Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Kódování dat

Michal Kejval

Čestné prohlášení:

Prohlašuji, že svou diplomovou práci "Kódování dat" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2011 _____

Poděkování:

Rád bych touto cestou poděkoval panu Ing. Čestmíru Halbichovi, CSc., vedoucímu diplomové práce, za odborné vedení a rady při zpracování diplomové práce a panu Ing. Františkovi Moravcovi, oponentovi diplomové práce za inspiraci při jejím vzniku.

Kódování dat

Data coding

Souhrn

V této diplomové práci je popsán princip kódování dat. Obsahuje podrobnější vysvětlení vybraných kódovacích algoritmů včetně názorných příkladů. V druhé části je popsána implementace zvoleného řešení do provozu měření elektrické energie v kolejových vozidlech.

Summary

In this graduation theses principle of data coding is described. It contains detailed explication of the selected coding algorithms included synoptical examples. In second part is descrbibed implementation of chosen resolution in to operation of measuring electrical energy for train vehicles.

Klíčová slova:

Kód, kódové slovo, kódovací algoritmus, data, řetězec, symbol, bit, kryptologie, komprese.

Keywords:

Code, code word, scrambling algorithm, data, string, symbol, bit, cryptology, compression.

Obsah

1. Úvod

Diplomová práce navazuje na bakalářskou práci „A/D konverze a digitalizace dat“. V bakalářské práci byl obecně popsán princip digitalizace analogového signálu. V této diplomové práci se budu podrobně zabývat jednou z kapitol analogově-digitální konverze a to kódováním dat.

Lidé chtějí odjakživa získávat a shromažďovat informace v co největším množství. Touha po vědění dala možnost lidstvu postoupit do dnešní informační doby, kdy je běžné vlastnit osobní počítač a pomocí něj neomezeně získávat třídit a ukládat informace pro svou potřebu. Co je však omezené, je kapacita úložišť a kapacita přenosu dat od zdroje k uživateli. Proto vznikaly a stále vznikají různé metody, jak ušetřit místo pro větší množství dat. Také s kapacitou přenosových kanálů se musí nakládat ekonomicky a každá možnost, jak těmito kanály poslat více informací za stejnou časovou jednotku, přijde telekomunikačním společnostem vhod. Díky kompresi tak nyní můžeme například sledovat video ze vzdáleného úložiště on-line, bez sebemenší časové prodlevy.

Nezbytná komprese je vlastně speciální druh kódování dat za účelem zmenšení jejich objemu odstraněním nadbytečné informace tzv. „redundance“. Kompresní algoritmus definuje přesný postup, jakým způsobem je komprese docíleno. Musí obsahovat i odpovídající opačný postup pro dekompresi, který původní informaci zpětně zrekonstruuje. Kompresní algoritmus převádí zdrojové jednotky (symboly a posloupnosti symbolů – tzv. slova a posloupnosti slov) na posloupnosti bitů. Zdrojové jednotky se mohou označovat jako vzory „originály“ a výsledné posloupnosti jako „obrazy.“ [11]

2. Cíl práce a metodika

2.1. Cíl práce

- Zpracovat literární rešerši z oblasti kódování dat.
- Vysvětlit princip kódovacích algoritmů.
- Analyzovat a porovnat různé typy kódovacích algoritmů.
- Popsat technické prostředky pro realizaci kódování dat.
- Na vzorku dat aplikovat vybrané kódovací algoritmy a okomentovat výsledek.
- Implementovat vybrané kódovací algoritmy do projektu měření elektrické energie v kolejových vozidlech.
- Vytvořit souhrn informací ve tvaru použitelném pro edukační účely v oblasti kódování dat.

2.2. Metodika

- Nejprve byly shromážděny veškeré tištěné i elektronické informace, a to jak v češtině tak i angličtině, týkající se kódování dat a následně bylo provedeno vyhodnocení podstaty těchto informací. Soustředil jsem se na obecné principy a vyfiltroval matematicky složité popisy již známých skutečností. Důraz byl kladen především na princip a jednoduchost. Z těchto informací se vytvořil obecný přehled možných řešení a na zvoleném vzorku dat byla tato řešení aplikována. Výsledky byly mezi sebou porovnány a byla vyhodnocena efektivita těchto řešení. V druhé části práce se vybraly vhodné kódovací metody a implementovaly se do projektu měření elektrické energie v kolejových vozidlech.

3. Přehled řešené problematiky

3.1. Data

3.1.1. Možné formy dat.

Nositelem informací, které obohacují naše poznání a které si pořizujeme podle subjektivních kritérií, jsou data. Data neboli údaje, představují smysluplné posloupnosti signálů. Tyto signály se v přírodě vyskytují výhradně v analogové formě v podobě fyzikálních veličin. Takovou běžnou fyzikální veličinou může být například teplota vody. Pro číslicové zpracování a ukládání dat používáme digitální formy dat, které vzniknou analogově-digitální konverzí.

3.1.2. Analogová data

Analogové signály jsou spojité v čase a lze popsat jejich velikosti v daném okamžiku. To znamená, že mohou nabývat nekonečně mnoha hodnot z intervalu, který je pro jejich měření charakteristický. Zároveň nám tuto „aktuální“ hodnotu poskytují v nekonečně krátkém čase, neboli spojitě. Z praxe je známo, že většina signálů v měřící a sdělovací technice je interpretována elektrickým napětím. Například při měření teploty vody, za běžných podmínek, tak ze snímače teploty vystupuje spojitý signál odpovídající teplotě vody mezi 0°C a 100°C. Z důvodu bezpečnosti se toto rozpětí teploty převede na napětí v rozmezí pouze 0V až 10V. Výhodou je bezesporu rychlost odezvy. Nevýhodou pak obtížná komprese, která se v podstatě bez převodu na digitální formu neobejde, a především náchylnost na rušení, tj. na oddělení původního signálu od šumu. [1], [4]

3.1.3. Digitální data

Digitální signály mají následující dvě vlastnosti: jsou to signály s diskrétním časem a jejich množina možných hodnot je konečná. Takovéto signály můžeme reprezentovat posloupnostmi čísel majících konečný počet číslic. Většinou v technické praxi používáme číslice 0 a 1 neboli hodnoty jednoho bitu. Používání právě dvou hodnot je z technického hlediska velmi výhodné, neboť jednu hodnotu můžeme vyjádřit otevřeným tranzistorem a druhou zavřeným tranzistorem. Dá se také říci, že tranzistor pracuje ve spínacím režimu.

Tento režim se vyznačuje značnou nezávislostí na změnách teploty obvodu, na změnách parametrů prvků obvodu a vykazuje dobrou odolnost vůči rušivým vlivům. To je nejspíš hlavní příčina ústupu od jednoduchých a levných analogových systémů a stále větší rozšíření systémů číslicových. Nejen číslicové systémy, ale i číslicové signály mají své výhodné vlastnosti například z hlediska přenosu zpráv. Jednotlivé číslice jsou dobře rozeznatelné a to i po jejich zkreslení a po přidání aditivního šumu. Číslicově vyjádřená data lze snadno utajovat pomocí šifrování nebo zabezpečit proti chybám při přenosu, je možné potlačovat nadbytečnou informaci ve zprávách. [2], [3]

3.2. Kódování dat

3.2.1. Smysl kódování

Abychom mohli pracovat s moderními informačními systémy, musíme veškerá data, která jsou pro nás nositelem žádoucích informací, ukládat v digitální podobě. Nejvhodnější je číslicová forma, která používá binární soustavu, tedy hodnoty 0 a 1. Vychází to z hardwarové konfigurace PC. Moderní velkokapacitní úložiště používají různé principy záznamu těchto dvou hodnot. Například na plotně pevného disku osobního počítače jsou tyto hodnoty reprezentovány kladným a záporným magnetickým polem. Na CD nosiči jsou reprezentovány místy, která buď odráží, nebo neodráží čtecí paprsek. Analogový signál se tedy pro použití v moderním osobním počítači musí konvertovat na digitální a poté jej teprve můžeme zpracovávat a ukládat na datová úložiště. V průběhu digitalizace se po vzorkování a kvantování signálu provádí kódování. Výsledek ale ještě nemusí být zcela uspokojivý, a proto výsledná digitální data, ať už jsou pořízená přímo z digitálního zdroje, nebo nepřímo analogově-digitální konverzí, můžeme dále kódovat.

Smyslem kódování je přiřadit k jednotlivým kombinacím čísel kódová slova, která jsou unikátní a vedou právě k jedné kombinaci zdrojových čísel. Kódová slova jsou reprezentována kombinací binárních znaků v předem zvoleném počtu. Jde v podstatě o pouhé přidělení podle předem definované tabulky. Kódováním dat si můžeme dopomoci k vyšší spolehlivosti při manipulaci s výsledným signálem. Také můžeme, vhodně zvoleným přiřazením binárních kombinací pro jednotlivá kódová slova, docílit zrychlení zpracování a přenosu digitálního signálu. Další výhodou kódování může být utajení informace, kterou data přenášejí, a to tak, že bude čitelná jen pro cílovou skupinu

uživatelů. V neposlední řadě je při kódování možnost snížit objem přenášených, nebo ukládaných dat.

3.3. Samoopravné kódy

U samoopravných kódů nás, jak už název napovídá, zajímá především schopnost opravovat chyby vzniklé při přenosu například rušením apod. Zároveň od kódu požadujeme, aby celkové množství přenesených dat bylo co nejmenší, především kvůli nákladům vynaloženým na přenos dat. Kvalitu takových kódů můžeme ohodnotit pravděpodobností, s jakou jsou schopné opravovat chyby přenášených dat. Předpokládejme, že vysíláme data tvořená posloupností 100 bitů, tedy posloupností symbolů 0 a 1. Příjemce může s jistou pravděpodobností jednotlivé bity přijmout chybné, čili opačné, a tato chyba je mezi jednotlivými bity nezávislá. Pro porovnání samoopravných kódů si zvolíme počet přenášených bitů na 100 bitů a pravděpodobnost, že se každý bit změní, bude 1 procento. Zvolený přenos dat proběhne bez chyby pouze s 36,6 procentní pravděpodobností. Vzorec pro výpočet je: $(1-p)^n = (1-0{,}01)^{100} = 36{,}6\%$ kde p je pravděpodobnost vzniklé chyby na jednom bitu a n je počet přenášených bitů.

3.3.1. Kód s opakováním

Nejjednodušším řešením, jak docílit schopnosti opravovat přenášená data pomocí kódu je opakování jednotlivých vyslaných bitů. Každý bit se vyšle třikrát a vyhodnotí se přijatá většina podle pravdivostní tabulky viz. Tab. č. 1 – Kód s opakováním 3x.

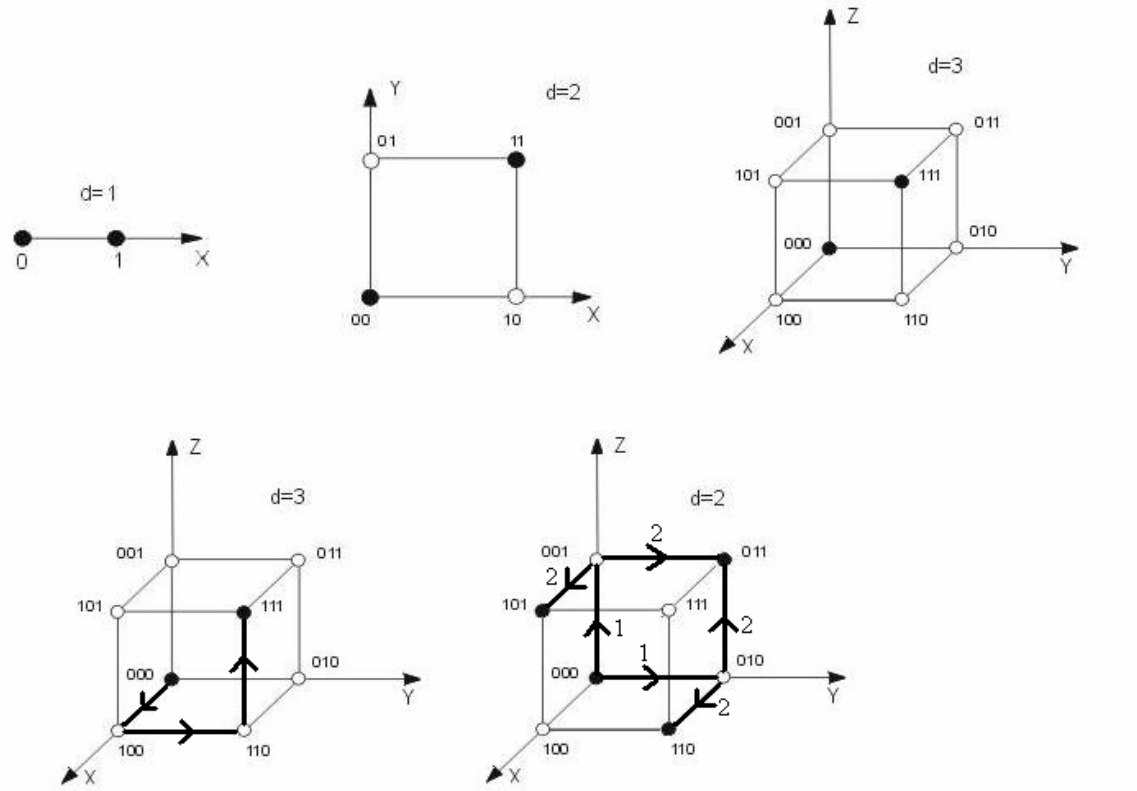
N1 poprvé	N1 podruhé	N1 potřetí	N1 výsledné
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tab. 1 – Kód s opakováním 3x

Vyslaná sekvence N1 poprvé + N1 podruhé + N1 potřetí, tvoří kódové slovo. V takovém případě se pravděpodobnost správného dekódování, čili přijetí správných dat, zvýší na celých 97 procent. Vzorec pro výpočet vychází z předpokladu, že pro přijetí jednoho správného bitu stačí, aby sekvence tří opakování tohoto bitu měla maximálně jednu chybu. Pravděpodobnost žádné nebo jedné chyby v sekvenci opakování jednoho bitu se vypočítá: $(1-p)^3 + 3p(1-p)^2 = 0{,}999702\%$ => 1-p = 0,000298 => nové p. Pravděpodobnost, že se všech 100 bitů přenese bez chyby je: $(1-p)^n = (1-0{,}0000298)^{100} = 97\%$ Tato nesporná výhoda je vykoupena množstvím přenášených dat. Objem přenášených dat se nám zvýší na trojnásobek a tím vzroste i časová náročnost přenosu. Místo původních 100 bitu přeneseme 300 bitů!

3.3.2. Hammingův kód

Dalším způsobem jak docílit samoopravného přenosu je zakódovat data hammingovým kódem. Jmenuje se po jeho autorovi Richardu Hammingovi. Cílem je doplnit kódová slova tak, aby obsahovala paritní bity na takových pozicích, kde budou schopné nejen detekovat chybu, ale zároveň polohu této chyby, čímž tuto chybu opraví. Hammingův kód vychází z minimální kódové vzdálenosti. Minimální kódová vzdálenost nám určuje, v jakém minimálním počtu bitů se od sebe liší veškerá kódová slova. Minimální kódová vzdálenost, také nazývaná Hammingova vzdálenost, je znázorněna v následujícím obrázku viz. Obr. č. 1. - Hammingova vzdálenost.



Obr. 1 – Hammingova vzdálenost

Abychom získali kódová slovo s požadovanou Hammingovou vzdáleností od námi zvoleného počátku, musíme použít právě jeden posun v každém rozměru. Počet rozměrů nám určuje možnou minimální Hammingovu vzdálenost d_{min}.

Schopnost detekce chyb Hammingovým kódem se vypočítá podle vzorce**:**

$Td = d_{min} - 1$, kde Td je počet chyb, které kód detekuje. Schopnost detekce chyby včetně její polohy a tím pádem schopnost korekce chyby Hammingovým kódem se vypočítá podle vzorce **:** $Tc = (d_{min} - 2) / 2$ pro sudá d_{min}, a $Tc = (d_{min} - 1) / 2$ pro lichá d_{min}, kde Tc je počet chyb, které kód opraví. Závislost detekce a korekce na Hammingově vzdálenosti je zobrazená následující tabulce viz. Tab. č. 2 – Závislost detekce a korekce na d_{min}.

Hammingova vzdálenost d_{min}	Detekce chyb Td	Korekce chyb Tc
1	0	0
2	1	0
3	2	1
4	3	1
5	4	2
6	5	2
7	6	3

Tab. 2 – Závislost detekce a korekce na d_{min}

Hammingův kód patří mezi tzv. perfektní kódy, to znamená, že má nejmenší možnou redundanci. Hammingův binární kód s m kontrolními znaky pro m > 2 (m=2,3,4…) má délku 2^m-1. Z toho tedy dostáváme binární (2^m-1, 2^m- m – 1) kód. Nejčastěji tedy: (3,1) - kód, (7,4) – kód, (15,11) – kód, atd… Pro korekci jedné chyby v přenosu se používá Hammingův kód (7,4) který má Hammingovu vzdálenost rovnou 3 a je schopen opravit jednu chybu. Takový kód rozdělí data na čtyřbitové bloky. Každý čtyřbitový blok zakóduje na sedmibitové kódové slovo, které se vysílá příjemci. U symetrické varianty probíhá kódování vynásobením čtyřbitového bloku s tzv. Hammingovou generující maticí (7,4) viz Tab. č. 3. – Hammingova generující matice (7,4). Ta nám generuje pravdivostní tabulku, viz Tab č. 4 – Hammingova pravdivostní tabulka – symetrický kód. Dekódování pak probíhá ve dvou fázích. V první se vynásobením přijatého slova s opravnou Hammingovou maticí viz Tab. Č. 5 – Opravná Hammingova matice – symetrický kód, a následným dělením modulo 2 získá tříbitový syndrom, který nám určuje řádek, z Hammingovy opravné matice, na kterém vznikla chyba. Pořadí řádku je pak rovno pořadí chybného bitu z leva. Nultý řádek je brán jako přenos bez chyby. V druhé fázi se pomocí pravdivostní tabulky viz. Tab. č. 4 - Hammingova pravdivostní tabulka – symetrický kód, získají původní nekódovaná data.

1	0	0	0	1	1	1
0	1	0	0	1	1	0
0	0	1	0	0	1	1
0	0	0	1	1	0	1

Tab. 3 – Hammingova generující matice (7,4)

Data					Kódové slovo						
0	0	0	0		0	0	0	0	0	0	0
0	0	0	1		0	0	0	1	1	0	1
0	0	1	0		0	0	1	0	0	1	1
0	0	1	1		0	0	1	1	1	1	0
0	1	0	0		0	1	0	0	1	1	0
0	1	0	1		0	1	0	1	0	1	1
0	1	1	0		0	1	1	0	1	0	1
0	1	1	1	=>	0	1	1	1	0	0	0
1	0	0	0		1	0	0	0	1	1	1
1	0	0	1		1	0	0	1	0	1	0
1	0	1	0		1	0	1	0	1	0	0
1	0	1	1		1	0	1	1	0	0	1
1	1	0	0		1	1	0	0	0	0	1
1	1	0	1		1	1	0	1	1	0	0
1	1	1	0		1	1	1	0	0	1	0
1	1	1	1		1	1	1	1	1	1	1

Tab. 4 – Hammingova pravdivostní tabulka – symetrický kód

0	0	0	Bez chyby
1	1	1	Chyba 1.bit
1	1	0	Chyba 2.bit
0	1	1	Chyba 3.bit
1	0	1	Chyba 4.bit
1	0	0	Chyba 5.bit
0	1	0	Chyba 6.bit
0	0	1	Chyba 7.bit

Tab. 5 – Hammingova opravná matice – symetrický kód

Postup kódování a dekódování například bloku (1,0,1,0) je následující:

1) Blok čtyř bitů se vynásobí s Hammingovou generující maticí a tím nám vznikne sedmimístné kódové slovo.

$$1010 \times \begin{matrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{matrix} = 1\,0\,1\,0\,1\,0\,0$$

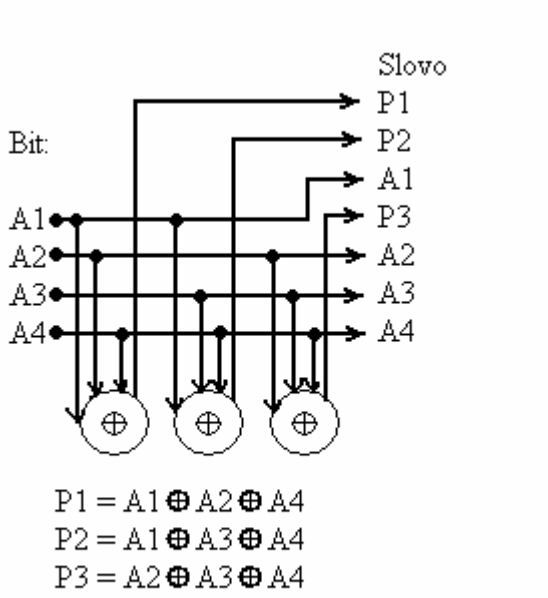
2) Příjemci se vyšle kódové slovo: 1,0,1,0,1,0,0. Pokud příjemce přijme kódové slovo s jednou chybou například v pátém bitu, přijme: 1,0,1,0,0,0,0. Díky použití Hammingova kódu (7,4) je schopen tuto chybu opravit následujícím způsobem:

3) Přijaté kódové slovo se vynásobí s Hammingovou opravnou maticí viz. Tab. č. 4 - Hammingova pravdivostní tabulka – symetrický kód, a dílčí výsledky se vydělí modulo 2, tj zapíše se zbytek po dělení dvěma. Tím vznikne tříbitový příznak, který nám určuje kombinaci z Hammingovi opravné matice viz. Tab. č. 5 - Hammingova opravná matice, na které vznikla chyba. Matice je sestavena tak, aby pořadí řádku určovalo pořadové číslo chybného bitu z leva.

$$1010000 \times \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} = \begin{matrix} \text{Výsledek} = 1\ 2\ 2 \\ \text{Modulo 2} = \mathbf{1\ 0\ 0 \Rightarrow 5.\ bit\ opravit} \end{matrix}$$

4) Po opravení přijaté sekvence 1,0,1,0,0,0,0 na 1,0,1,0,1,0,0 provedeme dekódování podle pravdivostní tabulky viz Tab. č. 4 -Hammingova pravdivostní tabulka – symetrický kód. Tím získáme původní blok čtiř bitů 1, 0, 1, 0.

Princip Hammingova kódu je ještě lépe vidět na jeho nesymetrické variantě, kde se využívá paritních bitů. Opět použijeme Hammingův kód (7,4). Na následujícím obrázku je vidět generátor kódových slov viz. Obr. č. 2. – Generátor Hammingova (7,4) kódu.



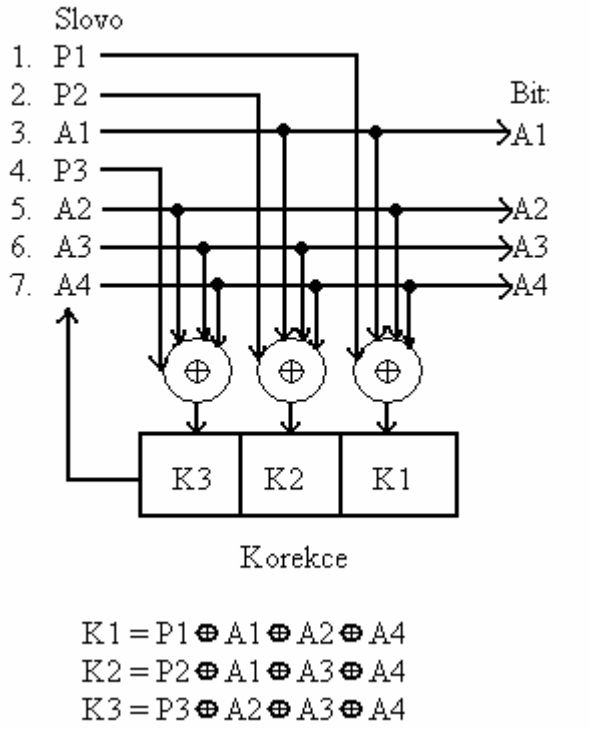
Obr. 2 – Generátor Hammingova (7,4) kódu

Takový generátor nám generuje Hammingův nesymetrický kód (7,4) s pravdivostní tabulkou viz. Tab. č. 6 - Hammingova pravdivostní tabulka – nesymetrický kód.

Data					Kódové slovo						
A1	A2	A3	A4		P1	P2	A1	P3	A2	A3	A4
0	0	0	0		0	0	0	0	0	0	0
0	0	0	1		1	1	0	1	0	0	1
0	0	1	0		0	1	0	1	0	1	0
0	0	1	1		1	0	0	0	0	1	1
0	1	0	0		1	0	0	1	1	0	0
0	1	0	1		0	1	0	0	1	0	1
0	1	1	0		1	1	0	0	1	1	0
0	1	1	1	=>	0	0	0	1	1	1	1
1	0	0	0		1	1	1	0	0	0	0
1	0	0	1		0	0	1	1	0	0	1
1	0	1	0		1	0	1	1	0	1	0
1	0	1	1		0	1	1	0	0	1	1
1	1	0	0		0	1	1	1	1	0	0
1	1	0	1		1	0	1	0	1	0	1
1	1	1	0		0	0	1	0	1	1	0
1	1	1	1		1	1	1	1	1	1	1

Tab. 6 – Hammingova pravdivostní tabulka – nesymetrický kód

K dekódování Hammingova (7,4) kódu slouží dekodér, který je na následujícím obrázku viz. Obr. č. 3. – Dekodér Hammingova (7,4) kódu, jehož součástí je i tříbitový syndrom tzv. korekce.



Obr. 3 – Dekodér Hammingova (7,4) kódu

Korekce se dále použije k opravě příslušného bitu z množiny A1, A2, A3, A4 podle korekční tabulky viz. Tab. Č. 7 –Hammingova opravná matice – nesymetrický kód.

0	0	0	Bez chyby
0	0	1	Chyba 1.bit
0	1	0	Chyba 2.bit
0	1	1	Chyba 3.bit
1	0	0	Chyba 4.bit
1	0	1	Chyba 5.bit
1	1	0	Chyba 6.bit
1	1	1	Chyba 7.bit

Tab. 7 – Hammingova opravná matice – nesymetrický kód

Postup kódování a dekódování stejného bloku (1,0,1,0) je následující:

1) Blok čtyř bitů se zakóduje pomocí generátoru kódu viz. Obr. č. 2. – Generátor Hammingova (7,4) kódu a tím nám vznikne sedmimístné kódové slovo.

A1	A2	A3	A4		P1	P2	A1	P3	A2	A3	A4
1	0	1	0	=>	1	0	1	1	0	1	0

$P1 = 1 \oplus 0 \oplus 0 = 1$
$P2 = 1 \oplus 1 \oplus 0 = 0$
$P3 = 0 \oplus 1 \oplus 0 = 1$

2) Příjemci se vyšle kódové slovo: 1,0,1,1,0,1,0. Pokud příjemce přijme kódové slovo s jednou chybou například v pátém bitu, přijme: 1,0,1,1,1,1,0. Díky použití Hammingova kódu (7,4) je schopen tuto chybu opravit následujícím způsobem:
3) Z přijatého kódového slova se pomocí Hammingova dekodéru viz Obr. č. 3. – Dekodér Hammingova (7,4) kódu, vyhodnotí potřeba opravovat přijaté slovo, v našem případě je hodnota korekce rovna 1, 0, 1 a z Hammingovi korekční tabulky viz. Tab. Č. 6 – Opravná Hammingova matice – nesymetrický kód, odpovídá potřebě opravit pátý bit.

P1	P2	A1	P3	A2	A3	A4		A1	A2	A3	A4
1	0	1	1	1	1	0	=>	1	0	1	0

$K1 = 1 \oplus 1 \oplus 1 \oplus 0 = 1$
$K2 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$ => **opravit 5. bit**
$K3 = 1 \oplus 1 \oplus 1 \oplus 0 = 1$

4) Po opravě příslušného bitu nám již syndrom 0, 0, 0 ukazuje na správně přijaté slovo 1,0,1,1,0,1,0 a to se může dekódovat pomocí Hammingovi pravdivostní tabulky viz. Tab. Č. 6 – Hammingova pravdivostní tabulka – nesymetrický kód. Tím získáváme původní data 1, 0, 1, 0.

Obě tyto metody jsou schopné opravovat pouze jednu chybu. Pokud se při přenosu vyskytne chyba ve dvou nebo více různých bitech, pak jsou pro nás data nenávratně ztracena.

Pro porovnání Hammingova kódu (7,4) s opakovacím kódem použijeme stejný případ, čili počet přenášených bitů bude 100 bitů a pravděpodobnost, že dojde k chybě přenášeného bitu, bude 1 procento.

Aby došlo ke shodě příjemce a odesilatele, je potřeba, aby při přenosu kódového slova o sedmi bitech nedošlo k žádné chybě, nebo aby v tomto přenosu došlo pouze k jedné chybě. Pravděpodobnost se vypočítá:

$$(1-p)^7 + 7p(1-p)^6 = (1-p)^6(1+6p)$$

A protože kódových slov je celkem n/4 čili 25, příjemce dekóduje celou zprávu s pravděpodobností 95 %, která se vypočítá:

$$\left((1-p)^6(1+6p)\right)^{25} = 95\%$$

Takto chráněný přenos je vykoupen větším objemem dat. Místo n bitů musíme přenést $7n/4 = 175$ bitů. [9]

3.3.3. Golayův kód

Dalším způsobem jak docílit samoopravného přenosu je zakódovat data Golayovým kódem. Tento kód objevil švýcarský matematik Marcel J.E. Golay v roce 1949. Golayův kód je poslední kód ze skupiny perfektních kódů. Perfektní kódy mají kódová slova v prostoru rozmístněna velice efektivně. Pokud si představíme, že slova s povolenou chybou tvoří kolem kódových slov koule o poloměru jedna, tak v případě Hammingova kódu tyto koule vyplňují celý prostor. V případě Golayova (23,12) kódu je také vyplněn celý prostor, ale od Hammingova kódu se liší velikostí tohoto prostoru. Golayův kód má minimální kódovou vzdálenost rovnou sedmi a je tedy schopen opravovat až tři chyby. Vzhledem k vysokému počtu kombinací není účelné prezentovat postup kódování a dekódování, neboť je velice podobný Hammingovu kódu a vychází ze stejných principů.[12]

Pokud bychom použili Golayův kód na námi zvolený případ přenosu 100 bitů zjistíme, že pravděpodobnost bezchybného přijetí zprávy je 99,9 %.

Pro výpočet přenosu 23 bitů s maximální chybou na třech bitech se používá součet pravděpodobností pro přenos bez chyby, s jednou chybou, s dvěma chybami a se třemi chybami. Podle binomického rozdělení je vzorec pro výpočet:

$$(1-p)^{23} + 23p(1-p)^{22} + 253p^2(1-p)^{21} + 1771p^3(1-p)^{20}$$

A protože kódových slov je celkem: n / 12 čili 9, příjemce dekóduje celou zprávu s pravděpodobností 99,9 %, která se vypočítá:

$$\left((1-p)^{23} + 23p(1-p)^{22} + 253p^2(1-p)^{21} + 1771p^3(1-p)^{20}\right)^9 = 0{,}9993$$

Takto chráněný přenos je vykoupen větším objemem dat. Místo 100 bitů musíme přenést 9 kódových slov po 23 bitech => $9 * 23 = 207$ bitů. [9]

3.3.4. Porovnání samoopravných kódů

Všechny typy samoopravných perfektních kódů jsou porovnány v tabulce, viz Tab. č. 8 – Porovnání perfektních kódů. Výpočty hodnot jsou uvedeny v předchozích kapitolách. Pro porovnání samoopravných kódů byl zvolen počet přenášených bitů n = 100 bitů a pravděpodobnost, že se každý bit změní, je p = 1 procento.

Použitý typ kódů	Počet přenesených bitů [n]	Pravděpodobnost správného přenosu [%]
Bez kódování	100	36,6
S opakováním	300	97
Hammingův	175	95
Golayův	207	99,9

Tab. 8 – Porovnání perfektních kódů

Všechny tři typy kódů výrazně zvýší pravděpodobnost bezchybného přenosu dat. Při volbě typu kódu je potřeba určit poměr mezi bezchybností přenosu a náročností přenosu. Golayův kód je v obou sledovaných parametrech lepší než kód s opakováním. Proto by v tomto případě nemělo smysl kód s opakováním použít. Pro velké objemy dat kde nezáleží na rychlosti přenosu ale primárně na bezchybnosti přenosu je jasným vítězem Golayův kód. Dokáže opravit velké množství chyb a co do počtu přenášených bitů je oproti Hammingovu kódu jen o 32 bitů náročnější. Pokud by byl primární cíl co nejvíce snížit počet přenášených bitů při zachování vysoké úrovně zabezpečení bezchybnosti, zvolil by se Hammingův kód, který má méně než dvojnásobnou datovou náročnost a přesto je schopen zvýšit pravděpodobnost správného přenosu dat více než dvojnásobně.

Pro správný výběr kódu v konkrétních situacích je nutné plně pochopit problematiku daného přenosu a zajistit správné hodnoty pravděpodobnosti vzniku chyby a

počet přenášených bitů. Jiné hodnoty budou v páteřní optické síti mezi kontinenty a jiné hodnoty budou v domácí wifi síti.

3.4. Kompresní kódy

Komprese je speciální druh kódování dat, kde požadujeme co nejmenší výsledný objem dat. Kompresní algoritmy odstraňují z dat nadbytečnou informaci. Taková informace se nazývá redundantní a bývá charakterizována především: opakováním symbolů, opakováním slov, kontextovou závislostí, neefektivní přímou reprezentací, nejednotným rozložením symbolů. Kompresní algoritmy musí přesně definovat postup jak pro kompresi, tak i pro dekompresi, aby bylo možné z komprimovaných obrazů získat zpět původní originály.

Kompresní algoritmy se dělí na ztrátové a bezeztrátové. U bezeztrátových algoritmů je výsledný obraz identický s originálem a po procesu komprese a dekomprese se výsledek nijak neliší od originálu. U ztrátové komprese, která je co do objemu dat mnohem efektivnější, se z obrazu po kompresi už nikdy nezíská identický originál. Ztrátová komprese využívá nedokonalosti lidských smyslů. Ztrátová komprese odstraňuje takové informace, které jsou lidskými smysly nepostřehnutelné, a tudíž jsou pro člověka zbytečné. Obtížné je ovšem stanovit, do jaké míry chceme onu nadbytečnou informaci odstranit a tím ušetřit data, neboť se může stát, že právě tuto odstraněnou informaci budeme v budoucnu potřebovat a z komprimovaného obrazu už ji nikdy nezískáme. Příkladem může být záznam zvuku. Ztrátovou kompresí odstraníme ze zaznamenaného zvukového signálu takové frekvence, které lidské ucho neslyší. Pro poslech bude dekomprimovaný signál naprosto dostačující, ale pokud bychom chtěli odpověď na otázku, zda kolem snímače neproletěl netopýr, ani podrobnou analýzou dekomprimovaného obrazu to nezjistíme, neboť signál který netopýr vysílá, byl komprimací odstraněn. Je tedy potřeba rozhodnout o tom, které informace jsou podstatné a které už nikdy nebudeme potřebovat.

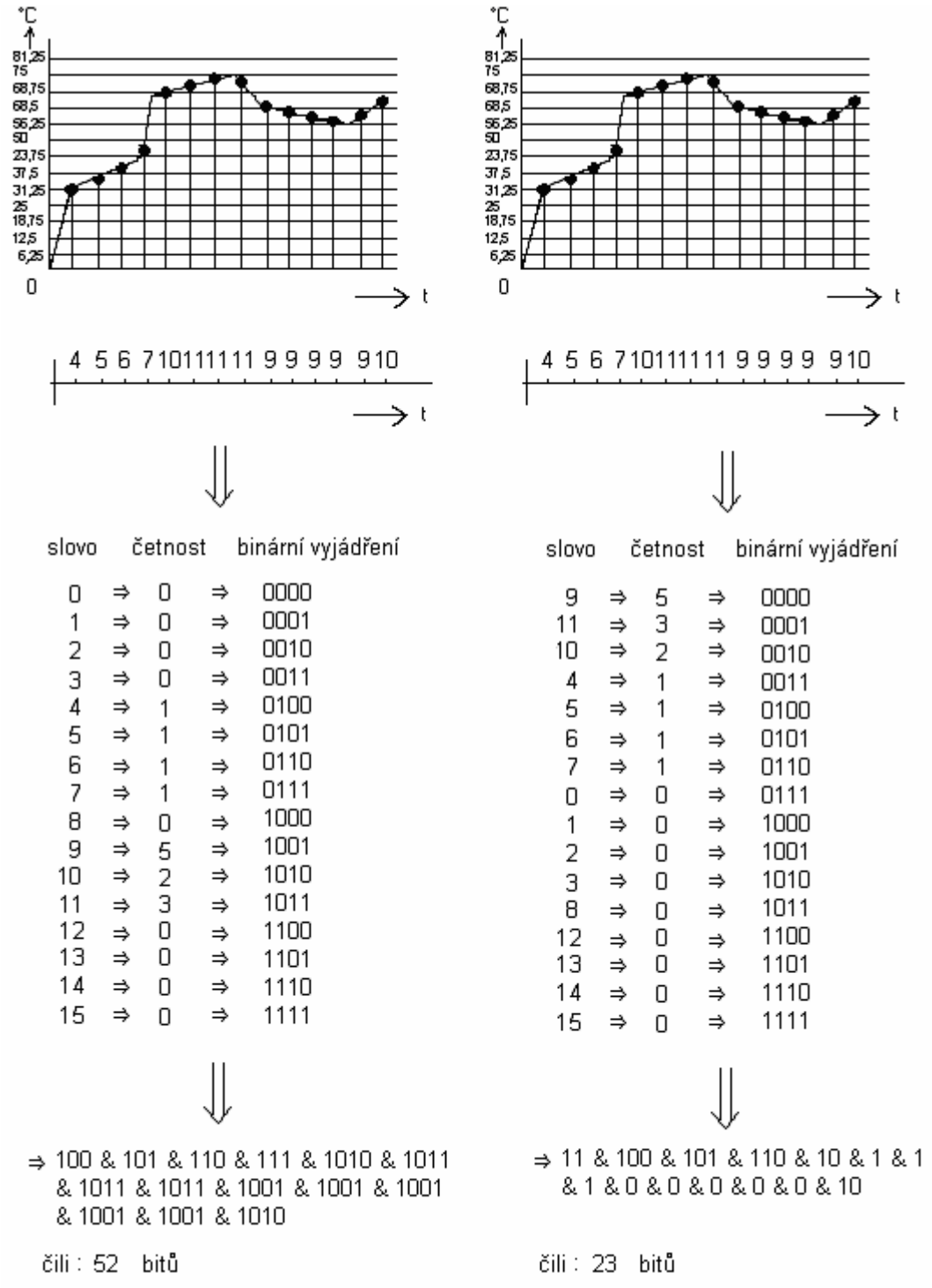
3.4.1. Morseův princip

S první myšlenkou jak zkomprimovat přenášená data, v tehdejší optické telegrafní soustavě, přišel na základě kongresem vypsané soutěže pan Samuel F. B. Morse. Nejenže zrealizoval elektromagnetický telegraf, ale také využil pravděpodobnostní rozdělení zdrojových znaků a docílil tak snížení nároků na přenos co se objemu přenášených dat týče. Morseův kód je technicky realizován přiřazením elektrických impulsů dvou délek,

které jsou po převedení na optický nebo akustický signál dodnes známé jako „tečka“ a „čárka“, jednotlivým znakům zdrojové abecedy tedy písmenovým symbolům. Morseův kód ovšem není binární, jak by se mohlo na první pohled zdát, ale ternární. Třetím znakem je mezera, která je stejně dlouhá jako „tečka“. „Čárka“ má trojnásobnou délku, z čehož vyplývá, že Morseův kód je nerovnoměrný. Další vlastností Morseova kódů je, že není prefixový. Prefixový kód je takový kód, který má tu vlastnost, že žádný symbol jeho kódové abecedy není předponou (prefixem, začátkem) jiného (delšího) symbolu abecedy. Pokud je nějaký kód prefixový, je možné řetězce symbolů tohoto kódu jednoznačně dekódovat, aniž by mezi jednotlivými symboly musely být oddělovače. Mezi prefixové kódy patří např. Huffmanovy kódy, prefixový kód tvoří také mezinárodní směrová čísla.

Morseův kód přiřazuje znakům s velkou pravděpodobností výskytu taková kódová slova, která jsou kratší, čili rychleji přenositelná a naopak pro znaky s malou pravděpodobností výskytu taková slova, která jsou delší a jejich přenos je náročnější. Podmínkou je znát pravděpodobnost výskytu znaků v množině slov nad zdrojovou abecedou.

Jako příklad si uvedeme převod analogového signálu, z čidla pro měření teploty vody, do digitální podoby. Na obrázku viz. Obr. č. 4 – Porovnání mezi rovnoměrným a nerovnoměrným kódováním, je použit nerovnoměrný kód, který nám může snížit počet bitů, potřebných k popisu daného signálu o více než polovinu. Jednotlivá kódová slova si seřadíme sestupně podle četnosti výskytu. Binární kombinace zvoleného počtu bitů se také seřadí sestupně a to podle náročnosti interpretace. Z praxe je známo, že kombinace „0000“ nebo „0001” jsou 4x rychleji zpracovatelné než kombinace „1000“ nebo „1011”, neboť v prvním případě nám postačí zpracovat pouze první bit „0“ nebo „1“ čili bit s nejmenší váhou. V dalším kroku se přiřadí tyto nejrychleji zpracovatelné binární kombinace, seřazené vzestupně od nejkratší po nejdelší, kódovým slovům, seřazeným vzestupně podle četnosti výskytu od nejpravděpodobnějšího po kódové slovo s nejmenším počtem výskytů. A to tak, že právě jednomu kódovému slovu s nejčastějším výskytem přiřadíme právě jednu binární kombinaci s nejkratší dobou zpracování. Tímto postupem docílíme nejefektivnější využití nerovnoměrného kódování, neboť tak zajistíme, že při zpracování analogového signálu má převodník na výstupu nejčastěji právě nejkratší binární kombinace. Je zde potřeba dobře zvážit, jakým způsobem vybrat část signálu z důvodu reprezentativnosti vzorku.



Obr. 4 – Porovnání mezi rovnoměrným a nerovnoměrným kódováním

3.4.2. Shannon-Fanovo kódování

Shannon-Fanovo kódování je jedna z technik, kterou lze získat prefixový kód. Je založena na principu seznamu symbolů a jejich výskytu stejně jako u Morseova kódu. Oproti Morseovu kódu tvoří prefixový kód, je tedy binární a nemusí používat oddělovače slov. Postup tvorby kódu je následující:

1) Pro danou zprávu se vytvoří seznam symbolů a zjistí počet jejich výskytů (nebo pravděpodobnost).
2) Seznam symbolů se seřadí podle počtu výskytů od nejčastějšího po nejméně častý.
3) Dále se rozdělí seznam na dvě poloviny tak, aby rozdíl součtu sledovaných hodnot byl v obou částech co nejmenší.
4) Ke kódům symbolů v levé polovině se připojí 0, ke kódům symbolů v pravé polovině se připojí 1.
5) Rekurzivně se opakuje od kroku 3 na levou i pravou polovinu seznamu až do rozdělení na jednotlivé symboly.

Jako příklad si uvedeme zprávu: (a,b,f,d,c,f,d,e,b,f,c,e,a,d,b,a,f,c,f,c,d). Prvním krokem je seřadit symboly podle jejich četnosti výskytu viz Tab. Č. 9 – Četnost symbolů ve zprávě.

Symbol	Četnost
F	5
D	4
C	4
A	3
B	3
E	2

Tab. 9 – Četnost symbolů ve zprávě

Dále se seznam (F,D,C,A,B,E) rozdělí na dvě poloviny se stejným součtem četností, nebo s co nejmenším rozdílem součtu četností.

(F,D) – součet četností: 5 + 4 = 9

(C,A,B,E) – součet četností: 4 + 3 + 3 + 2 = 12

Tato operace se pak opakuje s novými seznamy až do rozdělení na jednotlivé symboly. Nové seznamy jsou (F,D) a (C,A,B,E).

(F,D) – součet četností: 5 + 4 = 9

(F) – součet četností: 5

(D) – součet četností: 4

(C,A,B,E) – součet četností: 4 + 3 + 3 + 2 = 12

(C,A) – součet četností: 4 + 3 = 7

(B,E) – součet četností: 3 + 2 = 5

V posledním kroku seznamy (F) a (D) již rozložit nejdou, zbývá zpracovat seznamy (C,A) a (B,E). Opět se oba rozdělí na poloviny.

(F,D) – součet četností: 5 + 4 = 9

(F) – součet četností: 5

(D) – součet četností: 4

(C,A,B,E) – součet četností: 4 + 3 + 3 + 2 = 12

(C,A) – součet četností: 4 + 3 = 7

(C) – součet četností: 4

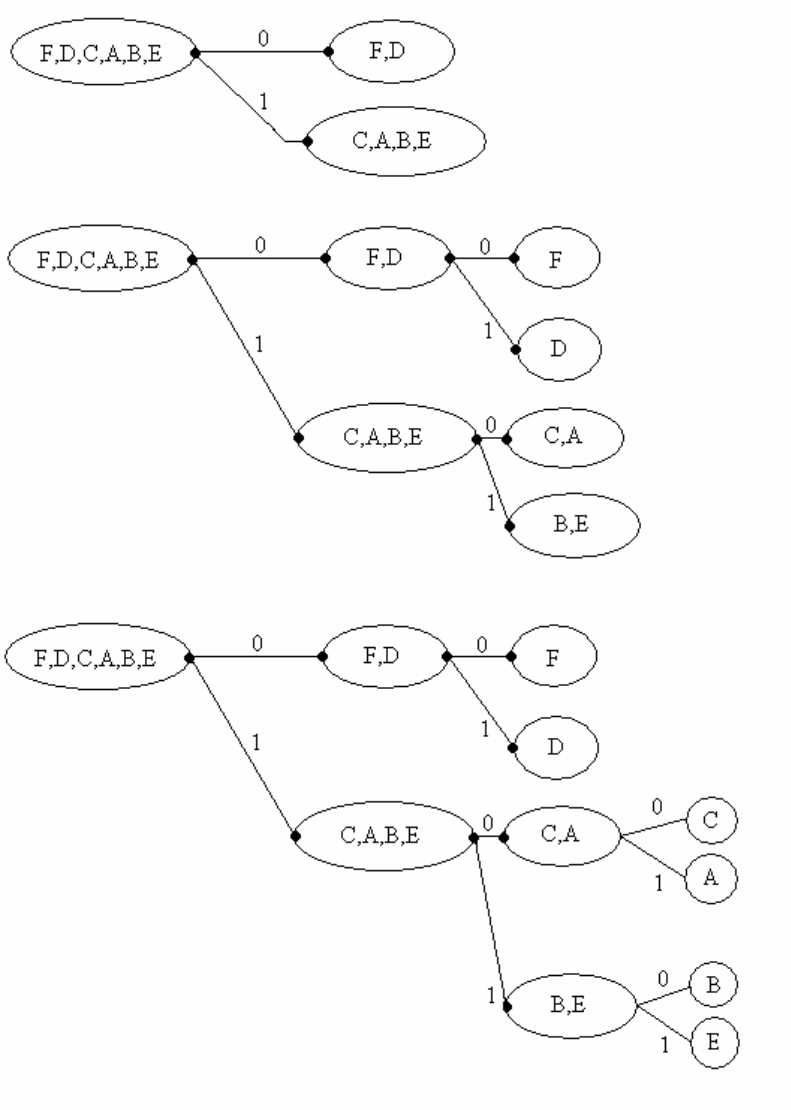
(A) – součet četností: 3

(B,E) – součet četností: 3 + 2 = 5

(B) – součet četností: 3

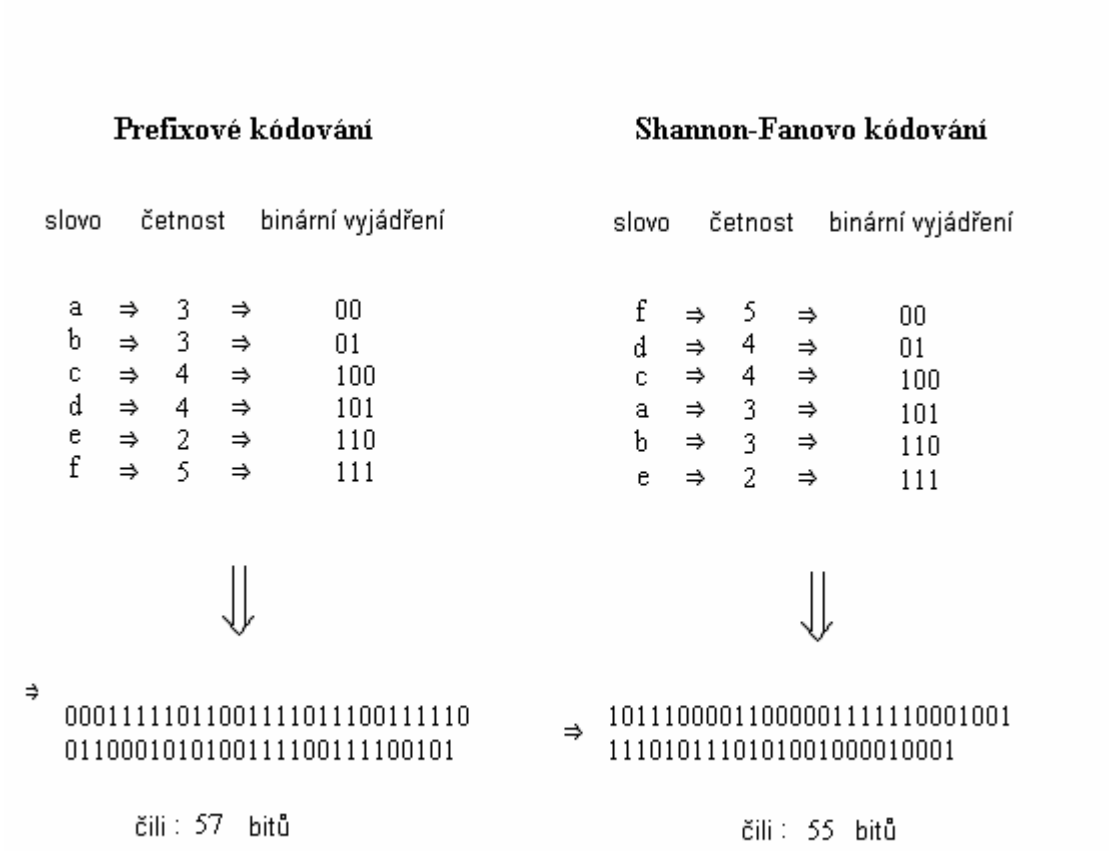
(E) – součet četností: 2

Graficky znázorněný postup je na následujícím obrázku viz. Obr. č. 5. – Postup Shannon-Fanova kódování.



Obr. 5 – Postup Shannon-Fanova kódování

Výsledkem takového postupu je pravdivostní tabulka viz Tab. Č. 10 – Shannon-Fanova pravdivostní tabulka, pomocí které se následně zakóduje požadovaná zpráva. Na následujícím obrázku viz. Obr. č. 6. – Porovnání Shannon-Fanova kódování s prefixovým kódem, je vidět porovnání obyčejného prefixového kódu a Shannon-Fanova kódu. I u takto krátké zprávy, ve které nejsou velké rozdíly mezi četnostmi znaků, nám Shannon-Fanovo kódování ušetřilo 2 bity.



Obr. 6 – Porovnání Shannon-Fanova kódování s prefixovým kódem

Data		Kódové slovo		
a		1	0	1
b		1	1	0
c	=>	1	0	0
d		0	1	-
e		1	1	1
f		0	0	-

Tab. 10 - Shannon-Fanova pravdivostní tabulka

Obě tyto metody patří mezi statické metody komprese dat. Využívají pravděpodobnosti výskytu jednotlivých symbolů. Dále si ukážeme další zástupce kompresních algorytmů, které využívají slovník. Slovníkové metody jako svůj model používají slovník. Slovník je speciální dynamická datová struktura, která se postupně vytváří během čtení vstupu.

3.4.3. Algoritmus LZ78

Kompresní metoda LZ78 je slovníková, jednoprůchodová, adaptivní a symetrická. Její autoři Lempel a Ziv ji uvedli v roce 1978. Jejím nedostatkem je veliká paměťová náročnost. Tento nedostatek se řeší zmrazením slovníku, nebo jeho reorganizací.

Slovník si můžeme představit jako tabulku, kde je v každém řádku uveden řetězec znaků neboli fráze a jemu odpovídající kódové slovo tzv. index fráze. Slovník musí splňovat podmínku: Je-li ve slovníku uložen nějaký řetězec znaků, pak jsou ve slovníku uloženy všechny jeho prefixy. Pokud je slovník již příliš veliký, můžeme jej zmrazit, tj. přestat do slovníku zapisovat další řetězce s výjimkou nových znaků vstupní abecedy.

V každé iteraci algoritmu se hledá nejdelší fráze ve slovníku, která odpovídá dosud nezpracované části vstupního řetězce. Na výstup je poté vloženo kódové slovo, které se skládá z indexu fráze ve slovníku a prvního znaku, který se v uložené frázi nenachází. Fráze ve slovníku je pak o tento znak rozšířena a označena dalším pořadovým číslem indexu fráze. [10][12][13]

Jako příklad si uvedeme zprávu „abcabcabcbcba“.

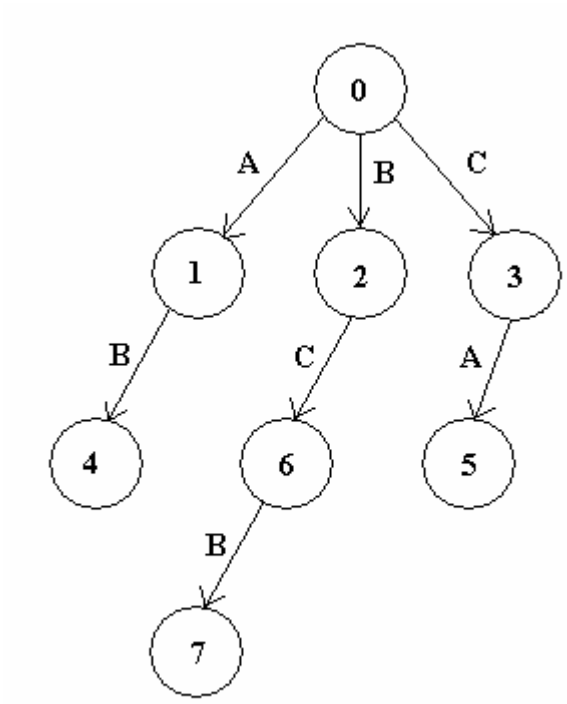
1) V prvním kroku zapíšeme do výstupního řetězce [0A], čili prázdný index fráze a znak A.
2) Následně zapíšeme [0B], čili prázdný index fráze a znak B.
3) Následně zapíšeme [0C], čili prázdný index fráze a znak C.
4) V dalším kroku najdeme řetězec A s indexem 1, zapíšeme tedy [1B], čili řetězec AB.
5) Následně najdeme řetězec C s indexem 3, zapíšeme tedy [3A], čili řetězec CA.
6) Následně najdeme řetězec B s indexem 2, zapíšeme tedy [2C], čili řetězec BC.
7) V dalším kroku najdeme řetězec BC s indexem 6, zapíšeme tedy [6B], čili řetězec BCB.
8) V dalším kroku najdeme řetězec A s indexem 1, zapíšeme tedy [1-], čili řetězec A - konec.

Během tohoto postupu kódování viz. Obr. č. 7. – Slovník algoritmu LZ78, nám zároveň vznikl slovník viz Tab. Č. 11 – Slovník algoritmu LZ78, který ovšem není potřeba

ukládat spolu s komprimovanou zprávou. Dekomprese probíhá opačným způsobem a během ní se slovník vytvoří znovu.

Index fráze	Fráze	Výstup
1	a	0A
2	b	0B
3	c	0C
4	ab	1B
5	ca	3A
6	bc	2C
7	bcb	6B
8	a -	1 - konec

Tab. 11 - Slovník algoritmu LZ78



Obr. 7 – Slovník algoritmu LZ78

Výstupní řetězec je ve tvaru: 0A0B0C1B3A2C6B1-. Následná dekomprese probíhá stejným způsobem, přičemž se opět tvoří slovník z již dekomprimovaných dat.

1) První slovo 0A přečteme jako A a do slovníku uložíme frázi A s indexem1.
2) Další slovo 0B přečteme jako B a do slovníku uložíme frázi B s indexem 2.
3) Další slovo 0C přečteme jako C a do slovníku uložíme frázi C s indexem 3.
4) Další slovo 1B přečteme jako frázi s indexem 1 a k ní připojíme B čili AB. Do slovníku uložíme frázi AB s indexem 4.
5) Další slovo 3A přečteme jako frázi s indexem 3 a k ní připojíme A čili CA. Do slovníku uložíme frázi CA s indexem 5.
6) Další slovo 2C přečteme jako frázi s indexem 2 a k ní připojíme C čili BC. Do slovníku uložíme frázi BC s indexem 6.
7) Další slovo 6B přečteme jako frázi s indexem 6 a k ní připojíme B čili BCB. Do slovníku uložíme frázi BCB s indexem 7.
8) Poslední slovo 1 – přečteme jako frázi s indexem 1 čili A a ukončíme dekompresi. Slovník můžeme stejně jako při kompresi vymazat z paměti.

Slovník, který vznikne při kompresi, je shodný se slovníkem, který vznikne při dekompresi. Právě tato vlastnost nám dovoluje slovník po kompresi či dekompresi vymazat a nemusí být uložen společně s komprimovanou zprávou. V praxi se implementace tohoto algoritmu liší velikostí použitého slovníku a způsobu vyhledávání v něm. Nejčastěji se používá slovník o 2^{12} položkách. Efektivita tohoto algoritmu je závislá na vstupních datech a zpravidla se tato metoda kombinuje s dalšími metodami.

3.4.4. Algoritmus LZ77

Předchůdce algoritmu LZ78 je algoritmus LZ77 jehož autorem je opět dvojice Lempel, Ziv. Tato metoda vznikla o rok dříve než LZ78 a z pohledu komprese a dekomprese je slovníková, jednoprůchodová, adaptivní a především asymetrická – komprese je mnohem náročnější než dekomprese.

Metoda LZ77 je založena na tzv. klouzavém okénku. Algoritmus využívá dva myšlené buffery. Jsou to Search buffer a Look-ahead buffer. Jejich velikost je konstantní a vzhledem ke vstupním datům malá. Algoritmus pracuje pouze s daty v těchto buffrech. Z toho vyplývá že, paměťová náročnost algoritmu neroste s délkou vstupního řetězce jako

u metody LZ78. Délka look-ahead bufferu musí být vždy menší nebo rovna délce search bufferu.

Algoritmus nejprve uloží první symbol zprávy bez prefixu a poté začne prohledávat zprávu v momentě, kdy je naplněn Look-ahead buffer a Search buffer obsahuje první symbol. V každém kroku vyhledá nejdelší prefix Look-ahead bufferu, který začíná v Search-bufferu. Ten zakóduje jako trojici obsahující pozici prefixu (počítá se jako vzdálenost od prvního znaku Look-ahead bufferu zleva), délku prefixu a první symbol za prefixem, který se nepodařilo nalézt. Poté posune zprávu v obou bufferech o délku právě nalezeného prefixu zvýšenou o 1.

Slovník je v tomto případě vytvořen a zapomenut při každé iteraci a nemá smysl se jím zabývat.

Jako příklad si uvedeme zprávu: „aababcababababcacc“ a použijeme délku Search bufferu 8 symbolů a Look-ahead bufferu 4 symboly.

1) V prvním kroku je v Look-ahead bufferu: [aaba] a v Search bufferu není nic [-]. Jako výstup se uloží první symbol zprávy [0,0,A]. A zprávu posune o 1.
2) V dalším kroku je v Look-ahead bufferu: [abab] a v Search bufferu [a]. Jako výstup se uloží prefix A, který je vzdálen o 1 symbol a má délku 1 symbol a k němu se připojí symbol B čili [1,1,B]. Zpráva se nyní posune o 2.
3) V dalším kroku je v Look-ahead bufferu: [abca] a v Search bufferu [aab]. Jako výstup se uloží prefix AB, který je vzdálen o 2 symboly, má délku 2 symboly a k němu se připojí symbol C čili [2,2,C]. Zpráva se nyní posune o 3.
4) V dalším kroku je v Look-ahead bufferu: [abab] a v Search bufferu [aababc]. Jako výstup se uloží prefix ABAB, který je vzdálen o 5 symbolů a má délku 4 symboly. K němu se připojí symbol A čili [5,4,A]. Zpráva se nyní posune o 5.
5) V dalším kroku je v Look-ahead bufferu: [bcac] a v Search bufferu [abcababa]. Jako výstup se uloží prefix BCA, který je vzdálen o 7 symbolů a má délku 3 symboly. K němu se připojí symbol C čili [7,3,C]. Zpráva se nyní posune o 4.
6) V posledním kroku je v Look-ahead bufferu: [c---] a v Search bufferu [bababcac]. Jako výstup se uloží prefix C, který je vzdálen o 1 symbol a má délku 1 symboly. K němu se připojí symbol - čili [1,1,-]. Zpráva končí.

Výstupní řetězec je ve tvaru: 00A11B22C54A73C11-. Následná dekomprese probíhá podobným způsobem, přičemž se používají již dekomprimovaná slova.

1) První slovo 00A přečteme jako A a do Search bufferu uložíme frázi A.
2) Další slovo 11B přečteme jako prefix dlouhý 1 symbol v Search bufferu vzdálený od Look-ahead bufferu o 1 symbol. Prefix je tedy A a k němu přidáme symbol B čili řetězec AB. Do Look-ahead bufferu uložíme vytvořený řetězec AB a celý řetězec posuneme o 2. Search buffer nyní obsahuje AAB.
3) Další slovo 22C přečteme jako prefix dlouhý 2 symboly v Search bufferu vzdálený od Look-ahead bufferu o 2 symboly. Prefix je tedy AB a k němu přidáme symbol C čili řetězec ABC. Do Look-ahead bufferu uložíme vytvořený řetězec ABC a celý řetězec posuneme o 3 symboly. Search buffer nyní obsahuje AABABC.
4) Další slovo 54A přečteme jako prefix dlouhý 4 symboly v Search bufferu vzdálený od Look-ahead bufferu o 5 symbolů. Prefix je tedy ABAB a k němu přidáme symbol A čili řetězec ABABA. Do Look-ahead bufferu uložíme vytvořený řetězec ABABA a celý řetězec posuneme o 5 symbolů. Search buffer nyní obsahuje ABCABABA.
5) Další slovo 73C přečteme jako prefix dlouhý 3 symboly v search bufferu vzdálené od Look-ahead bufferu o 7 symbolů. Prefix je tedy BCA a k němu přidáme symbol A čili řetězec BCAC. Do Look-ahead bufferu uložíme vytvořený řetězec BCAC a celý řetězec posuneme o 4 symboly. Search buffer nyní obsahuje BABABCAC.
6) Poslední slovo 11- přečteme jako prefix dlouhý jeden symbol v search bufferu vzdálený od Look-ahead bufferu o 1 symbolů. Prefix je tedy C a k němu přidáme symbol - čili řetězec C- konec. Do Look-ahead bufferu uložíme vytvořený řetězec C - konec a celý řetězec posuneme o 2 symboly. Search buffer nyní obsahuje BABCACC-.

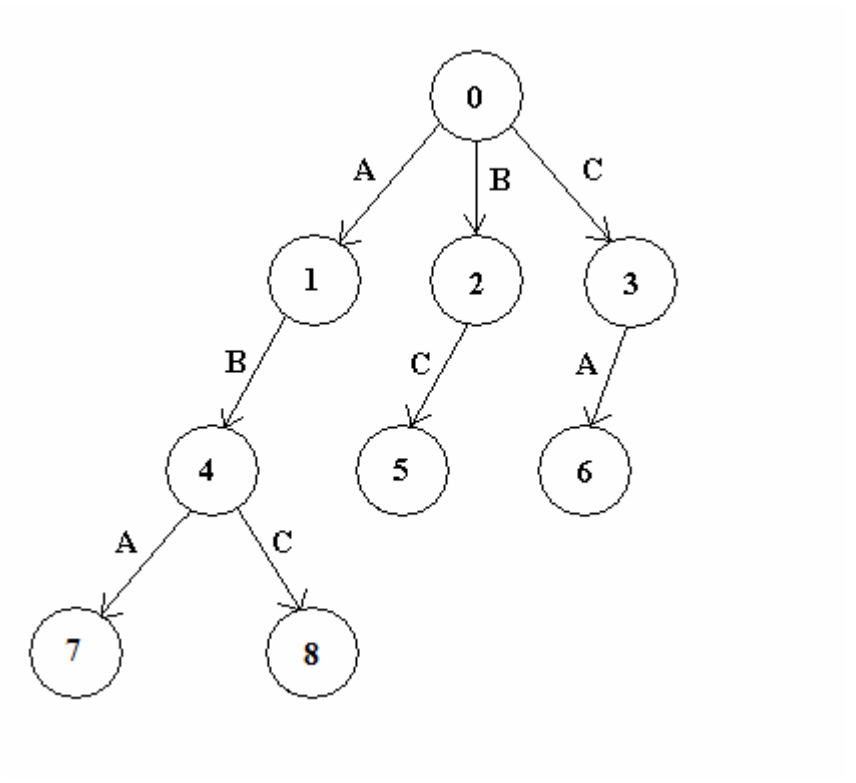
Výsledná dekomprimovaná zpráva „aababcababababcacc“ je shodná se zprávou před komprimací. Nevýhodou této metody je větší délka kódového slova oproti metodě LZ78 a náročnější komprese z pohledu hledání prefixu.

3.4.5. Algoritmus LZW

V roce 1984 autoři Lempel, Ziv a Welch ještě zdokonalili metodu LZ78. Podařilo se jim vymyslet algoritmus, kterému stačí kódové slovo délky pouze jednoho znaku! To má za následek, že se před vlastní dekompresí musí do slovníku načíst všechny symboly vstupní abecedy. Tento algoritmus je sice jednoduchý na implementaci, ale má velkou paměťovou náročnost.

V každé iteraci algoritmu se hledá nejdelší fráze ve slovníku, která odpovídá dosud nezpracované části vstupního řetězce. Na výstup je poté vloženo kódové slovo, které se skládá z indexu fráze ve slovníku. Fráze ve slovníku je pak o tento znak rozšířena a označena nejmenším možným číslem stejně jako je tomu u algoritmu LZ78.[10]

Jako příklad si uvedeme zprávu: „abcababca“. Nejdříve se musí uložit všechny symboly vstupního řetězce.



Obr. 8 – Slovník algoritmu LZW

1) Nejprve do slovníku uložíme symboly A, B, a C a přidělíme jim nejmenší možné indexy, čili A-1, B-2, C-3.
2) V dalším kroku se přečte nejdelší řetězec, který se zároveň nachází ve slovníku, čili A. Na výstup zapíšeme index tohoto slova – 1 a do slovníku zapíšeme slovo

rozšířené o následující symbol - B. Tomuto novému slovu přiřadíme nejmenší možný index čili AB-4

3) V dalším kroku se přečte následující nejdelší možný řetězec, který se zároveň nachází ve slovníku, čili B. Na výstup zapíšeme index tohoto slova – 2 a do slovníku zapíšeme slovo rozšířené o následující symbol - C. Tomuto novému slovu přiřadíme nejmenší možný index čili BC-5.
4) V dalším kroku se přečte následující nejdelší možný řetězec, který se zároveň nachází ve slovníku, čili C. Na výstup zapíšeme index tohoto slova – 3 a do slovníku zapíšeme slovo rozšířené o následující symbol - A. Tomuto novému slovu přiřadíme nejmenší možný index čili CA-6.
5) V dalším kroku se přečte následující nejdelší možný řetězec, který se zároveň nachází ve slovníku, čili AB. Na výstup zapíšeme index tohoto slova – 4 a do slovníku zapíšeme slovo rozšířené o následující symbol - A. Tomuto novému slovu přiřadíme nejmenší možný index čili ABA-7.
6) V dalším kroku se přečte následující nejdelší možný řetězec, který se zároveň nachází ve slovníku, čili AB. Na výstup zapíšeme index tohoto slova – 4 a do slovníku zapíšeme slovo rozšířené o následující symbol - C. Tomuto novému slovu přiřadíme nejmenší možný index čili ABC-8.
7) V posledním kroku se přečte následující nejdelší možný řetězec, který se zároveň nachází ve slovníku, čili CA. Na výstup zapíšeme index tohoto slova – 6. Ve vstupním řetězci už není žádný další symbol, proto je v tomto kroku komprese ukončena.

Takto jsme dostali na výstup řetězec ve tvaru: 123446 a seznam vstupních symbolů ABC. Zároveň nám vznikl slovník viz. Obr. č. 8. – Slovník algoritmu LZW, který opět nemusíme zachovat, neboť při dekompresi postupně vytvoříme původní vstupní data se stejným slovníkem. Dekomprese probíhá následujícím způsobem:

1) Nejprve do slovníku uložíme přijaté symboly A, B, a C a přidělíme jim nejmenší možné indexy, čili A-1, B-2, C-3.
2) Načteme první slovo ze vstupního řetězce – 1. Slovo podle slovníku přeložíme jako řetězec A který uložíme na výstup. Ve vstupním řetězci následuje slovo –

2, proto rozšíříme slovník o slovo složené z právě zpracovaného slova a prvního symbolu následujícího slova. Takto vzniklému slovu AB ve slovníku přiřadíme nejmenší možný index slova – 4.

3) Načteme další slovo ze vstupního řetězce – 2. Slovo podle slovníku přeložíme jako řetězec B, který uložíme na výstup. Ve vstupním řetězci následuje slovo – 3, proto rozšíříme slovník o slovo složené z právě zpracovaného slova a prvního symbolu následujícího slova. Takto vzniklému slovu BC ve slovníku přiřadíme nejmenší možný index slova – 5.
4) Načteme další slovo ze vstupního řetězce – 3. Slovo podle slovníku přeložíme jako řetězec C, který uložíme na výstup. Ve vstupním řetězci následuje slovo – 4, proto rozšíříme slovník o slovo složené z právě zpracovaného slova a prvního symbolu následujícího slova. Takto vzniklému slovu CA ve slovníku přiřadíme nejmenší možný index slova – 6.
5) Načteme další slovo ze vstupního řetězce – 4. Slovo podle slovníku přeložíme jako řetězec AB, který uložíme na výstup. Ve vstupním řetězci následuje slovo – 4, proto rozšíříme slovník o slovo složené z právě zpracovaného slova a prvního symbolu následujícího slova. Takto vzniklému slovu ABA ve slovníku přiřadíme nejmenší možný index slova – 7.
6) Načteme další slovo ze vstupního řetězce – 4. Slovo podle slovníku přeložíme jako řetězec AB, který uložíme na výstup. Ve vstupním řetězci následuje slovo – 6, proto rozšíříme slovník o slovo složené z právě zpracovaného slova a prvního symbolu následujícího slova. Takto vzniklému slovu ABC ve slovníku přiřadíme nejmenší možný index slova – 8.
7) Nakonec načteme poslední slovo ze vstupního řetězce – 6. Slovo podle slovníku přeložíme jako řetězec CA, který uložíme na výstup. Ve vstupním řetězci nenásleduje další slovo, proto ukončíme dekompresi a odstraníme slovník.

Výsledná dekomprimovaná zpráva „abcababca“ je shodná se zprávou před komprimací. Tato metoda se používá u formátu ZIP v mnoha různých modifikacích.

3.5. Kryptografické kódy

Kryptografické kódování dat má za cíl utajení přenášené informace. Kryptografické kódování dat je často označováno jako „šifrování“ dat. Důvodem k utajení informace citlivé na zneužití v určitém časovém horizontu je přenos přes nezabezpečené prostředí. Takovým prostředím může být například internet, pošta, či jiná veřejná datová síť. Časovým horizontem se rozumí doba, po kterou je potřeba informaci uchovat v tajnosti. Například informace o vítězi soutěže je tajná až do vyhlášení výsledků. Po vyhlášení výsledků už tuto informaci není potřeba tajit, ani šifrovat.

Šifrování obecně zahrnuje veškeré snahy o utajení informace počínaje jejím ukrytím tzv. steganografie až po složité algoritmy a mechanismy, které znemožní její přečtení. O tom, že informace mají cenu zlata, se můžeme přesvědčit z historie. USA například na některé speciální typy kódů uvalil vojenské embargo a vynakládají nemalé finanční prostředky z ministerstva obrany pro zdokonalení těchto kódů. Nejslavnější německý kódovací stroj má název Enigma a sehrál velikou roli ve 2. Světové válce.

S pojmem kryptografie úzce souvisí pojem kryptoanalýza. Kryptoanalýzou se snažíme získat ze zašifrovaných dat původní informaci bez znalosti šifrovacího klíče. Metody, které se ke kryptoanalýze používají, zahrnují tři základní postupy a to: prolamování analytickou myšlenkou, prolamování hrubou silou a prolamování hrubou lidskou silou tzv. „pendreková metoda“. Obranou proti těmto prolomení je použití „silné šifry“. U silné šifry je zajištěno, že ji nelze prolomit žádnou kryptoanalytickou metodou kromě použití hrubé síly. Použití hrubé síly znamená využití číslicového potenciálu k prohledání veškerých možných kombinací pro nalezení klíče. V případě silné šifry je však klíč tak složitý, že prohledání není možné provést v požadovaném čase, nebo je úplné prohledání technicky nemožné. Za silné šifry jsou považovány moderní symetrické metody s délkou klíče větší než 70 bitů a asymetrické šifry typu RSA s délkou klíče nad 700 bitů. [12]

3.5.1. Substituční algoritmy

Substituční šifra obecně nahrazuje každý symbol zprávy jiným symbolem podle určitého pravidla. Pro dešifrování tj. pro zpětnou transformaci zakódované zprávy zvanou též šifrovaný text, na původní informaci zvanou též otevřený text, je potřeba znát toto pravidlo a použít jej inverzně.

V klasické kryptografii se používají čtyři základní typy šifer:

1) Monoalfabetická substituční šifra – zvaná také jednoduchá substituční šifra. Spočívá v prostém nahrazení každého symbolu otevřeného textu jiným příslušným znakem šifrovaného textu.
2) Homofonní substituční šifra – je podobná jako monoalfabetická substituční šifra, ale místo přiřazení jednoho znaku šifrovaného textu je možné přiřadit více různých znaků pro jeden symbol otevřeného textu. Pro různé symboly otevřeného textu může existovat různý počet znaků šifrovaného textu.
3) Polygramová substituční šifra – je založena na šifrování skupin symbolů. To znamená, že skupině dvou nebo více symbolů otevřeného textu odpovídá skupina dvou nebo více znaků šifrovaného textu.
4) Polyalfabetická substituční šifra – se skládá z několika jednoduchých šifer, které se mezi sebou střídají podle určitého pravidla. [12]

Jednu z prvních substitučních šifer použil Julius Caesar a je také po svém autorovi pojmenována Caesarova šifra. Jedná se o Monoalfabetickou substituční šifru, která je symetrická. Princip spočívá v nahrazení právě kódovaného symbolu symbolem ležícím v abecedě o 3 pozice dříve. Algoritmus šifry je tedy posun symbolů v abecedě a tajným klíčem je „3“ tj. údaj o velikosti posunu.

Jako příklad si uvedeme otevřený text: Julius. Posuneme-li každý symbol o 3 znaky v abecedě zpět, dostaneme zašifrovaný text: Grifrp. Pro dešifrování použijeme inverzní postup, čili posuneme přečtené znaky v abecedě o délku tajného klíče dál. Dostaneme zpět otevřený text: Julius.

Takový algoritmus je snadné prolomit, neboť existuje pouze 30 variant tajného klíče. Nelze posunout o větší počet znaků než je počet znaků abecedy a realizace posunu vpřed je prakticky doplňkem posunu vzad do 31. Stačí tedy vyzkoušet aplikaci všech 31 klíčů a stoprocentně se dostaneme k původní informaci.

Dalším zdokonalením Caesarovi šifry se zabýval Blaire de Vigenere. Princip jeho šifry spočívá v tom, že používá pro každý symbol zprávy jinou Caesarovu šifru. Odesilatel i příjemce musí znát způsob, jakým určí velikost Caesarova posunu v abecedě. Tento způsob je definovaný klíčem, který se zapíše cyklicky do řádku nad otevřenou abecedu.

Vigenerova šifra využívá tzv. Vigenerův čtverec. Ten vznikne tak že se řádek abecedy otevřeného textu posune o jeden symbol doleva a zapíše na další řádek. To se opakuje až do zapsání stejného řádku jako je otevřený text.

Šifrování probíhá následujícím způsobem. Nad řádek otevřeného textu zprávy se zapíše šifrovací klíč cyklicky, dokud není zapsán nad celou zprávou. Pro zašifrování každého symbolu se použije Caesarova šifra s posunem, který odpovídá pořadí symbolu zapsaného nad symbolem z otevřeného textu. Jinými slovy se použije ten řádek z Vigenerovi šifry, který začíná stejným symbolem, jako symbol z řádku šifrovacího klíče, který se nachází nad právě šifrovaným symbolem. Dešifrování probíhá opačným způsob za použití téhož klíče. Vigenerova šifra je polyalfabetická substituční šifra.

Krátce po druhé světové válce, ve které vznikaly první kryptoanalytické matematické metody, se začal používat jednoduchý šifrovací systém a to pouhé sčítání otevřeného textu s náhodným heslem. Z teorie publikované americkým vědcem Shannonem vyplynulo, že jediný absolutně bezpečný systém je sčítání otevřeného textu se stejně dlouhým náhodným klíčem. Musí se ovšem dodržet určité podmínky:

1) Možnost vyrobit stejně dlouhé náhodné heslo se stejnou pravděpodobností.
2) Vlastnost dostatečně bezpečného kanálu pro přenos hesla.
3) Žádné heslo nesmí být použito opakovaně.

Po mnoha dalších pokusech o dostatečně silné šifrovací algoritmy vznikl celosvětově uznávaný Data Encryption Standard tzv. DES. Po něm pak následuje dodnes používaný Advanced Encryption Standard tzv AES.

V moderní kryptografii dělíme metody na symetrické a asymetrické. Symetrické metody používají jediný klíč a též se nazývají konvenční. Asymetrické metody pak používají veřejný klíč a soukromý klíč.

Převratem v přístupu je Kerckhoffův princip, který říká: „Utajení a bezpečnost zašifrovaných dat nesmí záležet na utajení postupu, kterým se šifrují. Naopak, vždy se musí předpokládat, že Váš nepřítel zná šifru (algoritmus) do nejmenších detailů. Utajení musí spočívat pouze v klíči (hesle), který nezná nikdo jiný“. [12]

3.5.2. Symetrické algoritmy

Symetrický klíč používá pro šifrování i dešifrování stejný klíč. Výhodou je rychlost šifrovacího algoritmu, nevýhodou pak problém se sdílením klíče mezi odesílatelem a

příjemcem zprávy. Nejjednodušším symetrickým algoritmem je již zmiňovaná Caesarova šifra. [14]

Symetrické šifry se dělí na proudové, kde se zpráva šifruje po jednotlivých symbolech, a na blokové, kde se zpráva rozdělí na jednotlivé bloky a ty se pak kódují samostatně. V praxi se více využívají blokové šifry s délkou bloku 64 bitů nebo 128 bitů.

Charakteristika vybraných nejpoužívanějších šifer naznačuje, jaký důraz byl v historii kladen na vývoj v oblasti kryptografie:

1) DES – Firmou IBM vyvinutý standard, který se v roce 1977 stal Americkou vládní normou pro šifrování. Dnes již neperspektivní. Jeho cílem je transformovat zprávu na šifrovaný text, tak aby z něj nebylo možné vytvářet statistické závěry. Jednoduché šifry například Caesarova nebo Vingenerova vytvářejí šifrovaný text, ze kterého se dá statisticky odvodit klíč. Standard DES proto zavádí podmínku, že algoritmus musí vytvářet šifrované texty se stejnou pravděpodobností, aby nebylo možné prolamovat klíč statistickými metodami.
2) TripleDES – Snaha o vylepšení předchozího DES.
3) IDEA – Firmou Ascom-Tech vyvinutý algoritmus se 128 bitovým klíčem. Jeho předností je vysoká rychlost a bezpečnost. Dodnes hojně využíván.
4) Blowfish – Brucem Screinerem vyvinutý algoritmus, který má proměnnou délku klíče od 32 bitů do 448 bitů.
5) CAST – Algoritmus navržený autory Carlisle Adamsem a Staffordem Tavernsem, který používá bloky o délce 64 bitů a proměnnou délku klíče, stejně jako Blowfish. CAST byl firmou Entrust Technologies postoupen pro volné užití na rozdíl od patentem chráněné šifry IDEA. ČÁST je imunní vůči diferenciální i lineární kryptoanalýze. Tyto dvě kryptoanalytické metody jsou nejúčinnější publikované metody ze všech doposud známých.
6) SKIPJACK – Blokový algorytmus s délkou bloku 64 bitů a délkou klíče 80 bitů byl vyvinut NSA (National Security Agency). Tento algorytmus obsahuje možnost zrekonstruovat zpětně klíč tzv. key recovery. Pro tuto vlastnost byl velice kritizován vzhledem ke Kerckhoffovu principu.
7) AES – Je norma blokové šifry, která udává minimální požadavky pro moderní šifry. Prvním požadavkem je podpora šifrování bloků o 128 bitech. Druhým požadavkem je podpora šifrovacích klíčů o délkách 128, 192 a 256 bitů.

V roce 2002 tyto požadavky splnila šifra Rijndaelova 128 bitová bloková šifra, která je nadále označovaná jako AES.

Pro demonstraci postupu při šifrování si ukážeme šifru DES:

DES používá 64 bitové bloky, to znamená, že se zpráva musí rozdělit po 64 bitech a poslední blok vhodně doplnit na 64 bitů. Šifrovací klíč má délku 56 bitů, to znamená, že existuje 2^{56} = 72057594037927936 možných klíčů.

DES provádí 16 iterací šifrovacího algoritmu, kde se používají pevně stanovené substituce a transpozice. V každé z iterací je použit jiný 48 bitový klíč, který je odvozen z původního 56 bitového klíče přesně danými pravidly. [15]

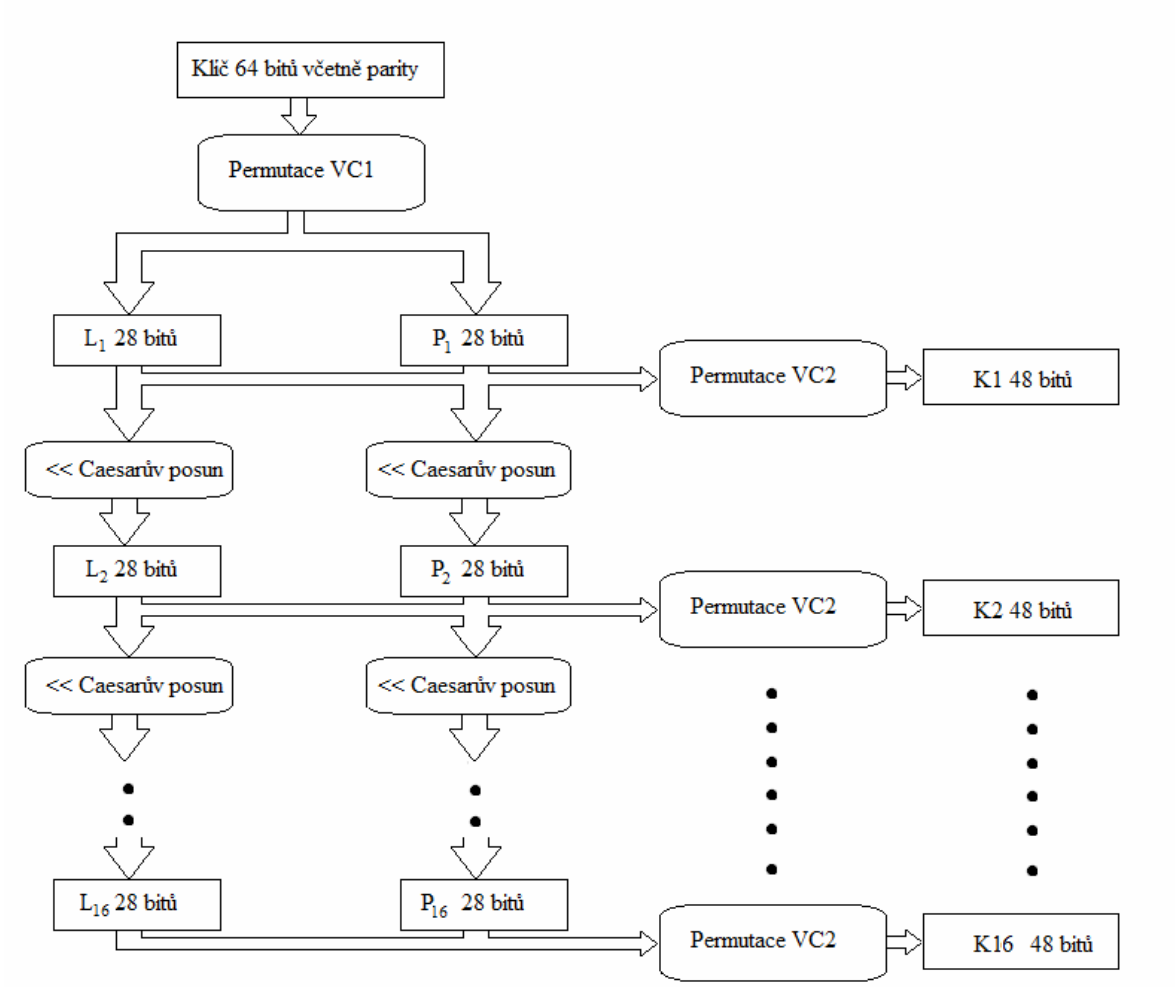
Před vlastním šifrováním je tedy potřeba vygenerovat iterační 48 bitové klíče K1 – K16. Zdrojový 56 bitový klíč je chráněn proti selhání přenosu pomocí další 8 paritních bitů.

Postup generování těchto klíčů je následující:

1) Vynechají se paritní bity, tj. bity, které slouží jen k ochraně proti chybám v přenosu klíče.
2) Provede se permutace zbylých 56 bitů podle tabulky permutace viz Tab. Č. 12 – Permutace klíče VC2.
3) Výsledný 56 bitový řetězec se rozdělí na dvě 28 bitové poloviny L a P.
4) Poté se každá z obou polovin L a P transponuje Caesarovou šifrou doleva o počet znaků daný pořadím iterace viz Tab. Č. 13 - Velikost levých posunů LS
5) Dále se obě poloviny dají zpět vedle sebe a na výsledný blok 56 bitů se použije permutace podle tabulky Permutace VC2 viz Tab. Č. 14 - Permutace klíče VC2. Při této permutaci dojde také k redukci na 48 bitů. Některé bity totiž nejsou použity, například není použit bit č. 9.
6) Po této permutaci se zapíše 48 bitový cyklický klíč Kx kde x značí pořadí permutace a vrátí se ke kroku 4.

Vzniknou nám tedy 48 bitové klíče K1 – K16.

Celý postup generování cyklických klíčů K1 – K16 je graficky znázorněn na následujícím obrázku viz. Obr. č. 9. – Generování cyklických klíčů DES.



Obr. 9 – Generování cyklických klíčů DES

57	49	41	33	25	17
1	58	50	42	34	26
10	2	59	51	43	35
19	11	3	60	52	44
63	55	47	39	31	23
7	62	54	46	38	30
14	6	61	53	45	37
21	13	5	28	20	12

Tab. 12 – Permutace klíče VC1

Iterace	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Posun	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tab. 13 - Velikost levých posunů LS

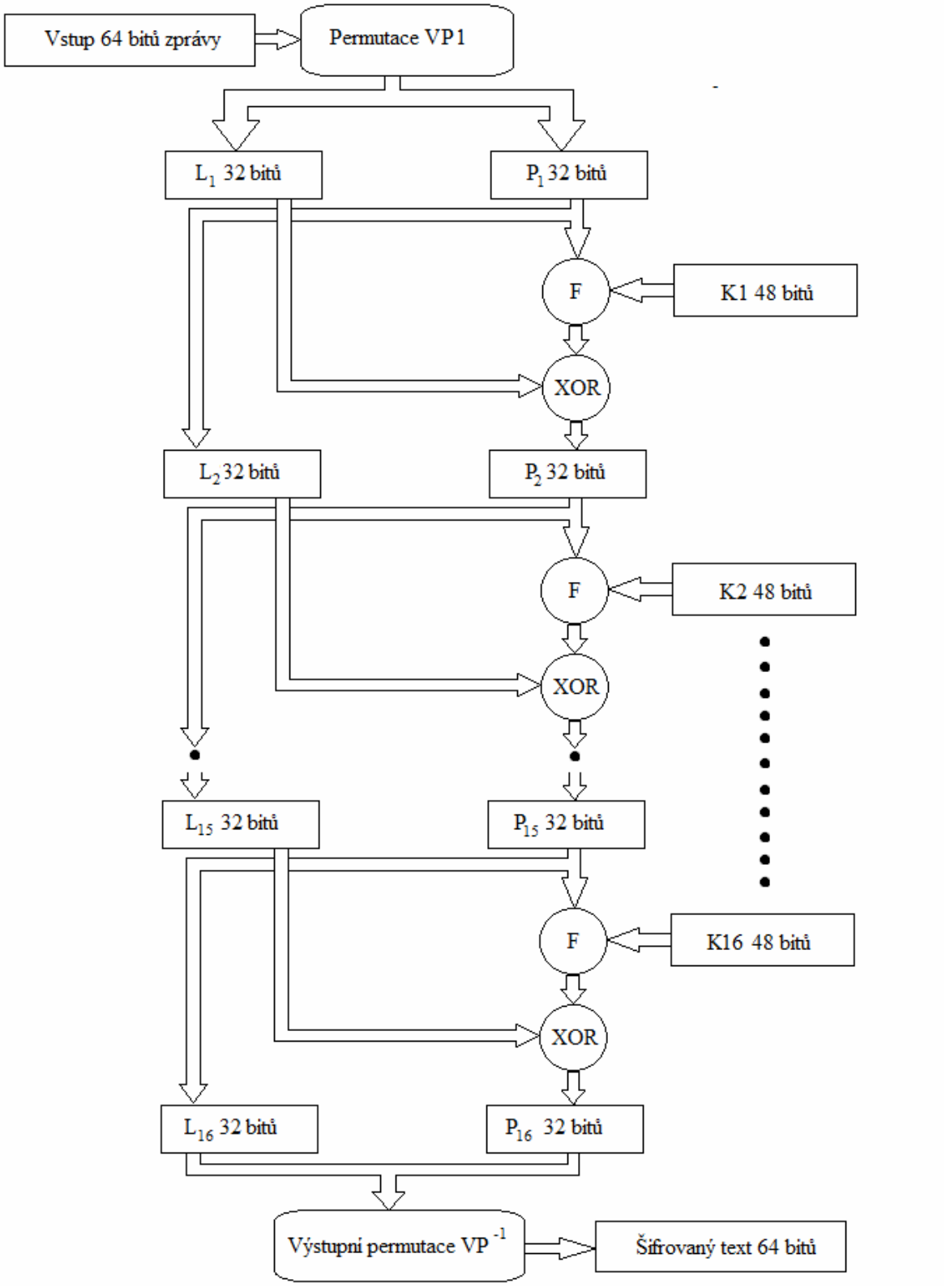
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tab. 14 - Permutace klíče VC2

Postup vlastního šifrování je zobrazen na následujícím obrázku viz. Obr. č. 10. – Postup šifrování DES.

1) Vstupem je blok 64 bitů zprávy a v předchozím kroku připravených 16 iteračních klíčů K1 - K16.
2) Dále se provede permutace vstupních 64 bitů podle tabulky permutace viz Tab. Č. 15 – Vstupní permutace VP. Výstupem je 64 bitový blok.
3) Tento blok se rozdělí na dvě 32 bitové poloviny L a P.
4) Následuje 16 iterací, ve kterých se pokaždé pravá polovina P zapíše do následující levé poloviny L a následující pravá polovina P se vypočítá jako funkce levé poloviny L XOR s upravenou pravou polovinou P (F). Úprava pravé poloviny funkcí F – Fiestelovou funkcí bude vysvětlena později.

Po proběhnutí všech iterací se levá i pravá polovina L a P spojí a transformuje pomocí výstupní permutace viz Tab. Č. 16 – Výstupní permutace VP^{-1}. Výstupem je 64 bitový blok šifrovaného textu.



Obr. 10 – Postup šifrování DES

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tab. 15 – Vstupní permutace VP

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tab. 16 – Výstupní permutace VP-1

Při postupu DES šifrování se v každé iteraci používá funkce F – Fiestelova funkci. Tato funkce realizuje použití klíče a pracuje následujícím způsobem:

1) Pravá 32 bitová polovina P je expandována (rozšířena) pomocí Fiestelovi expanze viz Tab. Č. 18 – Fiestelova expanze, na 48 bitů. Některé bity jsou použity vícekrát.
2) K tomuto expandovanému bloku se přičte příslušný klíč pomocí funkce XOR K a vznikne tak 48 bitový blok.
3) Tento 48 bitový blok rozdělíme na 8 bloků B1 – B8. Každý z bloku B1 - B8 má 6 bitů.
4) Každý z bloků je dále transformován tzv. S-Boxy na bolky S1-S8 viz Tab. Č. 17 – Fiestelovy S-boxy. Transformace je určena podle tabulky tak, že číslo řádku tabulky Fiestelova S-boxu je dáno prvním a posledním bitem transformovaného bloku čteným v binárním tvaru a číslo sloupce Fiestelova S-boxu je dáno 2. Až 5. bitem transformovaného bloku čteným v binárním tvaru.

5) Nakonec se bloky S1-S8 spojí v jeden 32 bitový blok a ten se transformuje pomocí Fiestelovy permutace viz Tab. Č. 19 – Fiestelova permutace. Výstupem je opět 32 bitový blok, který se nadále používá v dalších iteracích šifry DES.

Box	Řádek	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	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
S2	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	3	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3	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
S4	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
S5	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	2	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	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
S7	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
S8	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

Tab. 17 – Fiestelovy S-boxy

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Tab. 18 – Fiestelova expanze

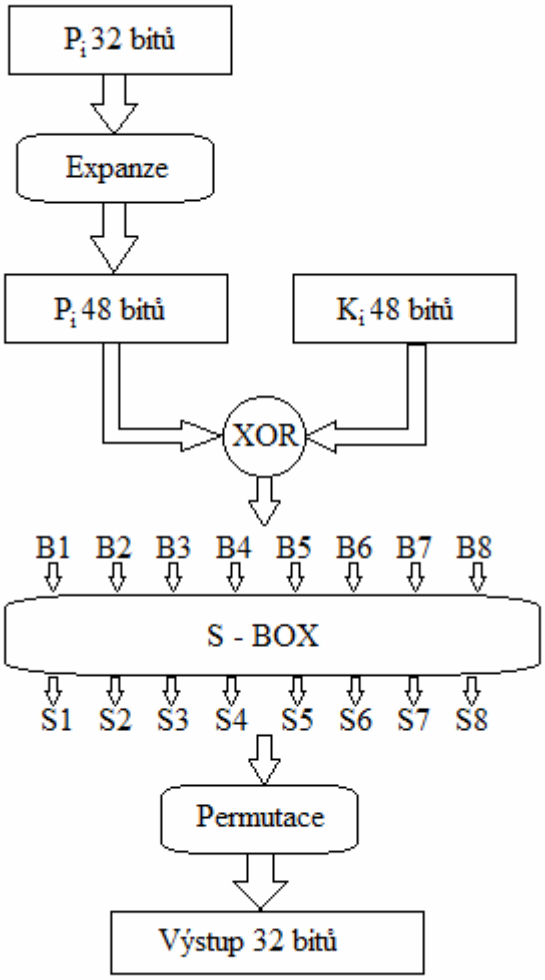
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Tab. 19 – Fiestelova permutace

Na následujícím obrázku je graficky znázorněna funkce F – Fiestelova funkce viz. Obr. č. 11. – Fiestelova funkce.

Pro dešifrování se používá stejný algoritmus jako pro šifrování s tím rozdílem, že klíče jsou používány v opačném pořadí, čili K16 - K1.

Cílem takto zdlouhavých a opakujících se operací je zabránit, popřípadě co nejvíce ztížit prolomení šifry analytickými metodami. Tyto metody využívají statistických metod a metod analogických úsudků k získání klíče nebo cesty vedoucí ke klíči. Vzhledem ke Kerckhoffovu principu je klíč upraven na 16 variant poměrně složitým způsobem a následně opět poměrně složitým způsobem 16 krát použit, aby případná kryptoanalýza byla co nejsložitější a časově tak náročná, že vyřešení kryptoanalytické úlohy nebude možné v dostatečně krátkém čase. Tím že jsou moderní stroje čím dál rychlejší, zkracuje se i doba potřebná k prolomení takových algoritmů. Proto se vyvinula nástavba DES zvaná TripleDES jinak také 3DES. Tato metoda používá tři implementace algoritmu DES.



Obr. 11 – Fiestelova funkce

3.5.3. Asymetrické algoritmy

Asymetrické algoritmy používají veřejný klíč. Pro každého uživatele existuje dvojice klíčů, z níž jeden je veřejný a druhý soukromý. Veřejný klíč je všeobecně přístupný a pro všechny uživatele stejný. Slouží k zašifrování zprávy pro konkrétního příjemce. Soukromý klíč má každý uživatel schovaný u sebe a pro ostatní uživatele je tento klíč tajný. Proto by měl být soukromý klíč chráněn proti odcizení. Soukromý klíč slouží k dešifrování zprávy určené právě pro dešifrování tímto klíčem. Ostatní uživatelé se svými klíči tuto zprávu nedešifrují. Jinými slovy zpráva se zašifruje tak, aby ji svým soukromým klíčem mohl dešifrovat jen jeden uživatel, právě ten, kterému je zpráva určena.

Asymetrické šifry využívají jednocestné funkce. Ze vstupu lze snadno získat výstup, ale z výstupu je velice obtížné znovu získat vstup. Příkladem takové operace, kterou lze provést pouze v jednom směru je faktorizace. Vynásobení dvou velkých čísel je triviální úloha. Ovšem získat z tohoto součinu jeho činitele, čili provést faktorizaci, je velice složité.

Soukromý i veřejný klíč spolu úzce souvisí, podmínkou je, aby se z veřejného (šifrovacího) klíče nedal spočítat soukromý (dešifrovací) klíč. Při nesplnění této podmínky postrádá šifrování smysl. Moderní počítače jsou schopné s použitím kryptoanalytických metod například TWINKLE z veřejného klíče soukromý klíč spočítat. Výpočet však potrvá řádově tisíce let. V takovém případě se dá možnost úspěšné kryptoanalýzy odpustit a považovat úlohu za nevyřešitelnou.

Nejvýznamnějším zástupcem asymetrických kryptografických algorytmů je RSA. Byl objeven v roce 1977 a pojmenovaný je po jeho autorech Ronu Rivestovi, Adi Shamirovi a Joeovi Adlemanovi. Vychází z jednocestné funkce násobení dvou dlouhých prvočísel. Předpokládá, že je snadné vynásobit dvě dlouhá prvočísla. Prvočísla řádově 100 místná. Dále předpokládá, že je téměř nemožné tento součin zpětně rozložit na dvě původní prvočísla. Součin nám tedy reprezentuje veřejný klíč a obě prvočísla jsou klíčem soukromým. Vzhledem k tomu že výše uvedené předpoklady nebyly vyvráceny, považuje se algoritmus RSA za bezpečný a běžně se používá například k digitálnímu podpisu.

RSA algoritmus používá pouze čísla, proto se případný text nejprve převede na čísla pomocí převodní tabulky dané abecedy. Znamená to, že se přiřadí každému znaku abecedy jeho unikátní číslo. Tento převod se dá pomocí statistických metod použít také ke komprimaci viz kapitola 3.4 – Kompresní kódy. Šifrovaná zpráva se pak rozdělí na bloky které mají velikost větší než 1 a menší než součin zvolených prvočísel.

Příklad šifrování a dešifrování zprávy z - 123 pomocí RSA algoritmu:

1) Nejprve se zvolí se dvě prvočísla například a = 61 a b = 53.
2) Jejich součin se nazývá veřejný modul v = 61 * 53 = 3233
3) Dále zvolíme veřejný šifrovací exponent e. Musí platit podmínky: $e \prec ((a-1)*(b-1))$, e musí být zároveň s tímto součinem nesoudělné tj. neexistuje jiný společný dělitel než číslo 1. Zvolí se číslo 17.
4) Soukromý klíč je pak definován jako řešení kongruence s * e = 1 mod $((a-1)*(b-1))$ = 2753. Řešení poskytuje Euklidův algoritmus.

5) Šifrování zprávy probíhá podle vzorce: $x = z^e \bmod v$.
   Čili $123^{17} \bmod 3233 = 855$

Uživatelům se odešle šifrovaný text 855. K dešifrování je použit soukromý klíč uživatele, kterému je zpráva určena. Vzorec pro dešifrování je $z = x^s \bmod v = 123$.

Bez znalosti dvou zvolených prvočísel je prakticky nemožné získat soukromý klíč i přes dokonalou znalost algoritmu kódu. Problém při kódování ovšem nastává ve chvíli, kdy se hledají dostatečně velká prvočísla. Proto se využívá pravděpodobnosti a postačují všechna vysoká čísla, která jsou prvočísly jen s velkou pravděpodobností, ale nemusí to být právě prvočísla.

4. Implementace zvoleného kódování

Následující realizace implementace zvoleného kódování byla provedena celorepublikově na kolejových vozidlech ČD a vzhledem k závazku mlčenlivosti nebudou některé konkrétní informace v této práci použity. Popis implementace bude obecný, přesto však jasný a názorný a splní svůj účel ukázat, jak se kódování dat používá v praxi.

Úkolem je realizace datového přenosu mezi jednotkou měření spotřeby elektrické energie kolejového vozidla a dispečinkem pomocí sítě GSM (Global system for mobile communications). Měření elektrické energie bylo zavedeno z důvodu kontroly využití přidělených zdrojů a určení spotřeby pro následnou fakturaci. Jednotka komunikuje s čidlem měření aktuální spotřeby a v diesel-elektrické variantě i s čidlem aktuální hladiny paliva v nádrži. Zároveň má jednotka přehled o své pozici a o čase pomocí technologie GPS (Global positioning system). Jednotka disponuje operačním systémem Linux, respektive jeho upravenou distribucí vyvíjenou speciálně pro tuto jednotku. Ostatní funkce jednotky nejsou popsány zcela záměrně a nadále se jimi nebudeme zabývat.

Nároky na zpracování a přenos informace jsou dány předmětem implementace a aktuálním sociodemografickým stavem obyvatelstva v okolí nádraží ČD. Informace by měla být efektivně komprimována, aby se ušetřily prostředky na zpoplatněném datovém přenosu. S ohledem na strukturu dat obsahujících potřebné informace ze všech měřících přístrojů a s ohledem na použitý operační software pro zpracování byla vybrána metoda komprimace ZIP. Datový přenos probíhá v intervalu 30 sekund. Data jsou z čidel sbírána v intervalu 1 sekundy a ukládána na lokální úložiště dat pro nashromáždění balíku dat k přenosu na dispečink ČD. Pro případ selhání kterékoli části přenosu dat ať už vinou ztráty signálu GSM (například vjede li vlak do tunelu) nebo neopravitelnou chybou v přenosu, která si vyžádá opakování přenosu, si jednotka loguje data na lokálním úložišti dat po celý den, čili 24 hodin. V případě potřeby si dispečink ČD může příkazem přes GSM síť vyžádat kompletní logy z jednotky za posledních 24 hodin najednou. V takovém případě se celý balík dat komprimuje stejně jako jednotlivé balíky aktuálních dat a zkomprimovaný soubor se odešle pomocí GSM sítě zpět. V jednotce je možné na dálku také aktualizovat její software. Funkce aktualizace software však není předmětem této diplomové práce a nebude zde popsána.

Nárok na bezpečnost přenosu dat v síti GSM z pohledu bezchybného přenosu dat není vysoký, neboť síť GSM s podporou GPRS (General packet radio service) už sama o sobě obsahuje kontrolu bezchybného přenosu. Technologie GPRS ovšem není stoprocentní a pro konzistenci dat přijatých dispečinkem ČD je potřeba využít dodatečné kontroly, aby se v případě neopravitelné chyby mohl zavolat požadavek na opětovné odeslání. K chybám by nemělo docházet ve velkém počtu, proto je výhodnější využít případného opětovného přenosu než snažit se veškerá přenesená data zabezpečit co nejlepšími samoopravnými kódy. Proto byla zvolena ochrana Hammingovým kódem, který zbytečně nezvětšuje objem přenášených dat, a přesto dostatečně zabezpečuje přenos.

Rychlost přenosu, která u technologie GPRS pokulhává, není v tomto případě podstatná. Důležité je, že se předpokládaný balík dat odešle i s případným opakováním přenosu za méně než polovinu doby intervalu přenosu.

Nárok na bezpečnost přenosu dat v síti GSM z pohledu utajení už je vyšší. Jednou z mnoha hrozeb zneužití informace z jednotek je na nádražích běžné vykrádání palivových nádrží dieselelektrických lokomotiv. Pokud by měl případný zloděj informace, které jednotka přenáší pomocí GSM a které jsou snadno zachytitelné, měl by přehled o tom, kde se nachází veškeré dieselelektrické lokomotivy, kolik pohonných hmot mají v nádrži a jestli je někdo na stanovišti strojvedoucího, nebo jestli je vlak opuštěný a je snadnou kořistí. Proto je potřeba stanovit čas, po který musí být informace uchována v tajnosti. Jedná se o čas, po kterém je informace pro případné zloděje bezcenná. Zvolený kryptografický kód musí odolat kryptoanalýze alespoň po tuto zvolenou dobu. Vzhledem k použitému softwaru linux, který již disponuje implementací kryptografického kódu DES, byl tento kód také zvolen a podroben kryptoanalýze. Čas potřebný k udržení utajení informace byl zvolen na 24 hodin a ani během dvojnásobku této doby nebyla kryptoanalýza úspěšná, proto lze považovat zvolený kryptografický kód TDES za dostačující řešení. Pokud by se případnému zloději podařilo prolomit šifru po 24 hodinách a získal by potřebné informace, tyto informace budou pro zloděje natolik staré, že je nebude možné zneužít. Jednoduše řečeno nemůže vykrást vlak, který byl na daném místě s plnými nádržemi minulý den.

4.1. Měření veličin

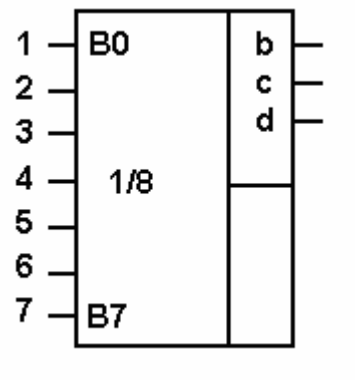
Měřené veličiny požadované ČD jsou: Aktuální stav nafty v nádrži, aktuální spotřeba elektrické energie, pozice vozidla a přesný čas. Další požadavky ze strany ČD zde nejsou uvedeny, neboť nejsou potřebné pro základní implementaci řešení a byly dodány jako nadstandard oproti původnímu zadání.

Aktuální stav nafty je realizován plovákovým čidlem, které je proporcionální. K digitalizaci dochází v A/D převodníku, který je součástí samotného čidla. Řešení A/D převodu bylo popsáno v bakalářské práci. [16][5][6][7]

Aktuální spotřeba elektrické energie je řešena jednak měřením přímo elektroměrem, a zároveň také komparátory na otáčkovém regulátoru trakce v osmi stupních. Zde je realizována první hardwarová komprese pomocí kvantizačního obvodu.
Z komparátorů máme k dispozici signál, který má binární charakter v podobě 7 bitů. V následující tabulce viz Tab. Č. 20 – Pravdivostní tabulka kodéru, je vidět jasně patrný rozdíl, mezi vstupem kvantizačního obvodu tj. sledování úrovní komparátorů a výstupem po zakódování tohoto signálu. Toto přiřazení realizuje kodér, který je na obrázku viz. Obr. č. 12. – Kodér.

B1	B2	B3	B4	B5	B6	B7		b	c	d
0	0	0	0	0	0	0		0	0	0
1	0	0	0	0	0	0		0	0	1
1	1	0	0	0	0	0		0	1	0
1	1	1	0	0	0	0	=>	0	1	1
1	1	1	1	0	0	0		1	0	0
1	1	1	1	1	0	0		1	0	1
1	1	1	1	1	1	0		1	1	0
1	1	1	1	1	1	1		1	1	1

Tab. 20 – Pravdivostní tabulka kodéru



Obr. 12 – Kodér

Funkcí kodéru je zredukovat počet vodičů potřebných k přenesení digitální informace. V našem případě se tento počet redukoval na tři vodiče. Jednotka pak podle těchto signálů zjistí, jaký výkon je právě používán a jakému stupni trakce odpovídá.

Aktuální pozice vozidla je realizována GPS přijímačem, který zabezpečuje sledování polohy a zároveň přesný čas.

V každé iteraci zápisu informací z čidel, kde se zapisuje v souřadnicovém systému poloha jednotky ve tvaru: Loc: 49°55'26.956"N, 14°38'3.672"E, dále hodnota z čidla měření množství paliva ve tvaru: Pal: 0666 l, dále hodnota z čidla měření aktuální spotřeby elektrické energie ve tvaru: Spo: 760,4 Kw, dále také zařazený stupeň trakce ve tvaru T3 a další údaje které nejsou v práci uvedeny a které mohou záviset na lokomotivě, v níž je řešena implementace. Tyto data jsou v jednotce nashromážděna v textovém souboru s příponou txt. V tomto balíku jsou zároveň uloženy data o přesném čase pro každou iteraci, ve které měření proběhlo a informace o lokomotivě, na které měření probíhalo.

4.2. Implementace komprese

Zvolená metoda komprimace ZIP byla využita především kvůli její efektivitě na opakujících se datech stejného typu. Po uplynutí časového intervalu pro přenos, tj. 30 sekund se textový soubor zabalí metodou ZIP do archivu s označením balíku podle pořadového čísla balíku ve formátu: 05022010.7z. Použitý program pro archivaci 7-ZIP je volně šiřitelný a dostupný jak pro unixové systémy, tak pro systémy windows.

Aplikace 7 – ZIP v tomto případě využívá metody LZMA. LZMA je nástavba na LZ77 s využitím Markových řetězců. Nastavení algoritmu je následující: velikost slovníku = 2Mb, velikost slova = 32 bitů. Předpokládaná spotřeba paměti pro slovník při kompresi

je 27 MB a pro dekompresi 4MB. Na vzorku přenesených dat o objemu 4004 Bajtů vyšel po kompresi přenášený objem dat pouhých 740 Bajtů. Pro upřesnění: 1 Bajt = 8 bitů čili 32032 bitů se zkomprimovalo na 5920 bitů. Z toho nám vyplývá kompresní poměr 0,185. To nám sníží celkový objem přenesených dat více než 5x. Tento vynikající výsledek komprese je daný především tím, že se v kódovaném textu jednotlivá slova nebo jejich části opakují v každé iteraci měření. Každá z měřených veličin se mění v malém počtu cifer a popis veličin zůstává stejný. Proto je využití slovníku při kódování tolik účinné. Ještě lepšího výsledku dosáhneme při kompresi celého 24hodinového logu uloženého v jednotce. Zvláště došlo li v průběhu logování k doplnění měřeného paliva v nádržích, nebo jede li vlak stejnou trasu vícekrát během cyklu logování. V takovém případě se opakují nejen části slov komprimovaného textu, ale celá slova. Na vzorku dat z vlaku, který během 24 hodin doplnil palivo a jel 3x stejnou trasu se ukázalo, že výsledný kompresní poměr dosáhl hodnoty 0,147. Objem přenesených dat se snížil z původních 11356312 bajtů na 1669376 bajtů. Výsledek je tedy více než 6x menší objem přenášených dat. Čas spotřebovaný pro kompresi a dekompresi takového souboru dat je více než desetinásobně kratší oproti intervalu vzorkování hodnot z měřících čidel což nám dovoluje tuto metodu používat. Pokud by byla doba potřebná pro kompresi delší, než je interval pořizování vzorků, muselo by se přistoupit na rychlejší metody komprese za cenu zhoršení kompresního poměru. Druhou možností by bylo prodloužit interval odebírání vzorků, ovšem za cenu zhoršení vypovídacích možností těchto pořízených dat. Na straně dispečinku jsou pak pořízená data archivována pro měsíční statistiky v komprimovaném stavu a před použitím programu pro vyhodnocování jsou tato data dekomprimována. Z principu kompresního algoritmu LZ77 vyplývá, že pro dispečink by bylo mnohem výhodnější si pro zvolený měsíční interval vyhodnocování pořízených dat tato data uchovávat v jednom komprimovaném archivu. Například ke konci každého měsíčního intervalu dekomprimovat všechny pořízené logy z konkrétní jednotky, sloučit je do jednoho textového řetězce a následně opět komprimovat stejnou metodou LZMA. Výsledek bohužel není možné prezentovat, neboť nejsou dostupná data z provozu jednoho kolejového vozidla v intervalu celého měsíce.

4.3. Implementace kryptografie

K přenosu mezi jednotkou a dispečinkem se používá síť GSM a služba GPRS. Služba GPRS má v sobě implementovaný kryptografický algoritmus A5 proti odposlechům. Dodnes používaný algoritmus A5/3 který zabezpečuje i informace o samotném přenosu je z hlediska odposlechu nedokonalý a zastaralý. Je známo několik úspěšných případů odposlechu a následném úspěšném provedení kryptoanalýzy a to jak při odposlechu mezi uživatelem a BTS (Base Transceiver Station) tak při odposlechu přímo v ústředně. Proto byl zvolen dodatečný kryptografický algoritmus TDES pro zabezpečení přenosu nezávisle na mobilních operátorech a na samotné síti GSM.

Použitý algoritmus TDES využívá 168 bitového klíče a je realizován samostatným modulem spolupracujícím s hlavním procesorem jednotky. Výpočetně náročné operace provádí výkonnější procesor jednotky, zatímco mikroprocesor šifrovacího zařízení provádí méně náročné operace s malým datovým tokem, avšak podstatné z hlediska principu šifrování. Tím je dosaženo vysoké rychlosti šifrování. Modul rozpoznává komunikaci šifrovanou i nešifrovanou a umožňuje tak krom realizovaného bezpečného spojení i běžné použití GPRS. Cestu distribuce klíče zde z pochopitelných důvodů nemohu uvést.

Výsledkem je nezávislá ochrana kryptografickým algoritmem TDES. Zkouška prolomení algoritmu při odposlechu trvala týden a výsledek byl zcela uspokojivý. Nepodařilo se šifru prolomit. V případě že by se to při delším prolamování podařilo výsledek by byl i nadále uspokojivý, neboť při implementaci tohoto řešení nám postačuje utajení přenášených dat po dobu 24 hodin.

4.4. Implementace bezpečnosti dat z pohledu konzistence

Pro přenos dat pomocí GPRS se v rámci GPRS používají 4 základní kódovací schémata. Jsou to CS1 – CS4. Každé schéma používá jinou samoopravnou metodu kódování. Jednotlivá schémata jsou volena podle síly signálu a s tím související kvalitu přenosu a odolnosti přenosu proti vzniku chyby. Se zhoršující se sílou signálu se postupně přechází od schématu s nejjednodušší metodou samoopravného kódování, která poskytuje jen slabou samoopravovací schopnost, ke složitějším metodám, které opravují větší množství chyb a jsou z hlediska konzistence přenášených dat bezpečné. Důvodem proč využívat v místech se silným signálem jednodušší schémata je vyšší datová propustnost. Jinak řečeno, v místech se silným signálem je menší pravděpodobnost vzniku chyby a tím,

že se omezení počet redundantních bitů realizujících samoopravnou schopnost, nám zbude více volných bitů pro přenos dat. Přenosová rychlost u GPRS dosahuje až 115kbps a teoreticky může dosáhnout hodnoty až 160 kbps. Ovšem obsluha GPRS je v hierarchii obsluhy přenosů až na posledním místě a proto se této rychlosti zpravidla nedosahuje.

Jelikož se obecně GPRS nepovažuje za spolehlivý přenos, jeho stávající zabezpečení se rozšíří o Hammingův (7,4) samoopravný kód. Ten je realizován programovou aplikací v rámci vlastní distribuce Linuxu. Na straně jednotky se zpráva obohatí podle Hammingova schématu o paritní bity a po přijetí na straně dispečinku se zkontroluje konzistence. Podle výsledku se data buď opraví, nebo se pošle požadavek o znovu odeslání správných dat. Na straně dispečinku by mohla být bezpečnost posílena kontrolou logiky naměřených hodnot. V případě přenosu chyby v hodnotě aktuálního stavu pohonných hmot v nádrži, kterou by neodchytil ani jeden ze zabezpečovacích systémů, by chybu mohl odhalit prohledávající software, který by kontroloval logickou posloupnost naměřených hodnot. Například pokud se v předešlé iteraci přijala naměřená hodnota pohonných hmot 500 litrů, v následující iteraci dojde k chybě a přijatá hodnota je 399litrů a v následující iteraci je opět správná hodnota 499 litrů, je patrné že prostřední hodnota je chybná a může se zažádat o znovu zaslání dat. A protože před odesláním jsou data nejprve zkomprimována, pak nám slovník jednu chybu roznese do dalších míst zprávy a chyba v konzistenci dat bude o to více zřejmá. Další možností kontroly je preventivně stahovat kompletní denní logy a ty porovnávat s dílčími přijatými zprávami.

5. Diskuse výsledků zvoleného řešení

Zvolené řešení využití algoritmů pro kompresi ZIP se ukázalo jako vysoce efektivní. Ušetřený objem dat na zkušebním vzorku byl více než čtyřnásobný. Zadavateli bylo navrhnuto využití této metody komprimování na měsíční zápisy dat, které zaručeně dosáhne ještě lepších výsledků než kódování menších balíků dat představujících půlminutový provoz vozidla. Kryptografický algoritmus TDES prošel testem kryptoanalýzy a splnil dané požadavky bez chyby. Byla by zde možnost využít i jiných kryptografických systémů a algoritmů. Přineslo by to však další náklady a výsledek by zůstal stejný. Pro zadání udržet informaci v tajnosti po dobu 24 hodin bohatě postačuje stávající použití kryptografie. Řešení bezpečnosti konzistence dat sice splňuje požadavky dané poptávkou ČD, ale není stoprocentní. Použití technologie GPRS je vhodné zejména kvůli jeho jednoduchosti a především kvůli pokrytí signálem GPS. GPRS není dokonalý a ani dodatečné zabezpečení konzistence nezajistí stoprocentní přenos informace, ale výhody, které přináší využití GPRS z hlediska nákladů jsou tak veliké, že nemá smysl řešit přenos jinými technologiemi.

Implementace měření spotřeby elektrické energie na kolejových vozidlech dopadla úspěšně, zákazník ČD je s řešením spokojen a nadále jej rozšiřuje na svá ostatní kolejová vozidla. Ke stávajícímu základnímu řešení bylo již dodáno několik vylepšení a nástaveb jako aktualizace firmware v jednotce, přenos balíku dat mezi jednotkou a řídícím systémem lokomotivy, komunikace s řídícím systémem lokomotivy na dálku a mnoho dalších funkcí. Základní řešení je použitelné na většinu kolejových vozidel bez ohledu na typ vozidla a typ měřících čidel použitých ve vozidle.

6. Závěr

V první části práce byl čtenář seznámen se základními principy a způsoby kódování dat. Kódy byly rozděleny do tří základních skupin a z obecného hlediska byl vysvětlen jejich princip. Na vybraných typech kódu byl názorně předveden postup kódování a zhodnocené výsledky kódů v rámci skupiny byly mezi sebou porovnány. Vytvořil se tak ucelený základ problematiky kódování dat, který by mohl sloužit jako materiál pro edukační účely na středních a vysokých školách.

V druhé části práce byla popsána implementace vybraných zástupců kódů z každé skupiny do projektu telemetrie kolejových vozidel. Telemetrie je v současné době rychle se rozvíjející odvětví a vzhledem ke zvyšujícím se nárokům na přenos dat se dají očekávat novinky v technologiích, které budou výzvou pro společnosti zabývající se touto problematikou. Implementace probíhala půl roku před vyhotovením této práce a po půlročním provozu byla přijata jako vyhovující. Po pilotním projektu následovaly další projekty ze stejné oblasti telemetricky a výše popsané řešení bylo zdokonalováno a využito i na jiných vozidlech a zařízeních jak v české republice tak v zahraničí.

Při sestavování teoretické části práce jsem vycházel z níže uvedených zdrojů a z vlastního studia informačních technologií. Práci jsem napsal jako zaměstnanec společnosti, v níž jsem spolupracoval na popsané implementaci kódování dat a zavedení do ostrého provozu. Vzhledem k závazku mlčenlivosti jsem nemohl uveřejnit některé konkrétní informace. I přes to byl obecný popis implementace sepsán srozumitelně a názorně tak, aby si čtenář mohl udělat představu o tom, jak se využívá kódování dat v praxi.

7. Seznam použitých zdrojů

[1] DEFATTA, D. J., LUCAS, J. G., HODGKISS, W. S. Digital Signal Processing - A System Design Approach, 1st. ed. John Wiley Sons, New York, 1988.

[2] KOTULIAKOVÁ, J., ROZINAJ, G. Číslicové spracovanie signálov. 1. vyd. Bratislava: FEI STU, 1999.

[3] MIHALÍK, J., GLADIŠOVÁ, I. Číslicové spracovanie signálov (Cvičenia). 1. vyd. Bratislava: Alfa, 1989.

[4] MIHALÍK, J. Číslicové spracovanie signálov. 1. vyd. Bratislava: Alfa, 1987.

[5] OPPENHEIM, A. V., SCHAFER, R. W. Digital Signal Processing, 1st. ed. Prentice-Hall, Inc. 1975.

[6] PROAKIS, J., MANOLAKIS, D. Digital signal processing. Principles, algorithms, and applications. 1st. ed. Prentice Hall, 1996.

[7] PROAKIS, J. G., MANOLAKIS, D. G. Introduction to Digital Signal Processing, 1st. ed. Macmillan Publishing Company, New York, 1988.

[8] Cohen,A.: Wavelets and multiscale Signal processing, Champan&Hall, 1995.

[9] Navara, M.: Pravděpodobnost a matematická statistika. Skriptum ČVUT, Praha, 1. vydání, 2007.

[10] Komprese dat [on line], © 2010, [cit. 2010-16-08].
URL: <http://voho.cz/wiki/komprese-dat/>

[11] Skripta T. Kaisera ze ZČÚ v Plzni. [on line], © 2010, [cit. 2010-20-08]. URL: <http://www.karlin.mff.cuni.cz/~stovicek/index.php/cs/0910zs-nmib004>

[12] Metody kódování [on line], © 2010, [cit. 2010-16-08].
URL: <http://www.uai.fme.vutbr.cz/~matousek/TIK/index_tik.html>

[13] Metody komprese dat [on line], © 2010, [cit. 2010-16-08].
URL: <http://www.ms.mff.cuni.cz/~malej9am/doc/komprese.php?lang=en>

[14] Kryptografie [on line], © 2010, [cit. 2010-16-08].
URL: <http://bobhy.wz.cz/clanky/kryptografie.html>

[15] Kryptografie [on line], © 2010, [cit. 2010-16-08]
URL: <http://www.kryptografie.wz.cz/data/des.html >

[16] Michal Kejval, A/D konverze a digitalizace dat, 2007.

8. Přílohy

8.1. Seznam obrázků:

8.2. Seznam tabulek: