required graphical libraries. Further development of the application could be oriented mainly toward optimization of disk operations and increasing its speed. That would mean more comfortable work with large files, such as multimedia.

A Obsah přiloženého CD

doc/bak/
: Text práce ve formátu PDF a příslušné zdrojové soubory.

doc/filecrypter/
: Dokumentace tříd programu Filecrypter.

filecrypter/
: Zdrojové soubory programu Filecrypter.

readme.txt
: Instrukce pro instalaci a spuštění programu `Filecrypter`, včetně všech požadavků pro jeho zprovoznění.

Literatura

[1] ADRIAN, David; BHARGAVAN, Karthikeyan; DURUMERIC, Zakir aj. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In. *22nd ACM Conference on Computer and Communications Security*. 2015.

[2] ALOUL, Fadi A.; RAMANI, Arathi; MARKOV, Igor L.; SAKALLAH, Karem A. Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 2006, roč. 22, č. 9, s. 1117–1137. Dostupný také z: ⟨http://dx.doi.org/10.1109/TCAD.2003.816218⟩. ISSN 0278-0070.

[3] DRÁPAL, Aleš. Teorie grup – základní aspekty. 2009. Dostupný také z: ⟨http://artax.karlin.mff.cuni.cz/~korbm0am/grupy.pdf⟩.

[4] EMC CORPORATION. *What are some of the more popular techniques in cryptography?* 2016. Dostupný také z: ⟨http://www.emc.me/emc-plus/rsa-labs/standards-initiatives/popular-techniques-in-cryptography.htm⟩.

[5] GOLLOVÁ, Alena. *BCH kódy*. 2015. Dostupný také z: ⟨http://math.feld.cvut.cz/gollova/tik/tik_h7.pdf⟩.

[6] GOLLOVÁ, Alena. *Diskrétní logaritmus a jeho využití v šifrování*. 2013. Dostupný také z: ⟨http://math.feld.cvut.cz/gollova/avt/dodatky-13-4.pdf⟩.

[7] GULLASCH, David; BANGERTER, Endre; KRENN, Stephan. Cache Games – Bringing Access-Based Cache Attacks on AES to Practice. In. *Proceedings of the 2011 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2011, s. 490–505. SP '11. Dostupný také z: ⟨http://dx.doi.org/10.1109/SP.2011.22⟩. ISBN 978-0-7695-4402-1.

[8] HABALA, Petr. *Diskrétní matematika – Počítání modulo*. Dostupný také z: ⟨https://math.feld.cvut.cz/habala/teaching/dma/dmknih07.pdf⟩.

[9] HASSINEN, Marko; MARKOVSKI, Smile. *Secure SMS messaging using Quasigroup encryption and Java SMS API*. Dostupný také z: ⟨http://www.cs.uku.fi/research/publications/reports/A-2003-1/page187.pdf⟩.

[10] HOGBEN, Leslie. *Handbook of linear algebra*. První vyd. Boca Raton: Chapman & Hall/CRC, 2007. ISBN 978-1-58488-510-8.

[11] HOLZER, Markus; KÖNIG, Barbara. On deterministic finite automata and syntactic monoid size. *Theoretical Computer Science*. 2004, roč. 327, č. 3, s. 319–347. Developments in Language Theory. Dostupný také z: ⟨http://www.sciencedirect.com/science/article/pii/S0304397504004840⟩. ISSN 0304-3975.

[12] HONLES, Terence. *fusepy*. 2016. Dostupný také z: ⟨https://github.com/terencehonles/fusepy⟩.

[13] HOŘČÍK, Rostislav. *Regulární jazyky a variety konečných monoidů*. 2012. Dostupné z http://www.cs.cas.cz/pospisil/web_odd/Rusalka2012/horcik.pdf.

[14] JIROUŠEK, Radim. *Principy digitální komunikace*. První vyd. Voznice: Leda, 2006. ISBN 80-7335-084-x.

[15] KATEBI, Hadi; SAKALLAH, Karem A.; MARKOV, Igor L. 2010. Symmetry and Satisfiability: An Update. Theory and Applications of Satisfiability Testing – SAT 2010: 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings, s. 113–127. Dostupný také z: ⟨http://dx.doi.org/10.1007/978-3-642-14186-7_11⟩. ISBN 978-3-642-14186-7.

[16] KOVÁŘ, Petr. *Teorie grafů*. Plzen: Západočeská univerzita v Plzni, 2013.

[17] MARKOVSKI, Smile; GLIGOROSKI, Danilo; STOJČEVSKA, Biljana. *Secure two-way on-line communication by using quasigroup enciphering with almost public key*. 2000. Dostupný také z: ⟨http://www.q2s.ntnu.no/~danilog/publications/Crypto/nsadfin.pdf⟩.

[18] MCKAY, Brendan D.; PIPERNO, Adolfo. Practical graph isomorphism, II. *Journal of Symbolic Computation*. 2014, roč. 60, s. 94–112. Dostupný také z: ⟨http://www.sciencedirect.com/science/article/pii/S0747717113001193⟩. ISSN 0747-7171.

[19] MENEZES, Alfred; OORSCHOT, Paul C. V.; VANSTONE, Scott A. *Handbook of applied cryptography*. První vyd. Boca Raton: CRC Press, 1996. ISBN 0-8493-8523-7.

[20] PÖLSTERL, Sebastian. *The PyGObject Tutorial*. 2016. Dostupný také z: ⟨https://github.com/sebp/PyGObject-Tutorial⟩.

[21] PYTHON SOFTWARE FOUNDATION. *Python 3.4.4 documentation*. 2016. Dostupný také z: ⟨https://docs.python.org/3.4/index.html⟩.

[22] SHCHERBACOV, Victor A. *Quasigroups in cryptology*. 2013. Dostupný také z: ⟨http://arxiv.org/pdf/1007.3572v1.pdf⟩.

[23] STINSON, Douglas R. *Cryptography: theory and practice*. Třetí vyd. Boca Raton: CRC Press, 2006. ISBN 978-1-58488-508-5.

[24] WACHARAMANOTHAM, Chatchavan. *Latin square generator*. 2010. Dostupný také z: ⟨https://hci.rwth-aachen.de/tiki-download_wiki_attachment.php?attId=1075⟩.

[25] WIKIPEDIA. *DES supplementary material — Wikipedia, The Free Encyclopedia*. 2016. Dostupný také z: ⟨https://en.wikipedia.org/wiki/DES_supplementary_material⟩.

[26] WIKIPEDIA. *Hammingův kód — Wikipedia, The Free Encyclopedia*. 2016. Dostupný také z: ⟨https://cs.wikipedia.org/wiki/Hamming%C5%AFv_k%C3%B3d⟩.

[27] WIKIPEDIA. *Shor's algorithm — Wikipedia, The Free Encyclopedia*. 2016. Dostupný také z: ⟨https://en.wikipedia.org/wiki/Shor's_algorithm⟩.

[28] XIAO, Yanghua; WU, Wentao; PEI, Jian; WANG, Wei; HE, Zhenying. Efficiently Indexing Shortest Paths by Exploiting Symmetry in Graphs. In. *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. Saint Petersburg, Russia: ACM, 2009, s. 493–504. EDBT '09. Dostupný také z: ⟨http://doi.acm.org/10.1145/1516360.1516418⟩. ISBN 978-1-60558-422-5.