Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Mobilní správce hesel

Bc. Tomáš Málek

Diplomová práce

2011

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš Málek**
Osobní číslo:	**I09371**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Mobilní správce hesel**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V úvodu práce bude popsána technologie zvolená pro vývoj mobilní aplikace, možnosti vývoje pro mobilní OS Symbian včetně nutného programového vybavení pro vývoj SW. Dále budou popsány principy šifrování včetně vybraných šifrovacích algoritmů. Budou popsána možná rozhraní pro komunikaci mezi PC a mobilním zařízením a navrženy datové struktury pro ukládání hesel. Primárním cílem bude zhodnotit a nakonec vybrat vhodný šifrovací algoritmus (DES, AES, IDEA, RSA atd.) z hlediska bezpečnosti kódování i výkonu mobilního zařízení. Dále vytvořit aplikaci pro uchovávání a správu hesel v zakódované formě a to na PC i na mobilním zařízení. A nakonec zprovoznit synchronizaci šifrovaných údajů mezi zařízeními.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

TOPLEY, Kim. J2ME v kostce : Pohotová referenční příručka. 1. vyd. Praha : Grada Publishing, 2004. 519 s. ISBN 80-247-0426-9. SINGH, Simon; KOUBSKÝ, Petr; ECKHARDTOVÁ, Dita. Kniha kódů a šifer : tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v českém jazyce. Dokořán : Argo, 2009. 382 s. ISBN 978-80-7363-268-7. GROŠEK, Otakar; PORUBSKÝ, Štefan. Šifrovanie - algoritmy, metódy, prax. Praha : Grada, 1992. 268 s. ISBN 80-85424-62-2.

Vedoucí diplomové práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání diplomové práce: **27. října 2010**

Termín odevzdání diplomové práce: **20. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

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

V Pardubicích dne 3. listopadu 2010

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární zdroje a informace, které jsem při jejím zpracování využil, jsou uvedeny na konci v seznamu použité literatury.

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

Ve Ždírci nad Doubravou 29. 8. 2011

Tomáš Málek

Anotace

V teoretické části jsou popsány některé šifrovací algoritmy, jejich rozdělení a vlastnosti. Je zde popsána technologie J2ME použitá pro vývoj mobilní aplikace, základní vlastnosti a omezení. V praktické části jsou uvedené charakteristické vlastnosti a vhodný šifrovací algoritmus použity při realizaci aplikace.

Klíčová slova

Mobilní aplikace; OS Symbian S60; Java ME; Java; Šifrovací algoritmus

Title

Mobile password manager

Annotation

The theoretical part describes some encryption algorithms, their distribution and properties. It describes technology used for the development of J2ME mobile applications, basic properties and limitations. In the practical part of the listed characteristics and suitable encryption algorithm used in the implementation of application.

Keywords

Mobile application; OS Symbian S60; Java ME; Java; Encryption algorithm

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Advanced RISC Machine
AWT	Abstract Windows Toolkit
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
DES	Data Encryption Algorithm
GUI	Graphic User Interface
IDE	Integrated Development Environment
IT	Information Technology
J2ME	Java Micro Edition
J2SE	Java Standart Edition
JAD	JAva Decompiler
JAR	Java Archive
JCP	Java Comunity Process
JDK	Java Development Kit
JRE	Java Runtime Environment
JSR	Java Specification Request
JTWI	Java Technology for the Wireless Industry
KVM	KiloByte Virtual Machine
MIDP	Mobile Information Device Profile
MS	MicroSoft
MSA	Mobile Services Architecture
OS	Operating System

OT	Otevřený text
PBP	Personal Basic Profile
PC	Personal Computer
PDA	Personal Digital Assistant
PHP	PHP: Hypertext Preprocesor
PIN	Personal Identification Number
PP	Personal Profile
RISC	Reduced Instruction Set Computer
RMI	Remote Method Invocation
RMS	Record Management System
SD	Secure Digital
SDK	Software Development Kit
SP2	Service Pack 2
ŠT	Šifrovaný text
USB	Universal Seriál Bus
VM	Vitrual Machine
XML	Extensible Markup Language
XOR	Exclusive OR

1 Úvod

Informační technologie se v moderní době staly součástí každodenního života většiny obyvatel. S jejich rozvojem však vznikly problémy s jejich bezpečností. Proto je kladen velký důraz zejména na zabezpečení uživatelských účtů, schránek elektronické pošty, komunikačních programů a podobně. S IT souvisí i problematika vstupních dat využívaných jako heslo ke vkladní knížce, kódy k nejrůznějším elektronickým bezpečnostním systémům, alarmům, domácímu sejfu či jen PIN k mobilnímu telefonu. Hackerské útoky i jiné druhy hrozeb napadení soukromí člověka vyžadují stále složitější hesla, která si lidský mozek jenom s velkými obtížemi dokáže zapamatovat. Jedním z možných řešení předejití zapomenutí hesla je jeho uchování pomocí mobilní aplikace.

Hlavním cílem této práce je právě návrh a následná realizace mobilní aplikace sloužící pro uchování hesel k různým službám a systémům.

V úvodní části práce jsou teoreticky popsány šifrovací algoritmy, jejich rozdělení a základní vlastnosti (včetně bližších popisů některých z nich). Dále jsou uvedeny základní vlastnosti a možnosti technologie Java ME pro vývoj mobilní aplikace. Je zmíněn i mobilní OS (včetně základních vlastností), pro který je mobilní aplikace určena. V neposlední řadě je popsáno potřebné softwarové vybavení nutné pro vývoj aplikací.

V následující praktické části jsou podrobně navrženy hlavní požadované funkcionality aplikace včetně detailního popisu jednotlivých funkcí. Následně je uveden postup realizace mobilní aplikace, využité možnosti a implementované funkce. Dále je zde také popsán návrh aplikace určené pro PC sloužící k synchronizaci dat s mobilní aplikací a způsob řešení synchronizace samotné.

V závěrečné části práce jsou shrnuty a zhodnoceny výsledky realizovaných aplikací včetně popisu nedostatků, doporučení k jejich odstranění a návrhů na další vylepšení.

2 Šifrovací algoritmy

Šifrováním, šifrovacími a kódovacími algoritmy se zabývá matematický vědní obor zvaný **kryptologie**. Tento obor se dělí na dvě velké podskupiny:

- **kryptografie** - zabývá se návrhem šifrovacích algoritmů
- **kryptoanalýza** - snaží se šifrovací algoritmy prolomit

Šifrovací algoritmus

Šifrovací algoritmus je algoritmus, který se snaží utajit data jejich zašifrováním. K této činnosti používá tajný šifrovací klíč. Více lidí může pro šifrování použít stejný šifrovací algoritmus, ovšem podmínkou pro šifrování dat je, že každý musí mít pro šifrování a dešifrování individuální šifrovací klíč.

Kódovací algoritmus

Kódovací algoritmy jsou algoritmy vytvářené za stejným účelem jako algoritmy šifrovací. Snaží se chránit data před neoprávněným přečtením. Do procesu kódování ovšem v tomto případě nevstupuje žádný tajný klíč, ale o utajení se stará vlastní algoritmus. Pokud by stejný algoritmus používalo více lidí, budou si moci vzájemně prohlížet utajená data [4]. Jako příklad lze uvést nejrůznější kódové knihy, při jejichž použití musí komunikující protistrany disponovat překladovou bází slov a kódů (tzv. slovníkem).

Prolomený algoritmus

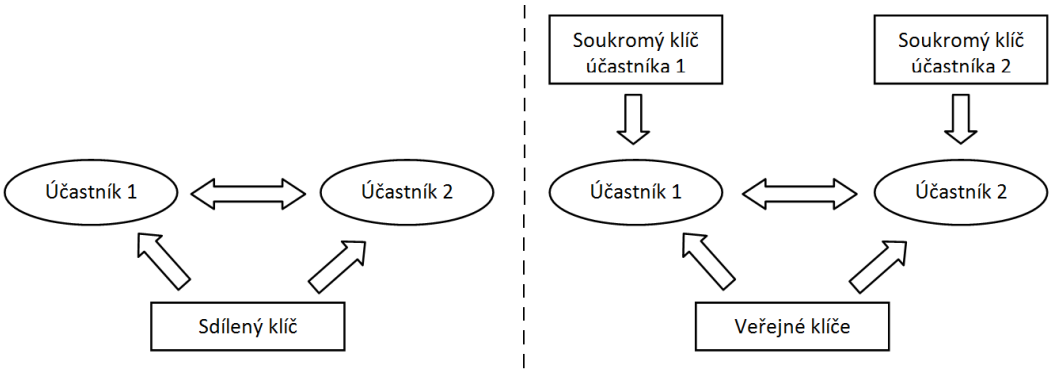
Šifrovaná data jsou většinou data důvěrného typu. Proto je důležité, aby data byla algoritmem důkladně zabezpečena a nebyla čitelná pro neoprávněnou osobu. Algoritmus lze označit jako prolomený, pokud lze přečíst chráněná data bez znalosti šifrovacího klíče nebo kódovacího algoritmu [4].

2.1 Symetrické a asymetrické algoritmy

Základním rozdělením šifrovacích algoritmů je dělení do dvou skupin:

- **symetrické algoritmy**
- **asymetrické algoritmy**

Principem symetrického algoritmu je využití pouze jednoho klíče k šifrování i dešifrování dat. Asymetrický algoritmus využívá klíčů dvou (veřejný a soukromý klíč) [17]. Z obrázku 1 jsou patrné rozdíly mezi použitím symetrických algoritmů (vlevo) a asymetrických algoritmů.

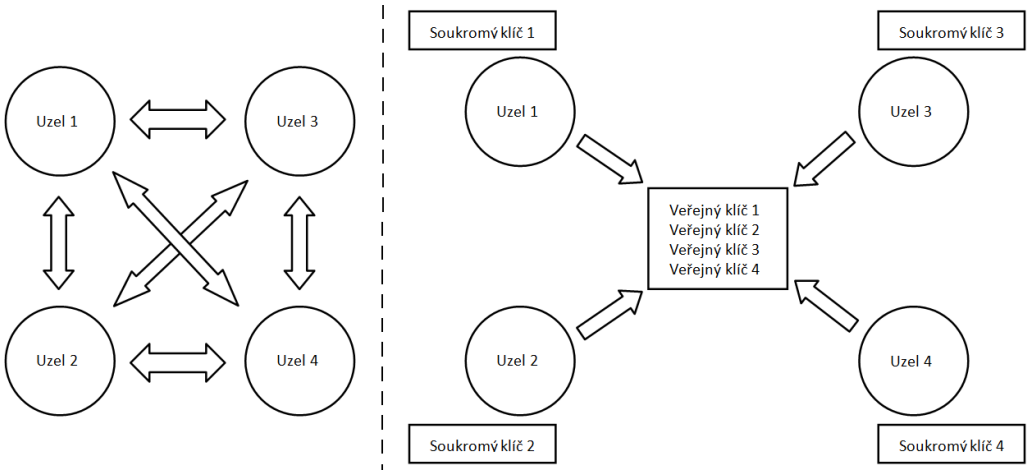


Obrázek 1 - Rozdíl mezi symetrickými a asymetrickým šifrovacími algoritmy [4]

2.1.1 Symetrické šifrovací algoritmy

Ve svých počátcích byly šifrovací algoritmy založeny pouze na symetrickém klíči. Využití symetrického šifrovacího algoritmu v praxi znamená, že pokud chtějí dva účastníci bezpečně komunikovat, domluví se bezpečným způsobem na tajném klíči. Utajenost klíče je zde velice důležitá a nesmí ho znát nikdo další [17]. Samotná zpráva (kterou si posílají) je stejným klíčem šifrována i dešifrována. Pokud by však tito dva potenciální účastníci chtěli šifrovaně komunikovat ještě s někým třetím, jsou v systému již tři klíče. Pro čtyři účastníky je klíčů šest a pro pět (vzájemně komunikujících účastníků) dokonce deset [4]. Počet klíčů tedy neúměrně roste s rostoucím počtem členů komunikující skupiny a jejich správa (bezpečné ukládání) se stává neúnosnou.

Obrázek 2 graficky demonstruje počty klíčů v případě využití symetrických a asymetrických šifrovacích algoritmů (na případě se čtyřmi uživatelskými uzly).



Obrázek 2 - Počty klíčů symetrické a asymetrické kryptografie [4]

2.1.2 Asymetrické šifrovací algoritmy

Problém symetrických algoritmů spočívá v neúměrně rostoucím množství klíčů. Proto byla vyvinuta koncepce tzv. asymetrických algoritmů, které problém s velkým počtem klíčů řeší a jejichž matematické základy jsou zcela odlišné. Každý z komunikujících účastníků si vytvoří dvojici klíčů (pár klíčů) [4]. Jeden z nich je jeho privátní a uchovává si ho v tajnosti. Druhý klíč je veřejný. Tento klíč rozdá všem, se kterými má zájem komunikovat (nahraje jej na veřejný server s klíči nebo zvolí jinou cestu jak ho zveřejnit). Pokud dotyčnému chce jiný účastník poslat zprávu, získá jeho veřejný klíč a zašifruje s ním tajnou zprávu. Zašle ji adresátovi, který k jejímu dešifrování použije svůj soukromý klíč [4]. Díky vlastnostem algoritmu je zajištěno, že zpráva zašifrovaná jedním klíčem je dešifrovatelná pouze klíčem druhým a nikoliv tím, kterým byla zašifrována. Je úplně jedno, který ze dvojice klíčů se k šifrování či dešifrování použije. Hlavní výhodou asymetrického šifrování je jednodušší správa klíčů. Každý uživatel se stará o bezpečné uchování pouze jednoho klíče. Počet klíčů v systému roste lineárně a je pouhým dvojnásobkem počtu komunikujících účastníků.

2.1.3 Rozdíly mezi symetrickými a asymetrickými algoritmy

Vzhledem k matematickému řešení jednotlivých algoritmů platí určitá pravidla. Symetrické algoritmy jsou mnohem rychlejší než asymetrické. Rozdíl se však uplatňuje (díky neustále rostoucím výkonům počítačů) až při šifrování většího objemu dat [4]. Dalším velkým rozdílem mezi uvedenými typy algoritmů je délka klíče nutná k zajištění bezpečnosti. Symetrické algoritmy si např. vystačí pouze se 128 bitovým šifrovacím klíčem, ale asymetrické algoritmy potřebují (pro zajištění zabezpečení na podobné úrovni) klíč s minimální délkou 1024 bitů [4].

Často se objevují tzv. hybridní algoritmy. Jejich princip je založen na kombinaci obou zmíněných metod. Před vlastním šifrováním je vytvořeno náhodné číslo (označováno jako relační symetrický šifrovací klíč), které je určeno pro jedinou operaci. Otevřený text (nešifrovaný běžně čitelný text) je tímto číslem zašifrován velice rychle a kvalitně [4].

Tento relační klíč je zašifrován pomocí asymetrického algoritmu. Díky malé délce klíče trvá šifrování klíče pouze zlomek času, který je vynaložen na šifrování dat. Zašifrovaný symetrický klíč je přiložen ke zprávě. Účastník na druhé straně pomocí asymetrického klíče dešifruje přiložený symetrický relační klíč. Relační klíč poté použije k dešifrování vlastního textu. Zajímavostí je, že relační symetrický klíč je po provedení kryptografického úkonu zlikvidován. V systému jsou tedy stále udržovány pouze asymetrické klíče v počtu dvojnásobku komunikujících uživatelů. Díky hybridním algoritmům byl tedy odstraněn nejen problém s mnoha klíči (kterým trpí symetrické algoritmy), ale i přílišná časová náročnost (typická pro algoritmy asymetrické). Navíc tím nebyla nijak poznamenána bezpečnost systému [4].

2.2 Proudové a blokové šifry

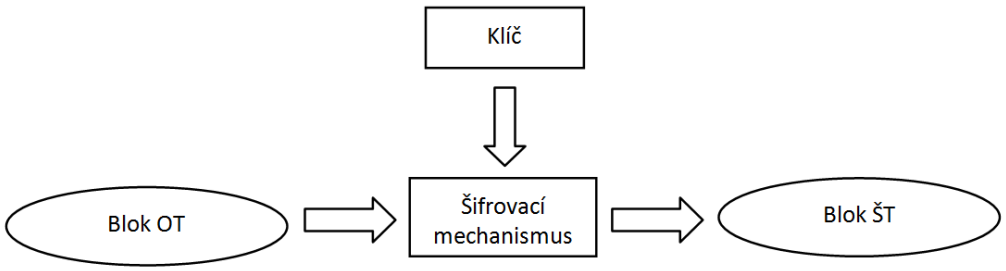
Z hlediska rozdělení šifrovacích algoritmů byly definovány i jiné pohledy než jen dělení na symetrické a asymetrické. Dalším způsobem klasifikace algoritmů je rozčlenění podle množství dat, která jsou v jednom časovém úseku šifrována. Pokud probíhá šifrování otevřeného textu po znacích, jedná se o **proudové šifry**. Je-li naopak otevřený text šifrován po větších blocích, jedná se o **šifry blokové**. Z uvedených definic vyplývá, že v jistém okamžiku dochází k prolnutí obou termínů. Velikost znaku lze modifikovat v poměrně velkém rozsahu (od 8 bitů až po 64 bitové bloky). Dále se také mohou objevit případy, kdy jsou data šifrována po jednotlivých bitech. Mohou nastat i extrémy, kdy délka bloku klesne na jediný znak [17].

2.2.1 Módy činnosti blokových šifer

Blokové šifrovací algoritmy šifrují otevřený text po blocích (menších oddílech). Ukázalo se ovšem, že takové šifrování může být poměrně snadno prolomitelné. Bylo proto definováno pět základních módů činnosti blokových šifrovacích algoritmů, které se liší umístěním šifrovacího mechanismu [4].

Mód ECB (Electronic Cipher Book)

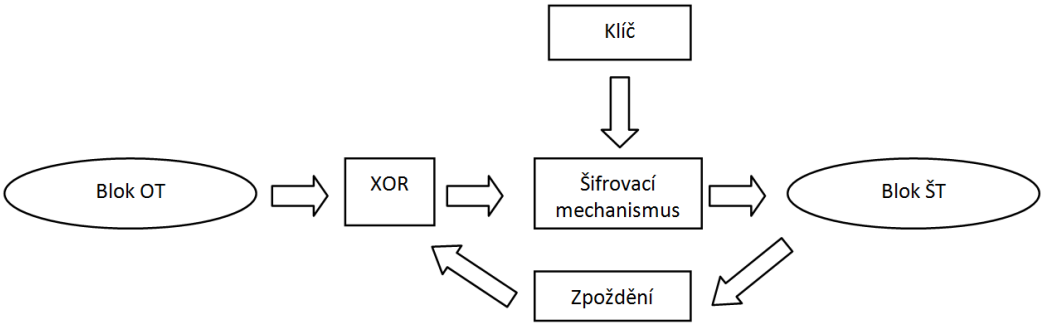
Jedná se o základní mód všech blokových šifrovacích algoritmů. Na vstup je přiveden blok otevřeného textu a na výstupu je text šifrovaný. Jestliže se na vstupu objeví totožný blok, výsledkem je opět stejný blok šifrovaný. Celou šifru lze nazvat tzv. překladovým slovníkem, který pracuje na principu nahrazení vstupního textu výstupním [17]. Slabou stránkou je, že bloky jsou šifrovány nezávisle na sobě. Příjemce tak nepozná, že byl blok eventuálně nahrazen blokem jiným [4]. Obrázek 3 graficky znázorňuje princip činnosti módu ECB.



Obrázek 3 - Schéma činnosti módu ECB [4]

Mód CBC (Cipher Block Chaining)

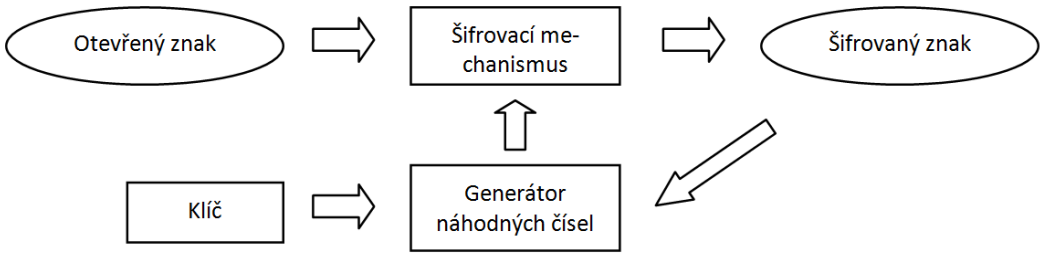
Oproti módu ECB se liší hlavně tím, že se snaží do blokové šifry zanést závislost mezi po sobě šifrovanými bloky. K vytvoření této závislosti byla použita logická nonekvivalence (XOR) [17]. Zašifrovaný blok textu je pomocí funkce XOR zkombinován s následujícím blokem otevřeného textu. Výsledek této operace vstupuje do šifrovacího algoritmu. Slabinou je stejné šifrování prvního bloku, na který je uplatněn pouze mód ECB. Kvůli omezení opakování úvodu zprávy se používá tzv. inicializační vektor. Obecně si lze pod tímto označením představit náhodné číslo, které je použito ke xorování prvního bloku otevřeného textu. Inicializační vektor musí znát obě komunikující strany. Nemusí být nijak šifrován a velice často bývá přikládán přímo ke zprávě [4]. Obrázek 4 graficky znázorňuje princip činnosti módu CBC.



Obrázek 4 - Schéma činnosti módu CBC [4]

Mód CFB (Cipher Feedback Mode)

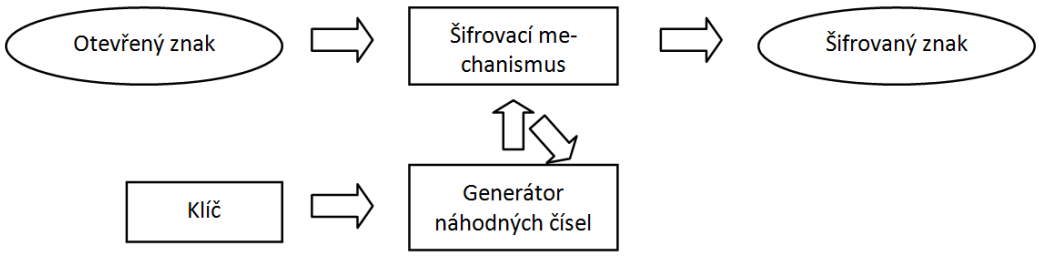
Tento mód přepíná blokovou šifru do proudového režimu. Aby nemohla nastat situace se stejným šifrováním stejného znaku, je do systému začleněn pseudonáhodný prvek. Vlastní data jsou šifrována pomocí posloupnosti pseudonáhodných čísel, jejichž generování je řízeno zpětnovazebním prvkem. Generátor pseudonáhodných čísel je řízen symboly šifrovaného textu. Vstupem zpětné vazby je konečný výstup šifrovacího systému [17].



Obrázek 5 - Schéma činnosti módu CFB [4]

Mód OFB (Output Feedback Mode)

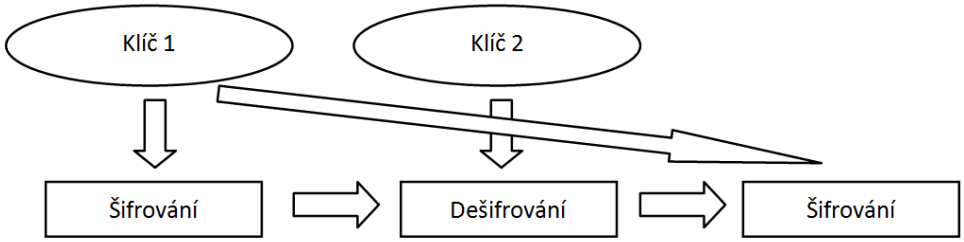
Hlavní odlišností módu OFB oproti CFB je umístění zpětné vazby, která je zařazena přímo na výstup generátoru pseudonáhodných čísel. Znamená to, že generátor je řízen svým vlastním výstupem [4].



Obrázek 6 - Schéma činnosti módu OFB [4]

Mód EDE (Encryption-Decryption-Encryption)

Pracuje na principu zapojení tří šifrovacích bloků za sebou. První a poslední blok šifrují data klíčem 1, prostřední blok naopak šifrovaná data dešifruje klíčem 2. Délka klíče je dvojnásobná. Kromě tohoto řešení existují i případy s využitím tří různých klíčů a výsledný klíč je délky trojnásobné. Příkladem využití tohoto módu zapojení je šifrovací algoritmus 3DES [17].



Obrázek 7 - Schéma činnosti módu EDE [4]

2.3 Symetrická kryptografie

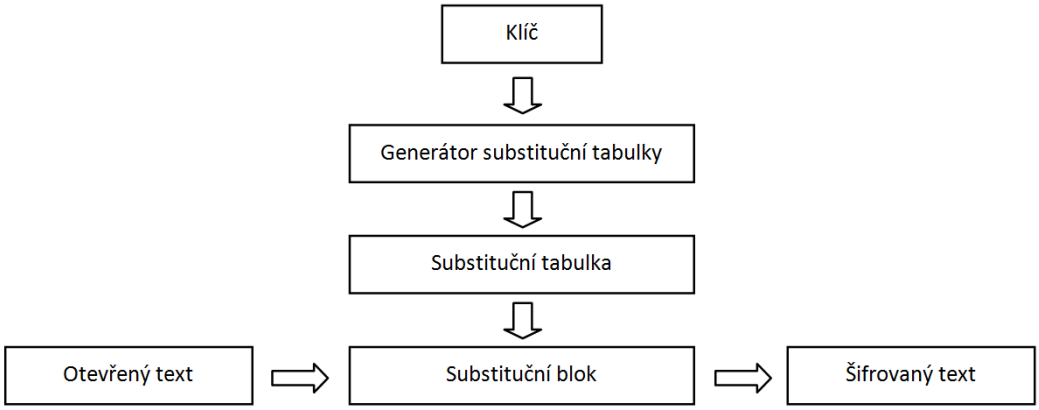
Symetrické šifrovací algoritmy pracují pouze s jedním klíčem, který musí být společný oběma komunikujícím účastníkům. Tento klíč se používá pro šifrování i dešifrování textu. Hlavní výhodou symetrických algoritmů je v porovnání s asymetrickými jejich velká rychlost. Nevýhodou je neúměrně rostoucí počet klíčů. Při tvorbě symetrických algoritmů se používají dva nejčastější přístupy:

- **substituce** - nahrazuje znak otevřeného textu znakem šifrovaného textu podle předem předepsaného klíče
- **transpozice** - zachová hodnotu znaku, ale změní jeho pozici v šifrovaném textu

Velká řada algoritmů přitom využívá pro svou práci kombinaci obou těchto přístupů [17].

2.3.1 Substituční šifry

Substituční šifra nahrazuje znaky otevřeného textu znaky jinými, čímž vznikne text šifrovaný. Přesná pravidla pro nahrazování znaků jsou dána šifrovacím klíčem a nahrazování je řízeno záznamy tzv. substituční tabulky [17].



Obrázek 8 - Princip činnosti substituční šifry [4]

Monoalfabetická substituční šifra

Monoalfabetická substituční šifra je založená na existenci statické tabulky (vytvořené podle šifrovacího klíče), kterou šifra používá k překladu znaku otevřeného textu na šifrovaný znak a obráceně. Jako typického představitele tohoto druhu šifer lze uvést historickou Caesarovu šifru (kde jsou znaky abecedy na druhém řádku posunuty o tři pozice oproti řádku prvnímu). Ze stejné skupiny je i šifra ROT3 hojně využívaná v Unixu (jako jednoduchý šifrovací algoritmus) provádějící posun o 13 pozic [4].

Homofonní substituční šifra

Hlavní nevýhodou monoalfabetické šifry je možnost použití frekvenční analýzy k odhadnutí znaků (nejčetněji využívaných v abecedách jazyků). Homofonní substituční šifra tuto nevýhodu řeší tím, že každý znak otevřeného textu lze šifrovat na několik různých znaků textu šifrovaného. Nicméně ani tyto šifry nemohou zcela zakrýt statistické četnosti využívání znaků abecedy daného jazyka [4].

Polygramová substituční šifra

Substituční tabulky v případě polygrafových substitučních šifer obsahují celé skupiny znaků. Například řetězec „DEF“ v otevřeném textu lze přeložit jako „UVW“ v textu šifrovaném apod. I v případě polygramové šifry jsou tabulky generovány na základě hodnot šifrovacího klíče. Jako příklad využití těchto šifer lze uvést algoritmus Playfair, který byl používán britskou armádou v první světové válce [4].

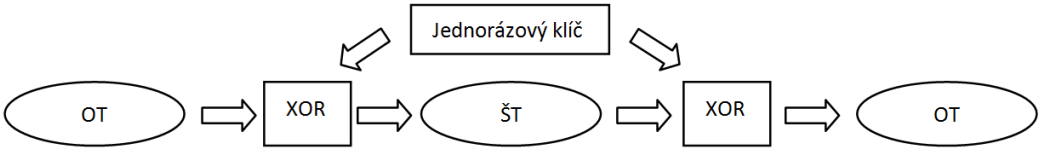
Polyalfabetická substituční šifra

Polyalfabetické neboli více abecední šifry jsou nejpokročilejšími představiteli skupiny substitučních šifer. Jedná se o skupinu monoalfabetických šifer, které jsou postupně použity na jednotlivé znaky otevřeného textu. První znak může být šifrován první monoalfabetickou šifrou, druhý druhou, třetí pomocí třetí a čtvrtý např. opět za pomoci první (protože počet šifer by neměl příliš narůstat). Základem je tabulka o rozměrech 26 x 26 znaků (takovýto rozsah tedy platí pro anglickou abecedu). Každý řádek i sloupec je oproti předchozímu posunut o znak. Zvolené heslo určí, které sloupce jsou při šifrování použity [4].

Nerozluštitelná šifra

Jedná se o tzv. jednorázový heslář. Platí zde pravidlo, že každý klíč lze použít jen jednou. Zároveň musí být naprosto náhodný a jeho délka musí být stejná jako délka otevřeného textu. Příkladem tohoto typu šifry je Vernamova šifra.

Podstatu Vernamovi šifry tvoří blok provádějící bitovou operaci XOR. Všechny znaky otevřeného textu jsou xorovány se znaky jednorázového klíče. Pro dešifrování je postup stejný. Jedinou slabinou je tedy klíč (jeho generování, distribuce a zlikvidování - nesmí se znovu použít ani padnout do nepovolaných rukou) [4].



Obrázek 9 - Princip činnosti Vernamovi šifry [4]

2.3.2 Transpoziční šifry

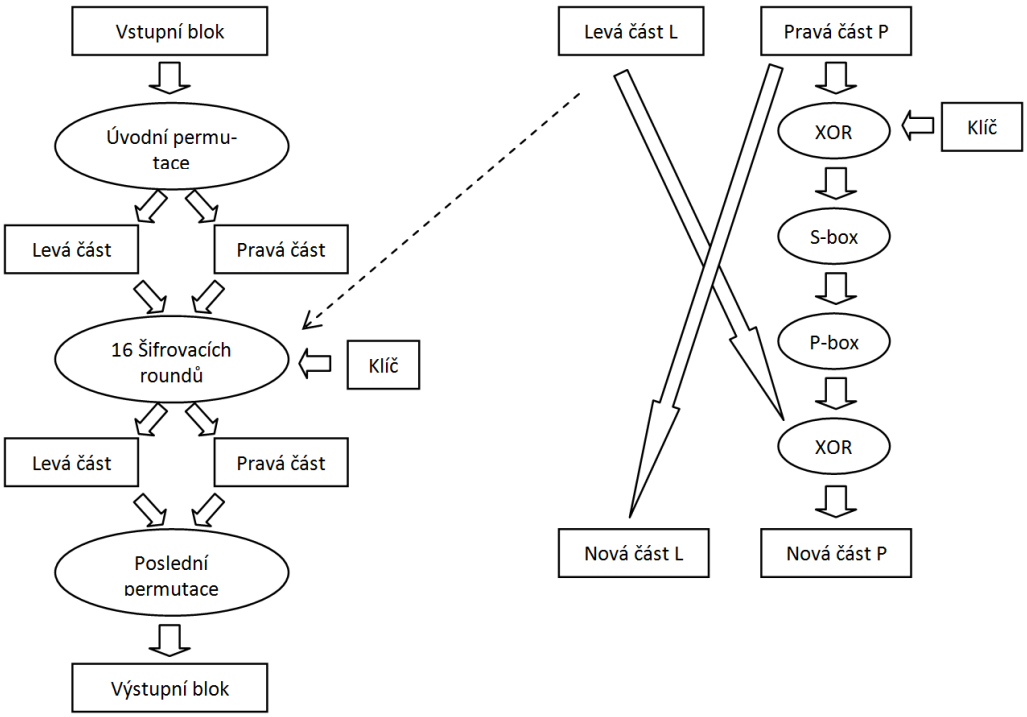
Hlavním rozdílem transpozičních šifer oproti šifrám substitučním je, že znaky nenahrazují jinými, ale pouze mění jejich pozici v textu. Protože se mění pozice znaků v textu, označuje se to někdy také jako permutace. Jako příklad jednoduché transpozice lze uvést text o sto znacích zadaný do matice 10 x 10 znaků po řádcích a její následné přepsání po sloupcích [4]. K transpozici lze využít jakéhokoliv permutačního algoritmu, který při použití v opačném směru dokáže rozházený text srovnat zpět do původní podoby. Výhodou transpozičních šifer je rozbití velkých struktur. Protože hodnoty znaků se nemění, ani frekvenční analýza nezabere. Pro prolomení algoritmu transpozice je nutné najít periodu, se kterou je text rozházen [17].

2.3.3 Šifrovací algoritmus DES

Algoritmus DES někdy též označován jako DEA (Data Encryption Algorithm) je dalším představitelem synchronních šifer. Původně byl zvolen jako norma pro účely šifrování dat amerických státních orgánů. Nyní je však z hlediska výkonů dnešních počítačů nevyhovující.

Jedná se o blokovou šifru pracující s bloky o délce 64 bitů. Stejné délky je i šifrovací klíč, jehož každý osmý bit je paritní. Paritní bity (nejméně významné bity v každém bytu klíče) jsou ve většině případů ignorovány. Samotná efektivní délka klíče je pouhých 56 bitů. Z hlediska generování klíčů je doporučeno vyhnout se klíčům evidentně slabým (složeným ze samých nul, nebo samých jedniček), které by mohly značně ulehčit kryptoanalýzu.

Algoritmus DES je kombinací transpozičního a substitučního typu šifry. Na blok otevřeného textu je nejdříve použita jednoduchá substituce. Následně se použije transpozice (u DES označována permutací). Tento krok s použitím šifrovacího klíče je proveden šestnáctkrát. Jeden tento cyklus se označuje pojmem round. Šifrování i dešifrování probíhá stejným postupem. Na samém začátku je provedena počáteční permutace. Potom je čtyřiašedesáti bitový blok rozdělen na poloviny. V dalším kroku je šestnáctkrát proveden round, ve kterém jsou data zkombinována s hodnotou klíče. Poté se obě poloviny dat spojí a provede se závěrečná permutace (inverzní k permutaci původní). Výsledkem algoritmu je šifrovaný text. V rámci jednoho roundu je nejdříve provedena substituce za pomoci tzv. S-boxů. S-boxy jsou přesně definované substituční tabulky [16]. Tyto tabulky by teoreticky mohly zapříčinit tzv. zadní vrátka v algoritmu, ale žádná provedená analýza to doposud nepotvrdila [4]. Následujícím krokem algoritmu je permutace provedená v P-boxu (opět pevně daná tabulka určující změnu pořadí bitů šifrovaného bloku). Tato funkce je aplikována pouze na pravou část bloku. Výstupní hodnota je následně xorována s částí klíče. Klíč je totiž v každém roundu rotován a je z něj vybráno jen 48 bitů. V dalším roundu jsou obě části prohozeny. Dosavadně zpracovávaná pravá část se vymění s levou a šifrování se provádí s původně levou částí [16].



Obrázek 10 - Princip činnosti algoritmu DES a znázornění jednoho roundu [4]

2.3.4 Šifrovací algoritmus AES

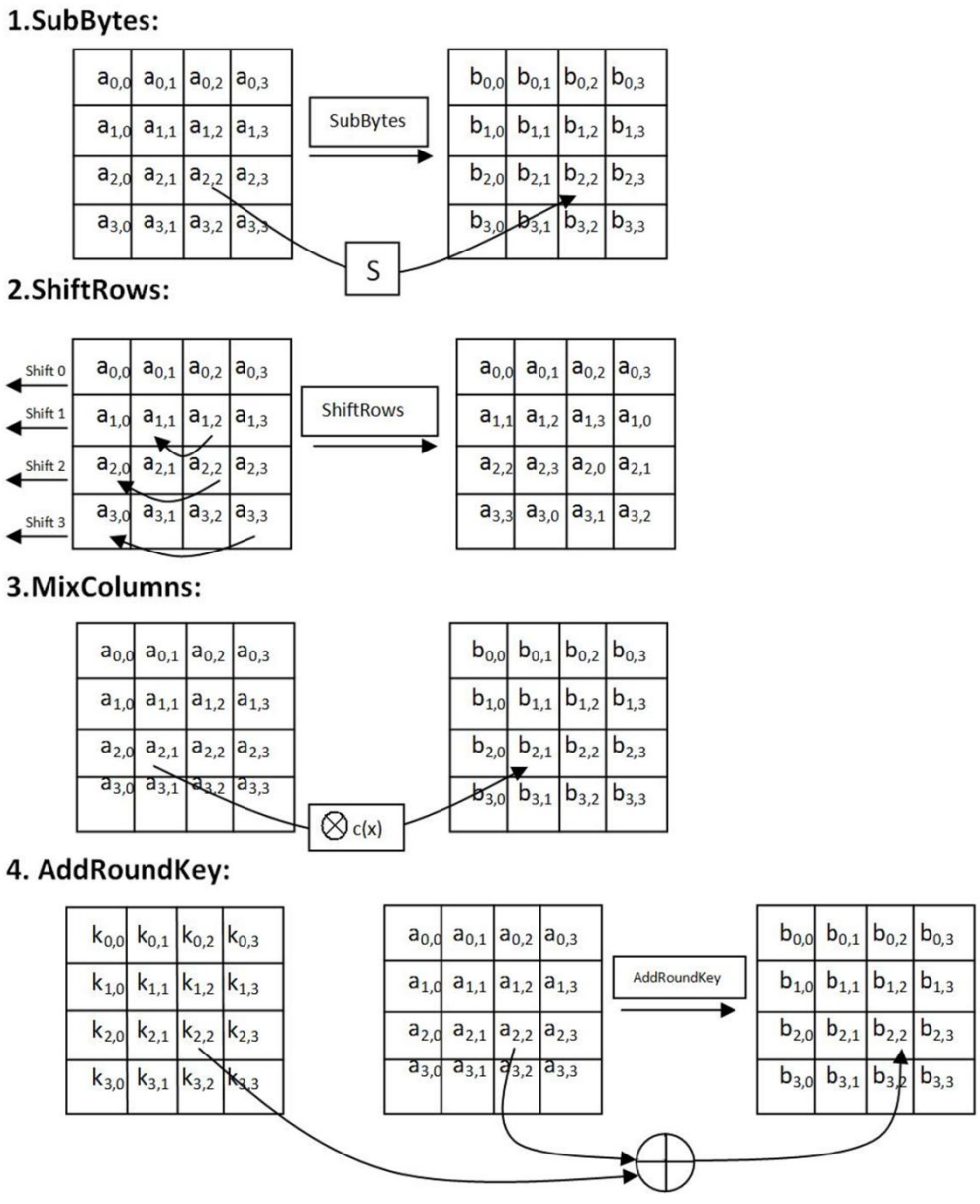
Následníkem algoritmu DES se stal algoritmus AES označovaný také jako Rijndael podle svých belgických autorů Vincenta Rijmena a Johna Daemena. Americké úřady však tento algoritmus označili jako AES (Advanced Encryption Standard). V praxi se používají oba názvy [17].

AES je bloková šifra s klíčem, který může nabývat hodnot 128, 192 a 256 bitů. Délka šifrovaného bloku je přitom ve všech případech stejná (128 bitů). Podle délky klíče se mění počet roundů. Pro 128 bitový klíč postačí deset roundů, pro 192 bitový dvanáct a pro 256 bitový je použito čtrnáct roundů [4].

V jednotlivých roundech je nejdříve provedena substituce podle S-boxů (které se od S-boxů u DES liší). Následují dva speciální transpoziční cykly. Blok je uspořádán do maticové struktury, kde jsou nejdříve rotovány řádky matice. V následujícím kroku jsou proházeny sloupce pomocí násobení speciální transpoziční maticí. V závěru jsou data zkombinována se šifrovacím klíčem. Šifrovací klíč se pro každý round mění [4].

AES je aplikován na data s pevně danou délkou. Pokud jsou šifrovaná data delší, jsou zpracovávána po blocích. Jestliže jsou data naopak kratší (zejména u posledního bloku, který obsahuje pouze zbytky dat), je blok dat doplněn na odpovídající délku. Toto doplnění se nazývá **padding**. Pro realizaci paddingu se používá několik algoritmů - od pouhého doplnění nulami až po složitější schémata [8].

Popis průběhu šifrování je znázorněn na obrázku 11.



Obrázek 11 - Princip šifrování algoritmem AES [8]

Popis kroků algoritmu AES:

1) **SubBytes** - provede jednoduchou substituci - původní bit je nahrazen jiným podle předem daného klíče (Rijndael S-Box). Zajišťuje nelineárnost šifry.
2) **ShiftRows** - posune řádky matice jak je znázorněno na obrázku 11.
3) **MixColumns** - prohází sloupce a každý z nich vynásobí polynomem c(x).
4) **AddRoundKey** - zkombinuje každý byt se subklíčem (který získá z klíče původního). Každý byt subklíče se zkombinuje s odpovídajícím bytem původní zprávy a na výstupu je výsledná šifra.

2.3.5 Další symetrické algoritmy

RC2

Jedná se o blokový šifrovací algoritmus, který pracuje se 64 bitovými bloky a proměnlivou délkou klíče. Jeho název je tvořen iniciály autora Rivesta Ciphera. Vyvinut byl ve firmě RSA, která ho tají [4].

RC4

Tento šifrovací algoritmus je proudový. Autorem je opět Rivest Cipher a vlastníkem ta samá firma jako u RC2. Jeho struktura byla roku 1994 odhalena poté, co se dostal na veřejnost [4].

RC6

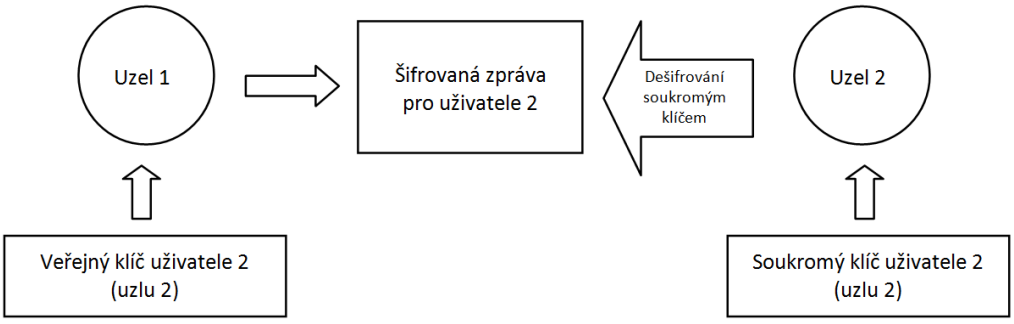
Nejnovější šifra z rodiny RC2 až RC6. Šifra má řadu volitelných parametrů. Lze definovat počet bitů slova, počet roundů a počet bytů klíče. Také proto bývá také někdy označována jako RC6-w/r/b [15].

IDEA

Blokový šifrovací algoritmus IDEA (International Data Encryption Algorithm) vytvořený ve Švýcarsku pracuje s blokem o délce 64 bitů. Šifrovací klíč je délky dvojnásobné [16].

2.4 Asymetrická kryptografie

Hlavním rozdílem asymetrických šifrovacích algoritmů oproti symetrickým, je existence dvou šifrovacích klíčů pro každého účastníka. Klíče navíc tvoří klíčový pár. Důležitou vlastností je, že text zašifrovaný jedním klíčem je dešifrovatelný pouze klíčem druhým. Šifrovaný text navíc nelze dešifrovat ani klíčem, který byl použit k jeho zašifrování. V případě páru klíčů nezáleží na tom, který z dvojice klíčů je použit k šifrování a který k dešifrování. Klíče páru se také označují jako veřejný a soukromý. Důležitým pravidlem je bezpečné uchování soukromého klíče. Veřejný klíč lze libovolně šířit mezi uživatele, se kterými má účastník zájem zabezpečeně komunikovat [4]. Grafické znázornění principu asymetrického šifrování je na obrázku 12.



Obrázek 12 - Princip asymetrického šifrování a dešifrování [4]

2.4.1 Algoritmus vah

Algoritmus vah slouží především k vysvětlení principu asymetrického šifrování. Princip je takový, že jsou dány pomyslné váhy se dvěma miskami. Na jedné je položeno břemeno o neudané hmotnosti. Na tu druhou je nutné naložit několik menších závaží, která váhy přesně vyváží. Úkol je jednoduchý, pokud je dostupná super-rostoucí posloupnost vah dodaných závažíček (každé těžší závaží má vyšší hmotnost nežli součet předchozích lehčích). Podobný princip využití super-rostoucích posloupností je aplikován i u asymetrické kryptografie. Jako soukromý klíč se nastaví podobná posloupnost jako u super-rostoucího závaží. Veřejným klíčem je posloupnost vah pro normální (nikoliv super-rostoucí) závaží. Kdokoliv tak může snadno určit, která závaží k vyvážení vah lze použít. Při šifrování se otevřený text rozdělí na bloky o stejné délce jako počet prvků posloupnosti závaží. Z binární reprezentace bloku je následně stanoveno, která závaží normálního břemene (veřejného klíče) se použijí. Součet těchto hodnot závaží je výsledným šifrovaným textem. Příjemce pomocí speciálních transformací převede získanou hmotnost na hmotnost odpovídajícího super-rostoucího závaží. Nakonec zjistí, která závaží byla použita a získá tak otevřený text [4].

2.4.2 Algoritmus RSA

Algoritmus RSA pojmenovaný po svých tvůrcích (Rivest, Shamir, Aleman) je matematicky velice vyspělý a doposud nepokořený algoritmus. Je založený na složitosti faktorizace velkých prvočísel [16].

Před samotným šifrováním je vygenerován klíčový pár. Pro tento účel jsou zavedena náhodná prvočísla `p` a `q`. Generování velkých prvočísel představuje problém (jedná se řádově o stovky cifer). Proto je tento úkol řešen tak, že se vygeneruje náhodné číslo, které je následně podrobeno testu prvočíselnosti. Ten spočívá v dělení generovaného čísla malými prvočísly. Tento úkon by měl (případně) odhalit, že číslo není prvočíslem. Dalším krokem algoritmu je součin čísel `p` a `q`, čímž je získáno číslo:

```
n = p * q
```

Veřejný klíč se generuje jako náhodné číslo e, které nemá společné součinitele s číslem daným vztahem:

$$(p-1) * (q-1)$$

Soukromý klíč je tvořen složitějším výpočtem:

$$d = e^{-1} \bmod ((p-1) * (q-1))$$

Veřejný klíč je tvořen dvojicí {n, e}. Soukromý klíč tvoří odpovídající dvojice {n, d}. Po vytvoření klíčů nejsou čísla p a q již dále využívána a je nutné je zlikvidovat, protože z nich lze klíče snadno určit. Šifrování a dešifrování je dáno vztahy:

$$ŠT = OT^{e} \bmod n$$

$$OT = ŠT^{d} \bmod n$$

Kde OT je zkratka pro otevřený text a ŠT pro šifrovaný. Protože u asymetrických algoritmů není pevně určeno k čemu který z klíčů použít, lze tyto vztahy otočit do podoby:

$$ŠT = OT^{d} \bmod n$$

$$OT = ŠT^{e} \bmod n$$

Tento algoritmus nebyl dosud úspěšně prolomen. Byl (možná také právě proto) normalizován několika standardizačními společnostmi a ve Spojených státech také patentován. Nyní je však algoritmus volně šiřitelný, neboť tomuto patentu vypršela platnost [4].

2.4.3 Další asymetrické algoritmy

El Gamal

Asymetrický algoritmus založený na principu problému složitosti výpočtu diskrétních logaritmů v algebraické struktuře (nazývanou též obecné těleso) [17].

Algoritmy eliptických křivek

Jedná se o algoritmy založené na principu eliptických křivek a definici matematických operací, které pracují nad těmito křivkami [17]. Tyto algoritmy dokázali šifrování provádět několikrát rychleji než RSA při zachování stejné bezpečnosti. Díky stále rostoucím výkonům počítačových systémů však téměř upadly v zapomnění [4].

3 Platforma Java ME

V následujících odstavcích této práce byly popsány základní vlastnosti platformy Java ME určené k tvorbě aplikací pro mobilní zařízení. Byla uvedena některá její omezení a specifické vlastnosti, které jsou při vývoji softwaru důležité.

3.1 Omezení při tvorbě mobilních aplikací

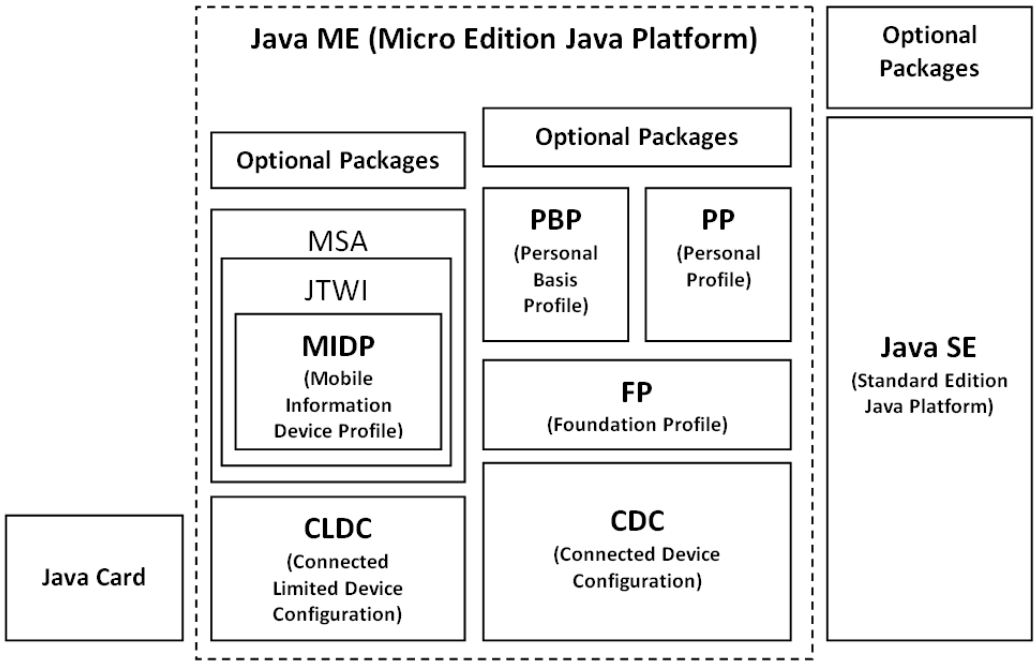
Při vývoji aplikací pro mobilní zařízení je vhodné přihlédnout k některým omezením, která tato zařízení oproti běžným počítačovým systémům limitují. Velikost obrazovky je zde oproti běžnému počítačovému systému zmenšena několikanásobně. Je však nutné počítat i s tím, že různí výrobci mají různě řešená API pro zobrazování. Například telefony Nokia s operačním systémem Symbian nemají nativně povoleno současně pracovat s třídou implementující rozhraní `CommandListener` a zároveň používat třídu `FullCanvas`, která umožňuje zobrazování na celé ploše obrazovky. Je proto lepší v případě potřeby vytváření grafického obsahu pracovat pouze s třídou `Canvas`, která použití rozhraní `CommandListener` neomezuje.

Dalším důležitým hlediskem je výkon zařízení. Je pravděpodobné, že některé telefony nejsou schopné zpracovat větší aplikace. Je to dáno vlastnostmi hardwaru telefonu nebo továrním nastavením. Další omezení může zapříčinit typ procesoru nebo velikost paměti. Některé telefony mají (díky svým konstrukčním vlastnostem) problém s hromadným zpracováním dat. Proto se někteří vývojáři uchylují k tomu, že omezují počty zpracovávaných záznamů, aby se aplikace při běhu nehroutily. Všechny tato hlediska jsou důležitou součástí programování mobilní aplikace a je nutné s nimi počítat.

3.2 Základní vlastnosti platformy Java ME

Java ME (někdy také označovaná jako J2ME nebo „Java Micro Edition“) je speciální micro-edice jazyka Java pro mobilní zařízení, která nezvládají edici standardní. Java ME je určená k vývoji aplikací pro mobilní zařízení (jako jsou mobilní telefony, PDA, pagery apod.). Tato zařízení jsou omezena velikostí a možnostmi svého hardwaru (paměti, displeje, zdrojem napájení nebo prostředky síťové komunikace). Hlavní výhodou je velká podpora pro běh javovských mobilních aplikací, kterou disponuje široká škála mobilních zařízení. Pro používání aplikací psaných v Java ME je nutné, aby telefon byl vybaven běhovým prostředím (tzv. virtuálním strojem), kterým drtivá většina dnešních přístrojů na trhu disponuje. Platforma Java ME je z hlediska uspořádání a rozdělení dále členěna na tzv. konfigurace a profily.

Obrázek 13 znázorňuje pozici platformy Java ME mezi ostatními javovskými platformami.



Obrázek 13 - Rozdělení javovských platforem

3.2.1 Java ME konfigurace

Pod pojmem konfigurace se v J2ME skrývá specifikace definující prostředí pro danou skupinu zařízení, která jsou určena množinou charakteristik:

- typ a frekvence procesoru
- druh a velikost dostupné paměti
- druh síťového připojení, kterým zařízení disponuje

Konfigurace má za úkol specifikovat minimální platformu pro cílové zařízení. Nejsou v ní definovány žádné volitelné funkce. Výrobci tyto specifikace musí plně implementovat z toho důvodu, aby vývojáři mohli tvořit aplikace co nejméně závislé na konkrétním zařízení. Pro platformu J2ME byly definovány dvě základní konfigurace - CLDC a CDC [1].

Každá konfigurace je složena z virtuálního stroje Javy a standardní kolekce javovských tříd podporujících programovací prostředí pro tvorbu aplikačního softwaru.

3.2.1.1 CLDC (Connected Limited Device Configuration)

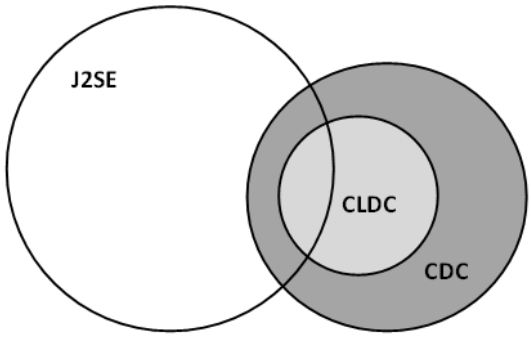
Konfigurace CLDC je určena pro nízko-úrovňovou oblast spotřební elektroniky se 16 bitovými nebo 32 bitovými procesory disponujícími navíc alespoň 192 kB paměti (kde min. 32 kB RAM je použito pro vlastní aplikace a 160 kB ROM paměti pro JVM a knihovny). Typická zařízení platformy CLDC jsou mobilní telefony a PDA. Tato konfigurace obsahuje zredukovanou podmnožinu balíčků `java.lang`, `java.io`, `java.util` a navíc obsahuje balíček knihoven `javax.microedition`.

Referenční implementace CLDC obsahuje zdrojový kód i binární produkt pro platformy Windows, Solaris i Linux. Obsahuje také virtuální stroj KVM, který je obdobou VM s redukovanými funkcemi [1].

3.2.1.2 CDC (Connected Device Configuration)

Konfigurace CDC je určena pro rychlejší 32 bitové procesory, které disponují minimálně 256 kB paměti RAM a min. 512 kB ROM. Nabízí prakticky plnou funkčnost platformy J2SE a podporuje dokonce i knihovny Swing. CDC tvoří mezistupeň mezi CLDC a úplnými stolními systémy, které používají právě platformu J2SE. Konfiguraci CDC lze najít například na dražších typech PDA a mobilních telefonech nebo i na Set-Top Boxech.

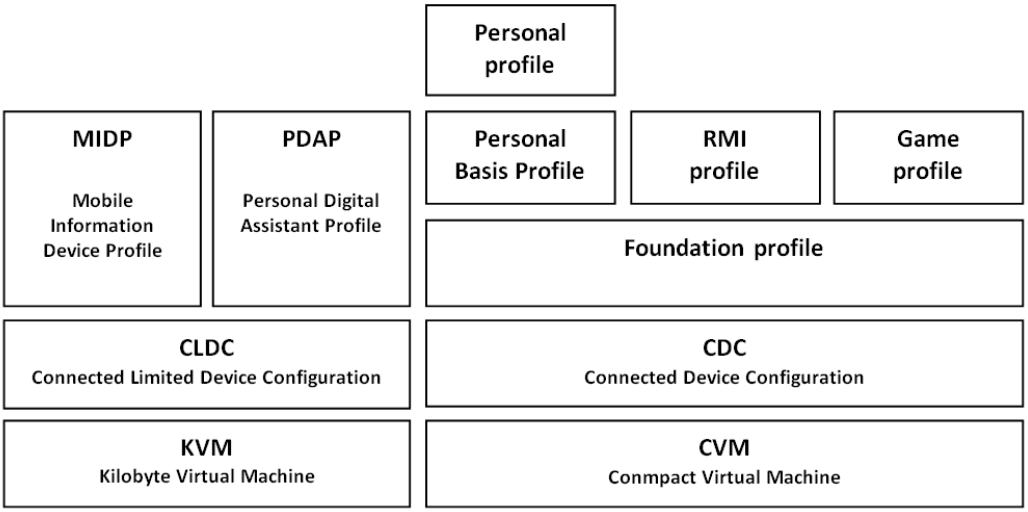
Vizuální náčrt prolnutí knihoven J2SE a knihoven konfigurací CDC a CLDC názorně vyjadřuje obrázek 14.



Obrázek 14 - Znázornění prolnutí knihoven CLDC, CDC a J2SE

3.2.2 Java ME Profily

Profil rozšiřuje konfiguraci přidáním dalších tříd poskytujících další funkce použitelné pro specifický druh zařízení nebo určitou skupinu zařízení na trhu. Obě uvedené konfigurace mají asociováno jeden nebo více specifických profilů [1]. Obrázek 14 znázorňuje konfigurace a profily, které je rozšiřují.



Obrázek 15 - Konfigurace a profily J2ME

MIDP (Mobile Information Device Profile)

MIDP je profil upřesňující CLDC konfiguraci přímo pro mobilní telefony. Přidává např. možnost ovládání aplikací pomocí tlačítek telefonu a upřesňuje nastavení minimální velikosti displeje na 96 x 54 pixelů. V tomto profilu je zaveden pojem MIDlet - označení javovské aplikace pro mobilní telefony CLDC. Dále přidává do CLDC místní úložný prostor RMS, součásti uživatelského rozhraní a síťové služby. Jedná se o nejznámější profil platformy Java ME. Tvoří základ platformy tzv. bezdrátové Javy (Wireless Java) [1].

PDAP

Profil PDA je velice podobný jako profil MIDP. Slouží však k vývoji aplikací určených k nasazení na PDA, které jsou (oproti mobilním telefonům) o něco vybavenější z hlediska velikostí displejů i pamětí. Obsahuje důmyslněji zpracované knihovny pro uživatelské rozhraní a rozhraní na bázi Javy pro přístup k některým funkcím hostitelského operačního systému [1].

Foundation Profile

Foundation Profile je základní profil pro ostatní profily CDC rozšiřující konfiguraci CDC o téměř všechny standardní knihovny Javy.

Personal Basic Profile

Personal Basic Profile přidává k základovému profilu funkce pro uživatelskou grafiku. Základní myšlenkou je použití na zařízeních, které nedisponují možností zobrazení více oken aplikace. Tyto zařízení mají pouze jednoduché zobrazovací schopnosti.

Personal Profile

Personal Profile je určen pro zařízení podporující složitější zobrazení uživatelského rozhraní. Přidává knihovny AWT.

RMI Profile

RMI Profile k základovému profilu přidává (prostřednictvím API) knihovny pro vzdálené volání metod platformy J2SE. Podporována je však pouze klientská část zmíněného API.

Game Profile

Game Profile rozšiřuje základní profil o Game Builder, který umožňuje snadné psaní herního kódu na zařízeních CDC.

3.2.3 Specifikace Java ME

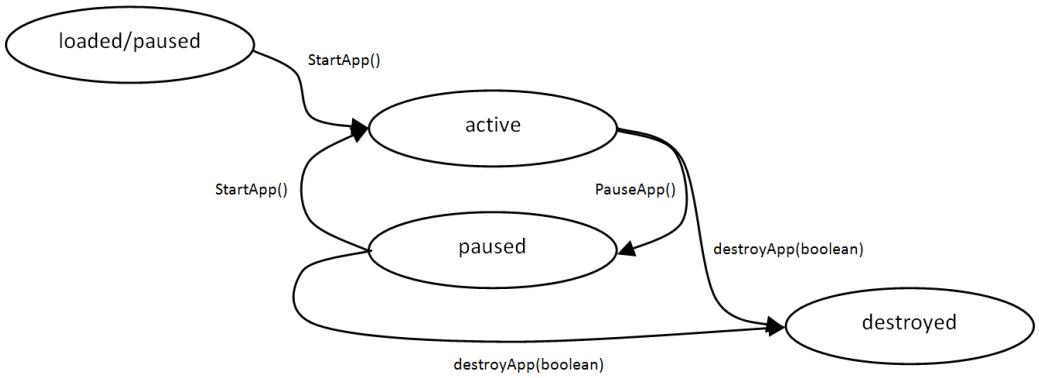
Konfigurace a profily platformy Java ME byly tvořeny jako součást procesu JCP (Java Comunity Process). Všechny konfigurace i profily byly vytvořeny jako žádost JSR (Java Specification Request) popisující zadání a definující přibližný popis oblasti, která má být pokryta [1]. Následující tabulka zobrazuje nejdůležitější JSR pro platformu J2ME a některé další, které s žádným profilem ani konfigurací J2ME přímo nesouvisí. V současné době je jich (jen pro J2ME) celkem 85. Kompletní aktuální seznam žádostí je přístupný ze zdroje [9].

Číslo JSR:	**Zadání:**
30	konfigurace CLDC v J2ME
37	profil MIDP pro J2ME
75	profil PDA pro J2ME - včetně přístupu do souborového systému
36	konfigurace CDC platformy J2ME
46	J2ME Foundation Profile
129	specifikace Personal Basis Profile
62	specifikace Personal Profile
66	RMI Profile
134	Java Game Profile
82	rozhraní pro Bluetooth
135	rozhraní pro multimédia J2ME
68	specifikace platformy J2ME
80	USB API
280	XML API pro Java ME

Tabulka 1 - Přehled nejdůležitějších JSR

3.2.4 Midlet

Pod pojmem midlet se ukrývá aplikace psaná v jazyce Java určená pro zařízení CLDC s MIDP. Každý midlet obsahuje alespoň jednu javovskou třídu, která je potomkem třídy `javax.microedition.midlet.MIDlet`. Midlety jsou spouštěny v běhovém prostředí VM (virtuálním stroji), který určuje přesně stanovený životní cyklus každého midletu. Každý midlet má tři stavy - pasivní, aktivní a zrušený. Stavy midletu řídí aplikační manažer. Obrázek 16 znázorňuje životní cyklus každého midletu.



Obrázek 16 - Životní cyklus midletu

Celou skupinu příbuzných midletů lze navíc sdružit do tzv. sady midletů. Sadu midletů lze do zařízení nainstalovat pouze jako ucelený balík. Celou ji lze ze zařízení i odebrat. Midlety v sadě sdílejí všechny prostředky běhového hostitelského prostředí. Zároveň spuštěné midlety sdílejí javovské třídy i ostatní prostředky, které jsou zavedeny do VM a mohou tak vzájemně sdílet i data [1].

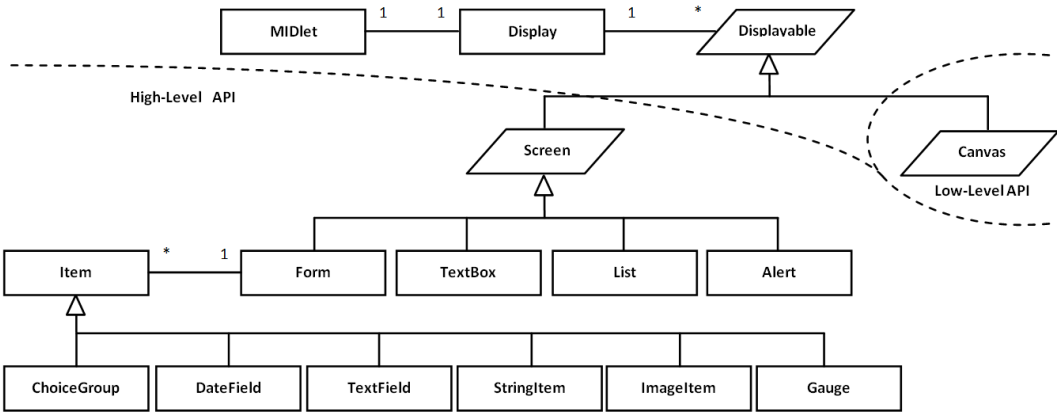
Úložiště trvalých záznamů RMS (Record Management Store) je pro celou sadu midletů také společné. Každému midletu je umožněn přístup k vlastním datům, ale i k datům ostatních midletů z téže sady. Midlet z jedné sady nemá přístup k záznamům midletů sady jiné díky omezení názvového mechanizmu použitého pro identifikaci dat [2].

Uživatelská rozhraní midletů

Midlety by měli být tvořeny tak, aby byly použitelné na co největším počtu různých zařízení. Tyto zařízení mají však mnohdy velice rozdílné možnosti zobrazení, od menších displejů mobilních telefonů až po větší displeje např. u PDA.

Jelikož mobilní zařízení (díky svým omezením) nemohou pracovat s klasickými komponentami platformy J2SE (sady AWT, nebo Swing), byl model uživatelského rozhraní definován jako poměrně hodně jednoduchý. Na rozdíl od aplikací J2SE (ve kterých lze běžně pracovat s několika formulářovými okny současně) má midlet možnost v jednom okamžiku zobrazit pouze jedno formulářové okno aplikace. Běží-li na zařízení současně více midletů, pouze formulář jednoho z nich lze zobrazit na displeji [1].

Základní knihovna pro vytváření uživatelských rozhraní midletů je definována v balíčku `javax.microedition.lcdui`. Tato knihovna obsahuje několik tříd reprezentujících obrazovku zařízení. Tyto třídy mohou poskytnout několik druhů oken. Obrázek 17 demonstruje diagram tříd používaných pro vytváření uživatelského rozhraní midletů.



Obrázek 17 - Struktura tříd GUI v MIDP

Třídy Display a Displayable

Třída Display je reprezentací logické obrazovky zařízení, na kterou midlety zobrazují své uživatelské rozhraní. Každý midlet má přístup pouze k jedné instanci této třídy. K získání odkazu na instanci se používá volání následující statické metody:

```
public static Display getDisplay(MIDlet midlet);
```

Většina midletů tuto metodu vyvolává hned při startu (po prvním zavolání metody `startApp()`). Následně je získaná instance třídy `Display` použita k zobrazení uživatelského rozhraní midletu. Každá instance třídy `Display` je pro každý midlet jedinečná. Metodu `getDisplay()` lze proto volat kdykoliv od prvního vytvoření po ukončení aplikace metodou `destroyApp()`. Všechny stavy obrazovky se konstruují přidáváním položek nebo vykreslením grafiky k oknu na nejvyšší úrovni. Tato okna jsou odvozena od abstraktní třídy `Displayable`, která je pro uživatele transparentní. Tato třída (resp. instance třídy od ní odvozené) je přidružena k objektu typu `Display` pomocí metody:

```
public void setCurrent (Displayable displayable);
```

Podobným postupem lze také instanci této třídy získat zpět - voláním metody:

```
public Displayable getCurrent();
```

Protože instance třídy `Display` může zobrazovat pouze jednu obrazovku, je voláním metody `setCurrent()` původní obsah odebrán a nahrazen novým (předaným v parametru).

Díky této metodě lze na displeji zobrazit instance tříd, které jsou jejími potomky. Jedná se o objekty typu `Screen` a `Canvas`. Třída `Screen` reprezentující obrazovku je určena pro zobrazení běžných vysokoúrovňových komponent jako je formulář (třída `Form`), seznam položek (třída `List`), výstražné hlášení (třída `Alert`) nebo textové pole (třída `TextBox`). Třída `Canvas` je určena pro vykreslování grafického obsahu (využitím Low-Level API).

Formuláře

Každý objekt třídy `Form` (jenž je potomkem tříd `Screen` a `Displayable`) může obsahovat několik položek třídy `Item`. Tato třída je předkem následujících důležitých komponent určených k umístění na formuláři:

Komponenta:	Popis:
TextField	textové pole
StringItem	textový řetězec
DateField	pole pro zobrazení data
ChoiceGroup	skupina vypratelných položek
ImageItem	položka pro zobrazení obrázku

Tabulka 2 - Důležité komponenty pro formuláře

Tyto a ještě některé další komponenty lze potom do formuláře vložit pomocí příkazu `append()` o následující syntaxi:

```
textField tfPwd = new TextField("Password:", "", 12, TextField.PASSWORD);
frmForm.append(tfPwd);
```

Dále je z hlediska ovládání aplikace do instancí zobrazovaných potomků abstraktní třídy `Screen` nutné přidat možnost ovládání. K tomu slouží příkazy.

Příkazy

Příkazy jsou v midletu objekty typu `Command` a slouží k ovládání aplikace. Každý příkaz je vytvořen pomocí svého konstruktoru o následující syntaxi:

```
public Command (String label, int type, int priority);
```

Argument `label` slouží k předání textu, který je použit k reprezentaci objektu typu `Command` v uživatelském rozhraní. Argumenty `type` a `priority` slouží midletu k rozhodnutí kam na formuláři příkazy umístit. Po vytvoření objektu už nelze tyto vlastnosti měnit. Příkazy lze do uživatelského rozhraní přidat pomocí příkazu `addCommand()`. Přidání je realizováno následující syntaxí:

```
cmdMenu = new Command("Menu", Command.Screen, 1);
frmForm.aaddCommand(cmdMenu);
```

Způsob ovládání midletu

Po vytvoření formuláře, přidání komponent a příkazů je nutné přimět midlet, aby po zadání příkazu provedl požadovanou akci. O rozpoznání příkazu se stará metoda `commandAction()`. Aby mohla být tato metoda využita, musí hlavní třída midletu implementovat rozhraní `CommandListener`. Teprve potom lze na příkaz příslušně reagovat. Následující příklad uvádí nástin postupu pro rozpoznání příkazů a vyvolání požadované metody:

```
public class Midlet extends MIDlet implements CommandListener {
…       // zkráceno o vytvoření formuláře přidání příkazu a vyvolání formuláře
public void commandAction(Command c, Displayable d) {
String label = c.getLabel();
if (label.equals("Exit")) {
        destroyApp(true);
else if ((label.equals("Menu") && d == frmForm))
        showMenu();    // metoda která nastaví menu na displej
```

3.2.5 Možnosti midletů pro ukládání uživatelských dat

Většina běžných aplikací ukládá nějaká data (ať už jde pouze o uložení vlastního nastavení, nějaká další data vytvořená během činnosti, nebo data zadaná uživatelem). Java ME na mobilních zařízeních nemá nativně k dispozici souborový systém jako v případě PC. Možnosti ukládání dat mobilní aplikace však existují.

RMS

Jedním z možných způsobů jak ukládat data mobilní aplikace je využití RMS (Record Mamagement System). Všechny potřebné nástroje pro práci s trvale ukládanými daty v systému RMS jsou v balíčku `javax.microedition.rms`.

Každý midlet má k dispozici v systému RMS vlastní databázi. Důležitou vlastností je, že po odebrání midletu z přístroje je zničena také báze jeho uložených dat.

K ukládání a načítání dat z RMS lze využívat balíčku `java.io`. Samotnou databázi je nutno před použitím otevřít metodou `OpenRecordStore()`. Po ukončení vstupně výstupních operací (zápisu, čtení, úprav) je nutné jí opět

zavřít pomocí metody `CloseRecordStore()`. Tyto přístupové operace jsou párové (kolikrát je databáze otevřena, tolikrát se musí zase uzavřít). Následuje ukázka kódu pro zápis a čtení:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
RecordStore rs = RecordStore.openrecordStore(nazev, true);
            dos.writeInt(cislo);
            dos.writeUTF(retezec);
byte[] bytes = baos.toByteArray();
rs.addRecord(bytes, 0, bytes.length);
…
// čtení zpět:
RecordStore rs = RecordStore.openrecordStore(nazev, false);
byte[] bytes = rs.getRecord(1);
rs.closeRecordStore();
DataInputStream dis = new DataInputStream(bytes);
Int cislo = dis.readInt();
String retezec = dis.readUTF();
…
```

K vypsání více záznamů jsou k dispozici funkce tříd `RecordEnumeration`, `RecordFilter` a `RecordCompaartor`. `RecordEnumeration` vytvoří výčet z celé databáze záznamů. Díky dalším dvěma lze uplatnit na tento výčet řazení nebo filtrování.

Vymazání nebo úprava záznamu v databázi je rovněž obsluhována metodami třídy `RecordStore`. Metoda `deleteRecord()` vymaže z databáze záznam o daném klíči. Metoda `setRecord()` záznam na zadaném indexu nahradí novým. Následující ukázka kódu demonstruje základní syntaxi:

```
rs.deleteRecord(klic);
rs.setRecord(klic, bytes, 0, bytes.length);
```

FileConnection

Další možností pro ukládání dat v mobilním telefonu je využít tzv. FileConnection API, které však nemusí mít všechny přístroje dostupné. Jedná se o zajištění přístupu do souborového systému přístroje (JSR 75). K implementaci je nutné využít balíku `javax.microedition.io.file.FileConnection`, v němž jsou obsaženy všechny potřebné knihovny. Následující ukázka kódu naznačuje způsob použití:

```
…
String url = path + fileName;
byte data[] = retezec.getBytes();
FileConnection fconn = (FileConnection) Connector.open(url, Connector.READ_WRITE);
if (!fconn.exists())
        fconn.create();
OutputStream os = fconn.openOutputStream();
os.write(data);
os.close();
fconn.close();
…
```

Využitím této knihovny lze snadno přistupovat k různým předdefinovaným adresářům souborového systému zařízení. Několik příkladů uvádí následující tabulka (další lze dohledat na adrese zdroje [10]):

Řetězec definice FileConnection:	**Typ adresáře:**
fileconn.dir.memorycard	paměťová karta přístroje
fileconn.dir.photos.name	adresář k ukládání fotek
fileconn.dir.tones.name	adresář pro vyzváněcí tóny
fileconn.dir.videos.name	adresář pro ukládání videí
fileconn.dir.music	adresář pro audio soubory (mp3…)

Tabulka 3 - Příklady přístupových řetězců pro FileConnection

3.3 Stručný popis OS Symbian S60

Operační systém Symbian S60 je svobodný mobilní operační systém určený pro tzv. „chytré“ mobilní telefony. Je doplněn knihovnami, grafickým uživatelským rozhraním a referenční implementací nástrojů vytvořených firmou Symbian Ltd. Symbian je vytvořen výhradně pro zařízení osazené procesory typu ARM. Do systému lze nainstalovat další aplikační software (bohužel často závislý na verzi OS). Používán je především v telefonech značky Nokia. Podporuje běh aplikací vytvořených v J2ME.

Výhody OS Symbian:	Nevýhody OS Symbian:
vysoká stabilita	vzájemná nekompatibilita aplikací
podpora Wi-Fi sítí	ukončení dalšího vývoje ze strany společnosti Nokia od 11. 2. 2011
možnost upgrade firmwaru přes internet	
přibývá podpora HW klávesnic	

Tabulka 4 - Výhody a nevýhody OS Symbian

Symbian S60 je nejčastěji používanou verzí a telefony na nichž je nainstalován se těší velké oblibě. Je to zapříčiněno jejich snadnou softwarovou rozšiřitelností a vzhledem k výbavě telefonů také poměrně nízkou cenou. Typické přístroje vybavené OS Symbian S60 jsou např. Nokia N70, Nokia N72, Nokia E51, Nokia E52, Nokia 6720, Nokia E90 nebo také Siemens SX1 [20].

3.4 Možnosti propojení mobilního telefonu s PC

3.4.1 USB kabel

Základní možností připojení většiny současných mobilních telefonů k počítači je využití USB kabelu. Po připojení telefonu k počítači přes USB kabel lze také běžně (pomocí nainstalovaného SW od výrobce - např. Nokia PC Suite) instalovat aplikace, aktualizovat firmware, spravovat kontakty, zprávy atd. Z některých telefonů (nebo jejich paměťových karet) se po připojení stane mass storage device.

3.4.2 Bluetooth

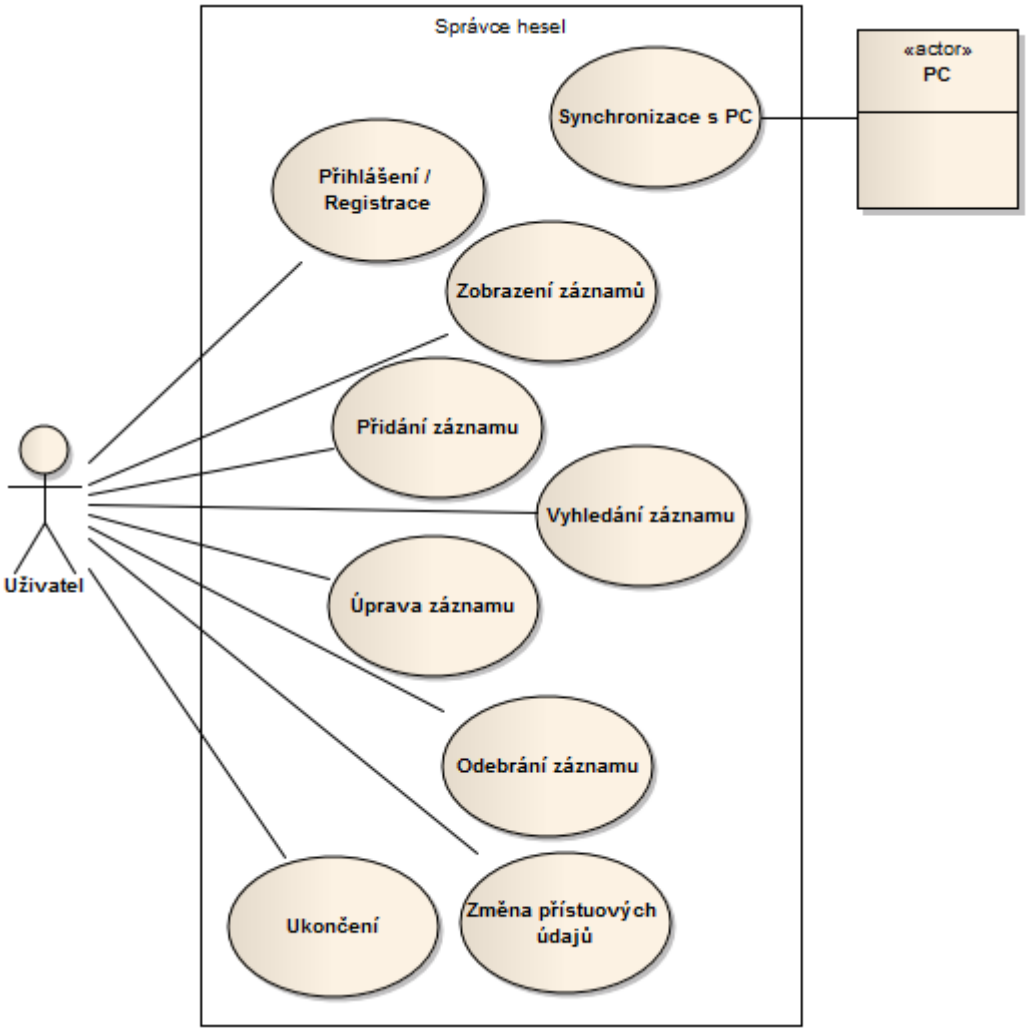
Bluetooth je bezdrátový standard pro propojení dvou zařízení na kratší vzdálenosti (běžně do deseti metrů). Záleží na typu zařízení a na použité specifikaci. I s využitím bezdrátového připojení přes bluetooth lze do telefonu instalovat aplikace, spravovat zprávy a kontakty, nahrávat nejrůznější soubory a např. i aktualizovat firmware přístroje. Tento způsob je navíc velice pohodlný, protože je bezdrátový a není nutné telefon připojovat kabelem.

4 Návrh mobilní aplikace

Další část této práce popíše návrh mobilní aplikace. Budou popsány jednotlivé třídy, jejich důležité metody a využité softwarové vybavení.

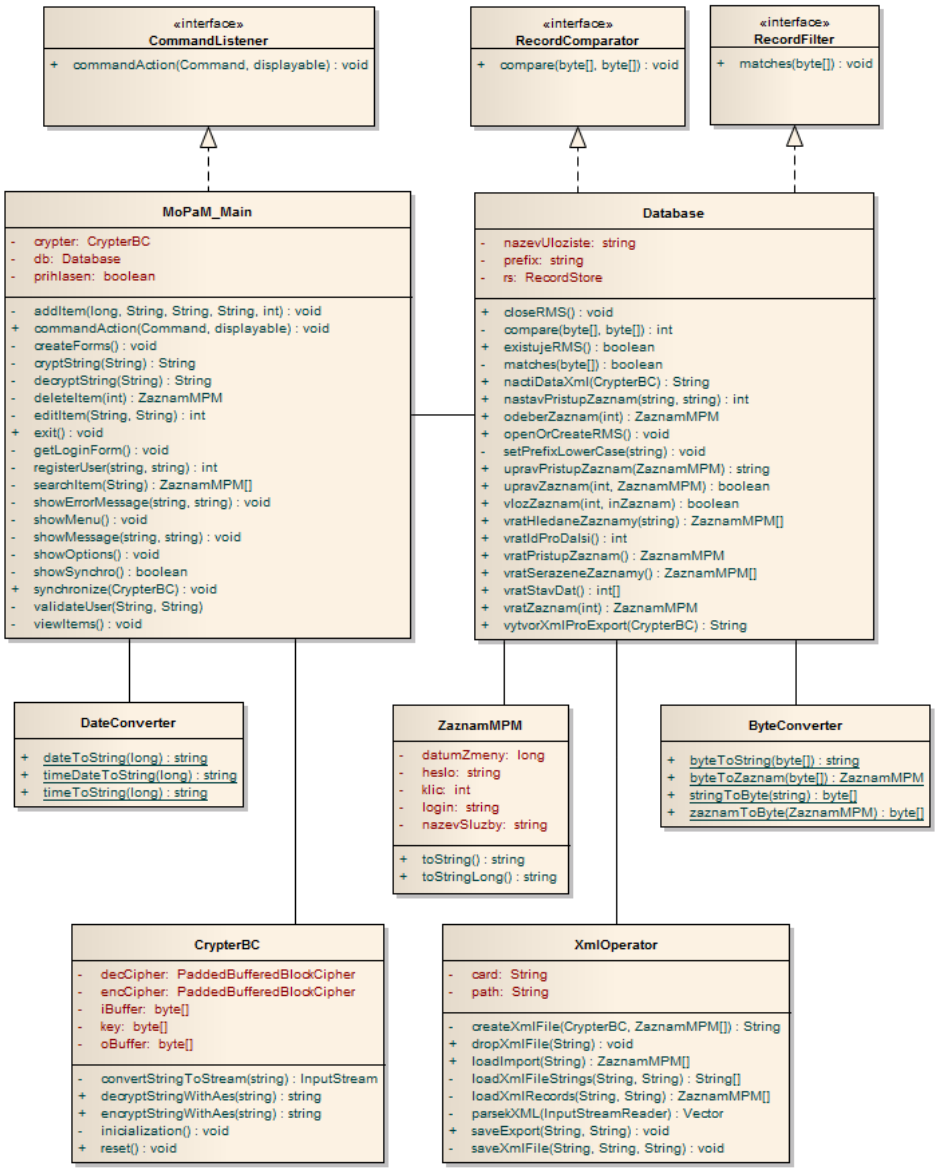
4.1 UseCase diagram

Následující UseCase diagram zachycuje přehled případů užití aplikace. Do diagramu byly zahrnuty všechny stěžejní funkcionality aplikace. Mobilní aplikace má přitom stejné požadavky na případy užití jako aplikace počítačová. Proto je tento diagram případů užití pro obě společný.



Obrázek 18 - Diagram případů užití aplikací

4.2 Diagram tříd mobilní aplikace



Obrázek 19 - Diagram tříd mobilní aplikace

4.3 Datová struktura záznamu

Datová struktura `ZaznamMPM` (viz diagram tříd) byla definována s ohledem na co nejmenší paměťové nároky (pouze nejnutnější údaje). Jako klíč je použita datová složka typu `int`. Název služby, přihlašovací jméno a heslo jsou do objektu typu `ZaznamMPM` vkládány jako datové složky typu `String`. Datum změny je uloženo v datové složce typu `long` udávající počet milisekund od 1. 1. 1970.

4.4 Databáze záznamů

Třída `Database` (viz diagram tříd) obsahuje implementace veškerých metod pracujících s RMS aplikace. Tyto metody jsou využity pro ukládání, úpravy, mazání, vracení výčtů uložených záznamů, jejich řazení a filtrování. Záznamy jsou v systému ukládány ve formě bytů a musí být před uložením převedeny na pole typu `byte[]` tuto funkci obsluhuje metoda `zaznamToByte()` pomocné třídy `ByteConverter` (viz diagram tříd). Inverzní funkci využitou při načítání záznamů z RMS zajišťuje metoda stejné třídy `byteToZaznam()`. Při implementaci metod pro ukládání, načítání, úpravy, mazání, vyhledávání a řazení záznamů bylo využito základních metod třídy `RecordStore`:

- `addRecord()`,
- `setRecord()`,
- `getRecord()`,
- `deleteRecord()`,
- `matches()` (metoda rozhraní `RecordFilter`),
- `compare()` (metoda rozhraní `RecordComparator`),
- `enumerateRecords()`.

Třída `Database` dále implementuje metody pro synchronizaci záznamů s počítačovou aplikací využívající metod třídy `XmlOperator`.

4.5 Šifrování položek záznamů

K implementaci šifrovacího algoritmu byla využita třída `CrypterBC` (viz diagram tříd), jejíž metody s využitím `Bouncy Castle Crypto API` jsou určeny pro šifrování a dešifrování řetězců.

4.5.1 Výběr vhodného algoritmu

Pro aplikaci provozovanou na mobilním zařízení bylo nutné dbát při výběru šifrovacího algoritmu na jeho výpočetní náročnost. Aplikace by měla být i při použití šifrování svižná a nemělo by se nijak projevit, že na pozadí běží náročná operace.

4.5.2 Volba volně dostupných existujících knihoven

Existuje poměrně hodně hotových knihoven, které poskytují metody šifrovacích algoritmů. Některé jsou navíc použitelné zdarma a ověřené mnoha uživateli, kteří je již do svých aplikací implementovali. Jedním z takových zdrojů užitečných šifrovacích knihoven je (výše zmíněný) projekt BouncyCastle.org poskytující knihovny nejen pro jazyk C# a Java, ale dokonce i pro mobilní odnož Javy J2ME. Z tohoto balíku šifrovacích knihoven bylo pro implementaci využito šifrovacího algoritmu AES. Jedním z kritérií výběru bylo, že se jedná o symetrický algoritmus (ne tolik náročný na výkon). Druhým důležitým důvodem k jeho použití byly informace o něm nalezené v literatuře (jeho deklarovaná vysoká bezpečnost a dlouhodobá použitelnost).

Po importování dvou archivů typu ZIP do vývojového prostředí NetBeans byly zpřístupněny knihovny algoritmů vhodné pro použití v J2ME. Aby nebyl výsledný soubor JAR (díky zanesení mnoha souborů knihoven) příliš velký, existuje ve vývojovém prostředí NetBeans funkce Obfuscator. Obfuscator odebere ze souboru JAR knihovny, které nebyly využity. Výsledná aplikace je tak jen o málo větší než před importováním knihoven. Toto řešení je velice rychlé, elegantní a implementačně nenáročné.

4.6 Operace s XML soubory

Pro export a import údajů z XML souborů byla implementována třída `XmlOperator` (viz diagram tříd) obsahující všechny potřebné metody (pro převod seznamu záznamů do XML a následný zpětný import). Třída `XmlOperator` ve svých metodách implementuje také možnost přístupu do souborového systému zařízení s využitím metod `FileConnection API`.

4.7 Pomocná třída DateConverter

Třída `DateConverter` byla vytvořena za jediným účelem. Datum je v platformě J2ME ukládáno ve formě datového typu `long`. Nelze využít žádné pokročilé konverzní funkce (dostupné pro platformu J2SE) pro převod data do normálního (lidsky čitelného) formátu. Proto byla implementována pomocná třída `DateConverter`, jejíž metody převádí zadané datum v parametru typu `long` do čitelného (a srozumitelného) řetězce typu `String`.

4.8 Popis potřebného softwarového vybavení

Pro vývoj mobilní i počítačové aplikace bylo využito následující softwarové vybavení a doplňující balíčky.

4.8.1 NetBeans IDE

Jako vývojové prostředí bylo zvoleno NetBeans IDE 7.0 dostupné z [11], které v základní výbavě obsahuje podporu pro programovací jazyky Java (pro platformy Java SE, Java EE, Java ME i Java Card), C/C++, PHP a Groovy. Toto prostředí navíc nabízí možnost využít k vývoji midletů vestavěné emulátory. Nabízí také variantu vytvoření základní kostry midletu v grafickém designeru. Pomocí designeru lze vytvořit jednotlivé obrazovky aplikace, přidat do nich položky, příkazy a také si definovat posloupnosti přechodů mezi jednotlivými obrazovkami.

4.8.2 Java Development Kit

JDK je sada vývojových nástrojů pro tvorbu aplikací, appletů a prvků pomocí programovacího jazyka Java. Před instalací vývojového prostředí NetBeans byl nainstalován nejnovější Java Development Kit pro OS Windows, ve verzi JDK 6 Update 26 dostupný z [12].

4.8.3 Software Development Kit pro Nokia S60

SDK pro vývoj aplikací určených k běhu na OS Symbian S60 obsahuje speciální emulátor simulující přesně daný typ telefonu. V SDK jsou také obsaženy další výrobcem přidané knihovny. Pro jeho instalaci je nutné přidat SDK do NetBeans jako novou platformu. Je zdarma dostupný na stránkách výrobce [13].

4.8.4 Šifrovací knihovny BouncyCastle.org

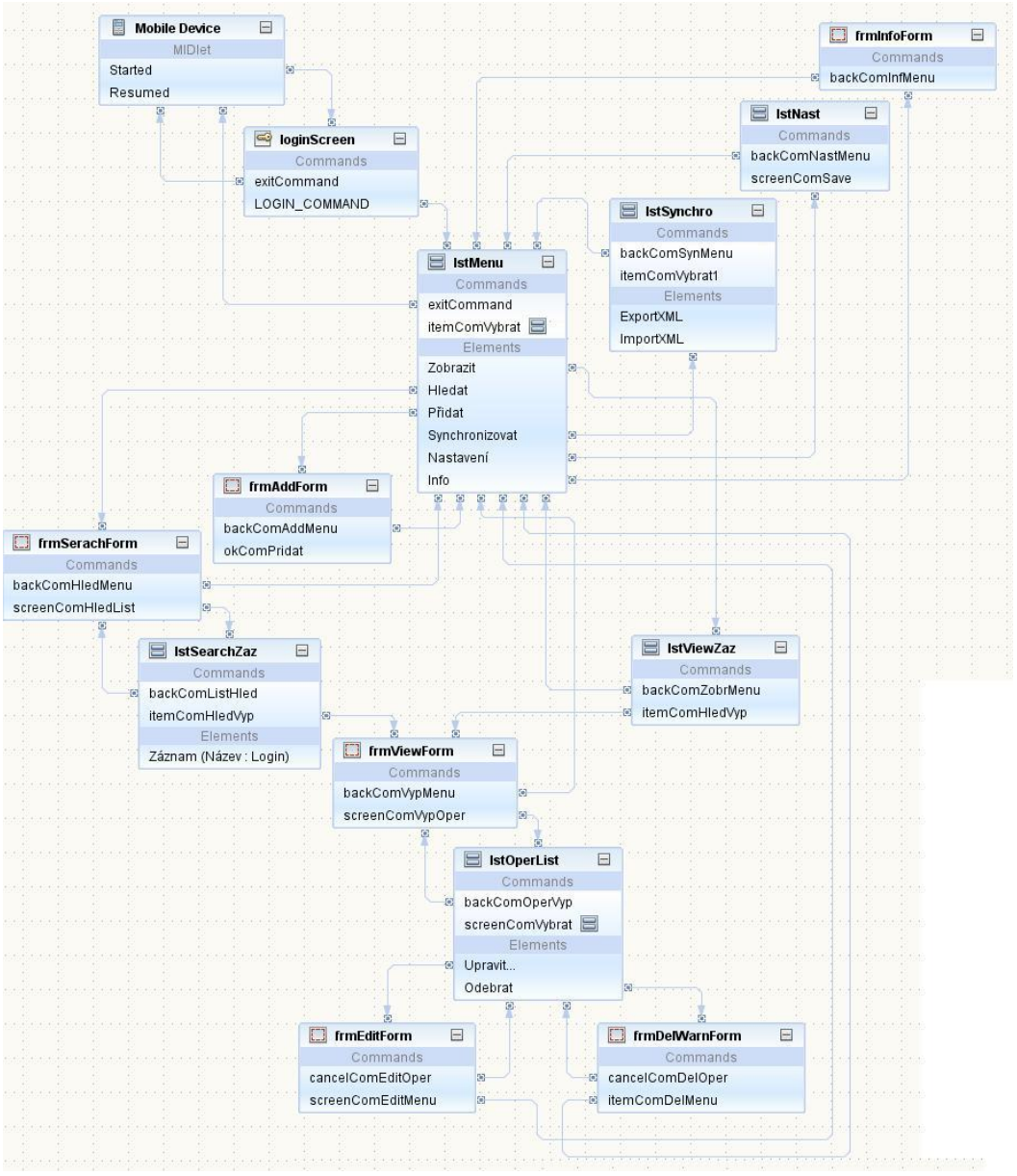
Pro implementaci šifrovacího algoritmu bylo využito volně dostupných knihoven projektu BouncyCastle.org. Využitím `Bouncy Castle Crypto API` byly knihovny elegantně přidány do aplikace. Balíky knihoven pro různé platformy v různých verzích jsou dostupné z webových stránek projektu [14].

4.8.5 Knihovny pro práci s XML

Pro zpracování XML souborů bylo využito hned několik balíčků. Pro implementaci exportu a importu záznamů na mobilním zařízení byly využity zdarma dostupné balíky knihoven `org.kxml.parser` a `kNanoXml`. Uvedené knihovny jsou z hlediska velikosti optimalizovány právě pro použití na mobilních zařízeních. Pro zpracování exportu a importu počítačové aplikace (serializaci do XML) bylo využito balíku `javax.xml.bind`.

4.9 Návrh struktury midletu v grafickém designeru

Na následujícím schématu je zobrazena struktura vytvářeného midletu. Schéma bylo vytvořeno v grafickém designeru vývojového prostředí NetBeans. Tímto způsobem lze kompletně vytvářet jednodušší midlety. Pro realizaci této aplikace bylo ovšem nutné psát midlet manuálně, protože bylo nezbytné „ručně“ programovat některé události (spouštění metod apod.). Zároveň bylo zapotřebí přidat další specializované třídy (viz diagram tříd). Schéma tedy udává pouze přibližnou informaci o řešení struktury hlavní třídy aplikace. Skutečná struktura je odlišná.



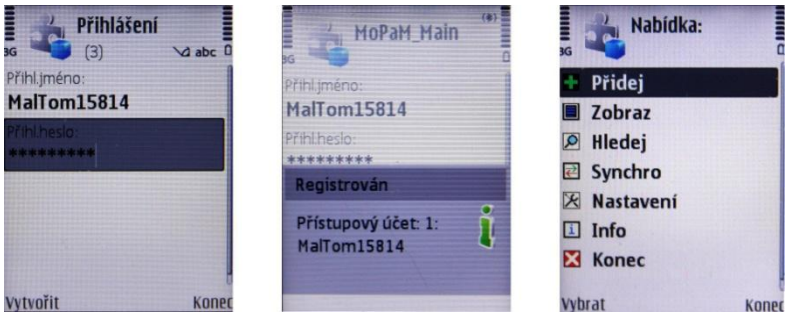
Obrázek 20 - Návrh struktury midletu (v grafickém designeru)

5 Popis implementovaných funkcí mobilní aplikace

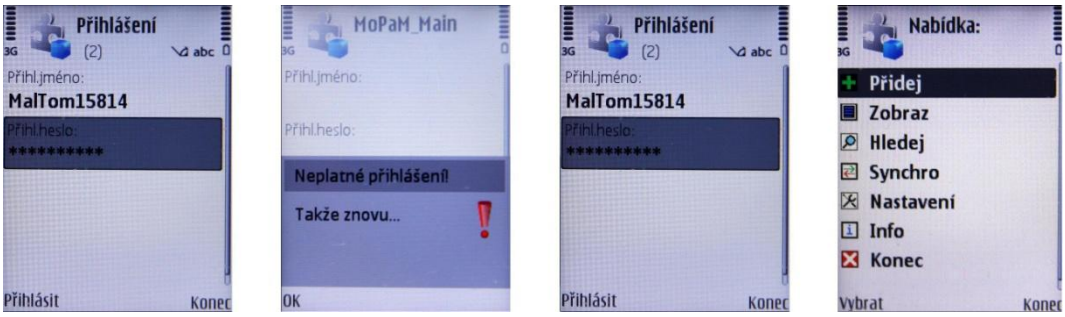
V následujících odstavcích byly popsány funkce mobilní aplikace, které byly implementovány. Jsou zde také přiloženy fotografie stavů obrazovky mobilního telefonu při jednotlivých úkonech.

5.1 Přihlášení při startu aplikace

Po startu aplikace je nutné ověřit identitu uživatele, neboť primárním účelem aplikace je ukládání a bezpečné přenášení přihlašovacích údajů do mnoha systémů. Při každém spuštění aplikace je vyžadováno ověření, zda je uživatel právě tím, kdo má právo na přístup k datům uloženým v aplikaci. Přihlášení zaniká při ukončení aplikace, aby nemohlo dojít ke spuštění a zobrazení záznamů cizí neoprávněnou osobou. Po instalaci aplikace do telefonu neexistuje RMS a je nutné ho nejdříve vytvořit. S tím je spojena i registrace uživatelského účtu. Poprvé zadané přihlašovací údaje jsou po ukončení aplikace použity pro ověření identity uživatele (při následných dalších spuštěních). Obrázek 21 ilustruje startovní přihlašovací formulář při registraci, obrázek 22 při následném spuštění a přihlášení (s případně napoprvé špatně zadanými údaji).



Obrázek 21 - Posloupnost obrazovek při registraci



Obrázek 22 - Posloupnost obrazovek při přihlášení

Následuje zkrácená ukázka kódu metody `validateUser()` sloužící pro ověření identity. Pro dešifrování porovnávaných údajů jsou volány metody třídy `CrypterBC`.

```
private void validateUser(String inLogin, String inPass) {
…
if (prihlasen != true) {
        …
        ZaznamMPM prist = db.vratPristupZaznam();
        String ulLogin = prist.getLogin();
        String ulHeslo = prist.getHeslo();
        ulLogin = desifruj(ulLogin);
        ulHeslo = desifruj(ulHeslo);
        if ((inLogin.equals(ulLogin)) && (inPass.equals(ulHeslo))) {
                prihlasen = true;
                menu();
        } else {
                prihlasen = false;
                tryAgain();
        }
…
}
```

5.2 Správa záznamů

Záznamy jsou ukládány do RMS aplikace fungujícího jako mobilní databázové úložiště dat. Pro každý midlet je toto úložiště vytvořeno samostatně a ostatní midlety k němu nemají přístup. Každé úložiště je identifikováno řetězcem sestávajícím z 0 až 32 unikódových znaků rozlišujícím velikost písmen.

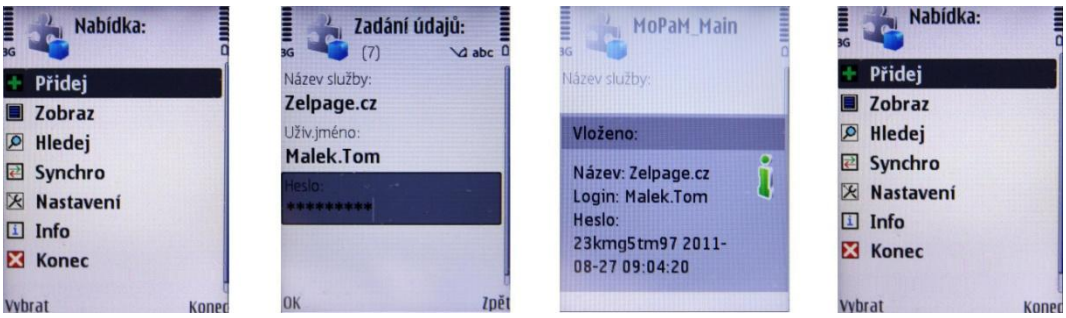
5.2.1 Přidání záznamu

Funkce k přidání záznamu byla implementována následujícím způsobem. Ze vstupního formuláře jsou načteny potřebné údaje. Následně je zavolána porovnávací funkce pro zjištění, zda se již v úložišti záznam se shodným názvem a uživatelským jménem nevyskytuje. Pokud je nalezen, je nabídnuta možnost jeho aktualizace (přepsání). Pokud dosud není podobný záznam (se stejným názvem a uživatelským jménem) v úložišti obsažen, je vytvořen objekt typu ZaznamMPM, do jehož datových složek jsou zadané údaje nastaveny. Položky uživatelského jména a hesla jsou šifrovány pomocí metody implementující šifrovací algoritmus AES. Nakonec je objekt převeden do formy pole bytů a zapsán do RMS.

Pokud daný záznam ještě v systému uložen nebyl, je mu přidělen nový klíč o jedničku vyšší než klíč posledního uloženého záznamu.

Pokud byl (podobný) záznam nalezen a přepsán, nadále si ponechává svůj klíč, název i uživatelské jméno. Pokud heslo nebylo stejné, je nahrazeno a datum změny je aktualizováno.

Na obrázku 23 je znázorněna posloupnost stavů obrazovky při přidání doposud neexistujícího záznamu. Obrázek 24 znázorňuje přepsání záznamu (stejného názvu a uživatelského jména).



Obrázek 23 - Přidání záznamu

Obrázek 24 - Výstraha a nahrazení existujícího záznamu

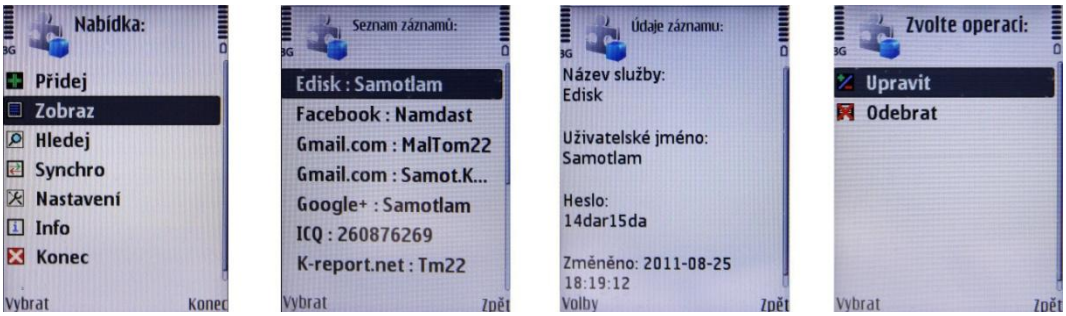
Následující ukázka kódu zobrazuje podstatu metody (třídy `Database`) pro vkládání záznamů do RMS aplikace:

```
public boolean vlozZaznam(int zazId, ZaznamMPM vklZaz) {
if (existujeRMS() == true) {
        openOrCreateRMS();
        byte[] poleBajtuZaz = ByteConverter.zaznamToByte(vklZaz);
        rs.addRecord(poleBajtuZaz, 0, poleBajtuZaz.length);
        closeRMS();
        return true;
} else
        return false;
…
```

5.2.2 Zobrazení záznamů

Implementace metody pro zobrazení seřazeného seznamu záznamů byla řešena pomocí nativních funkcí systému RMS. V metodě pro vrácení uložených záznamů bylo využito enumerátoru, který vrací všechny záznamy a implementace metody `compare()` (rozhraní `RecordComparator`) sloužící pro seřazení výčtu enumerátoru podle libovolné položky záznamu. Metoda tedy vrací množinu všech záznamů seřazených podle názvů (v případě shody názvů seřazených ještě podle uživatelských jmen). Množina záznamů je následně vypsána do seznamu na obrazovce. Jednotlivé záznamy jsou na displeji identifikovány pouze řetězcem ve formátu `nazev:login`. Po vybrání konkrétního záznamu lze zobrazit jeho detaily. Vybraný záznam je kompletně vypsán na obrazovku a zobrazí se všechny jeho položky kromě klíče, který není relevantní.

Obrázek 25 dokumentuje posloupnost stavů obrazovky při zobrazování kompletního seřazeného seznamu záznamů a následné zobrazení podrobností vybraného.



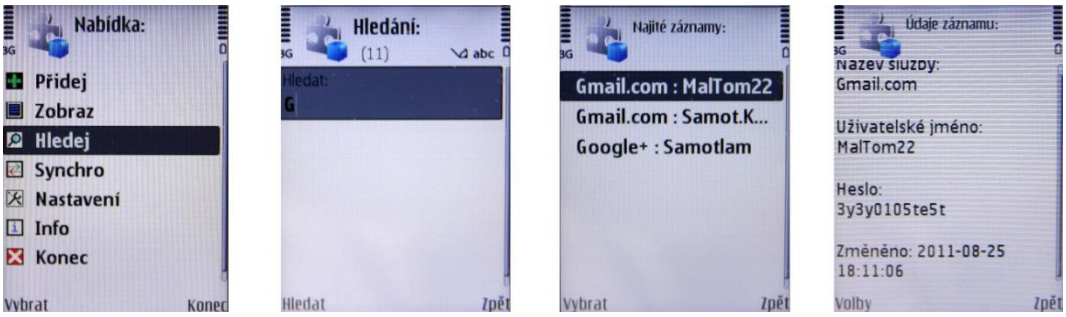
Obrázek 25 - Zobrazení záznamu a jeho detailů

Následující ukázka kódu demonstruje způsob využití metody `compare()` pro seřazení podle názvů a dále podle uživatelských jmen.

```
public class Database implements RecordFilter, RecordComparator {
…
int compare(byte[] rec1, byte[] rec2) {
…                                                    // načtení názvů a loginů
int compn = nazev1.compareTo(nazev2);
if (compn != 0) {                                    // vyhodnocení řazení dle názvů
if (compn < 0)
     return RecordComparator.PRECEDES;
       else
            return RecordComparator.FOLLOWS;
}
else {
int compl = login1.compareTo(login2);                // jinak dále dle jména
       if (compl == 0)
            return RecordComparator.EQUIVALENT;
       else if (compl < 0)
            return RecordComparator.PRECEDES;
       else
            return RecordComparator.FOLLOWS;
}
…
```

5.2.3 Vyhledání záznamů

Podobně jako metoda pro zobrazení kompletního seznamu záznamů je i metoda pro vyhledávání implementována pomocí standardních metod RMS a dalšího rozhraní `RecordFilter`. Metoda `matches()` slouží k porovnání, zda prvek výčtu odpovídá zadanému prefixu. V implementaci této metody bylo využito porovnání, které zamezí výstupu přihlašovacího záznamu (název začíná vykřičníkem) a zároveň porovnání se zadaným prefixem. Z výčtu záznamů jsou vráceny pouze ty záznamy, které projdou porovnáním (hledaného) názvu pomocí metody `startsWith(prefix)`. Takto získaný výčet záznamů je navíc ještě seřazen díky metodě `compare()`. Na obrázku 26 je demonstrována postupnost stavů obrazovky při vyhledávání záznamů (podle názvu služby).



Obrázek 26 - Postup vyhledání záznamu (dle názvu)

Následující ukázka kódu naznačuje způsob implementace hlavní rozhodovací podmínky metody `matches(byte[] data)`.

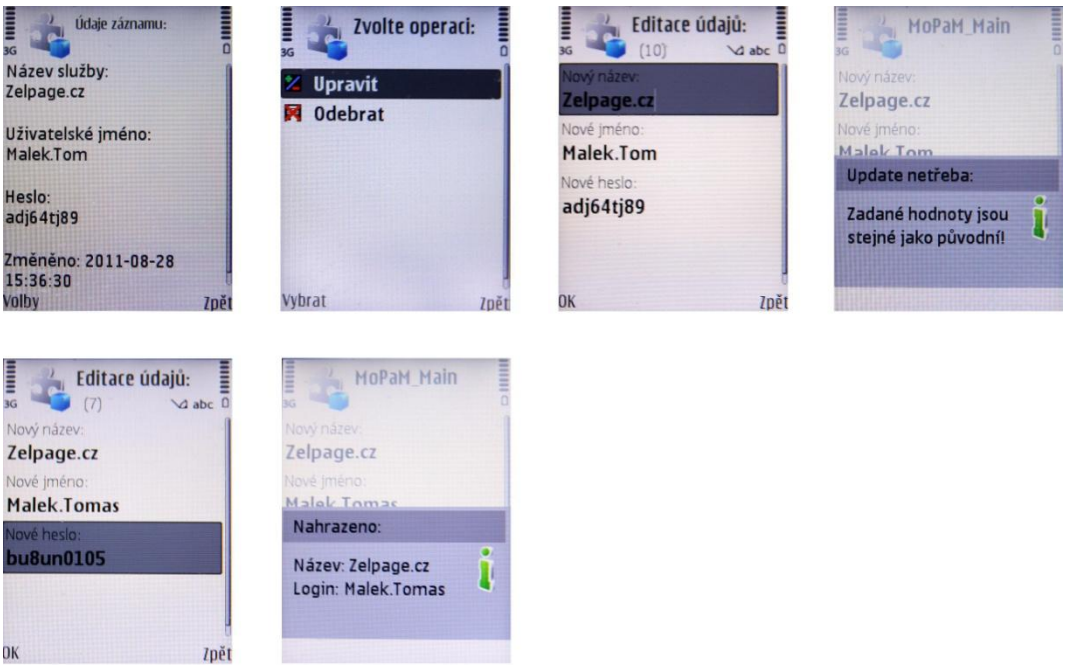
```
public class Database implements RecordFilter, RecordComparator {
…
private boolean matches(byte[] data) {
…
if ((nazev.toLowerCase().startsWith(prefix))
&& (!(nazev.toLowerCase().startsWith("!"))))        // odfiltrování přihl. záznamu
                return true;
            else
                return false;
…
```

5.2.4 Zobrazení podrobností záznamu

Zobrazení podrobností záznamu je zřetelné z obrázků 25 a 26. Zobrazit detaily záznamu lze nejen výběrem z kompletního seznamu, ale i výběrem z výčtu záznamů vyhledaných dle prefixu. Tento záznam je označen jako aktivní a jeho klíč je dočasně uložen. Aktivní záznam lze upravit nebo odebrat ze seznamu.

5.2.5 Úprava záznamů

Pro implementaci metody úpravy záznamu byl navržen následující postup. Aktivní záznam je načten z RMS a jeho současné údaje jsou předem vyplněny do položek typu `TextField` editačního formuláře. Pokud jsou po potvrzení položky stejné (jako ty původní), je díky obsluhující funkci pouze vyvoláno hlášení a editace se zruší. V opačném případě jsou záznamu (na daném klíči v RMS) upraveny datové složky na nové hodnoty a je mu aktualizováno datum změny. Obrázek 27 zobrazuje postup editace záznamu včetně upozornění na stejné hodnoty.



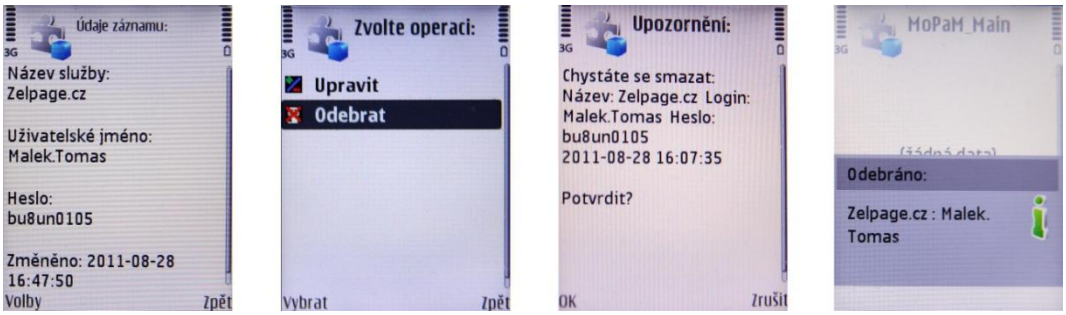
Obrázek 27 - Úprava záznamu

Následující ukázka kódu naznačuje způsob úpravy záznamu v RMS:

```
public boolean upravZaznam(int zazId, ZaznamMPM uprZaz) {
…
if (existujeRMS() == true) {
        openOrCreateRMS();
       byte[] nPoleBajtu = ByteConverter.zaznamToByte(uprZaz);
       rs.setRecord(zazId, nPoleBajtu, 0, nPoleBajtu.length);
       closeRMS();
…
```

5.2.6 Odebrání záznamu

Odebrání záznamů bylo implementováno opět pomocí nativní funkce systému RMS. Díky uloženému klíči záznamu lze vybraný záznam snadno identifikovat v RMS a po potvrzení je tento záznam z RMS aplikace odstraněn. Obrázek 28 demonstruje chování aplikace při mazání záznamu.



Obrázek 28 - Odebrání záznamu

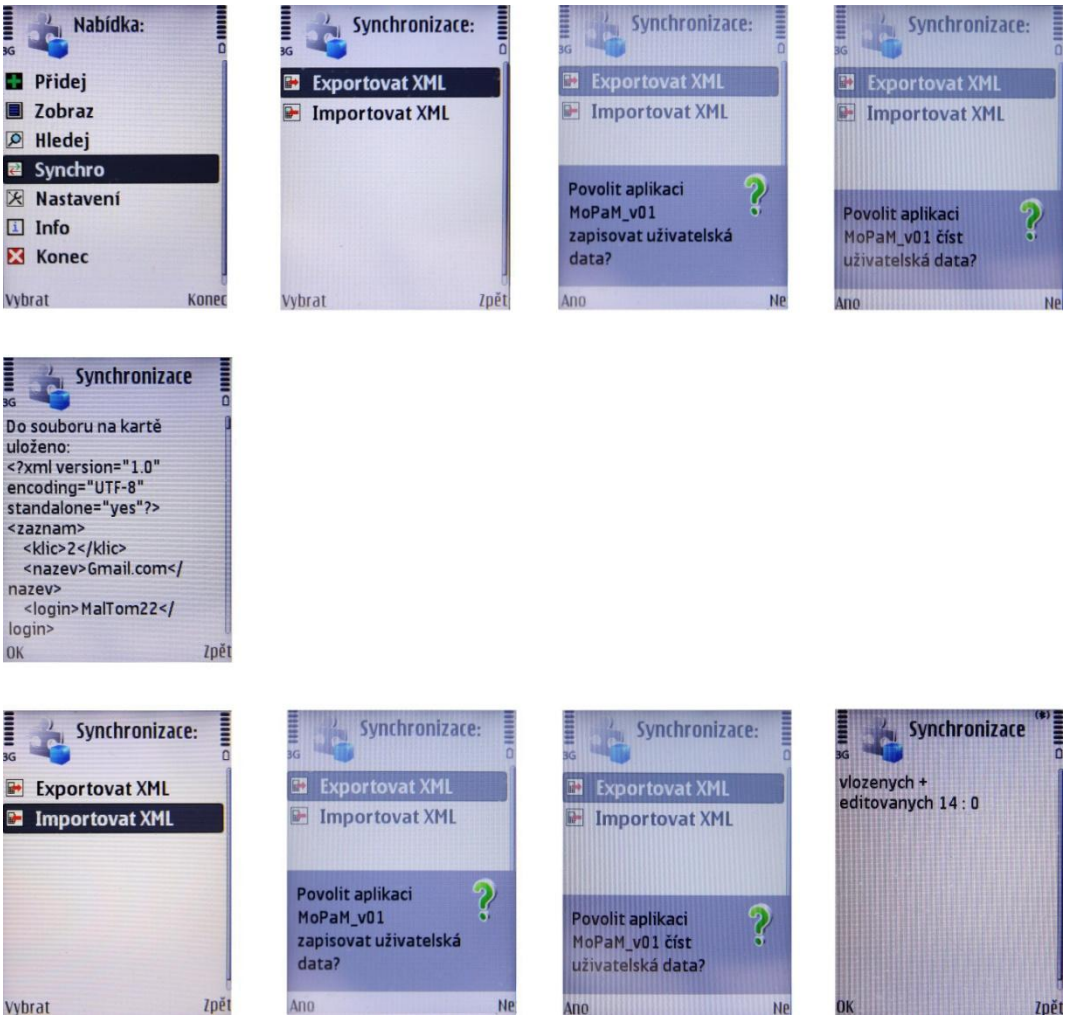
Následující ukázka kódu naznačuje způsob odstranění záznamu z RMS:

```
public ZaznamMPM odeberZaznam(int zazId) {
…
if (existujeRMS() == true) {
        openOrCreateRMS();
        ZaznamMPM odebrany = vratZaznam(zazId);
       rs.deleteRecord(zazId);
       closeRMS();
        return odebrany;
…
```

5.3 Synchronizace s PC

Synchronizace s počítačovým programem byla řešena neortodoxním způsobem. V telefonu jsou z aplikace na paměťovou (microSD) kartu (díky využití balíku `javax.microedition.io.file.FileConnection`) vyexportovány záznamy ve formě XML souboru (s názvem `Ph2Pc.xml`). Tento soubor je po připojení telefonu k počítači přes USB kabel manuálně načten do počítačové aplikace (popsané v kapitole 6). V počítačové aplikaci je soubor parsován a získané záznamy jsou uloženy do báze dat počítačové aplikace. Nové záznamy jsou přidány, upravené aktualizovány. Načtený soubor je vymazán. Následně je na paměťovou kartu zapsán druhý soubor XML (s názvem `Pc2Ph.xml`) obsahující rozdíly. Do souboru jsou exportovány záznamy, které v exportu z mobilního zařízení nebyly, ale v databázi počítačové aplikace ano. Po opětovném odpojení telefonu a výběru položky menu (mobilní aplikace) `Synchro/Importovat XML` jsou nové záznamy přidány s novými klíči. V případě shody názvů, uživatelských jmen a aktuálnějších časových údajů jsou stávající záznamy editovány. Načtený soubor je opět vymazán.

Na obrázku 29 jsou zobrazeny jednotlivé kroky synchronizace, zobrazení vytvořeného XML souboru a načtená data z XML po reinstalaci aplikace.



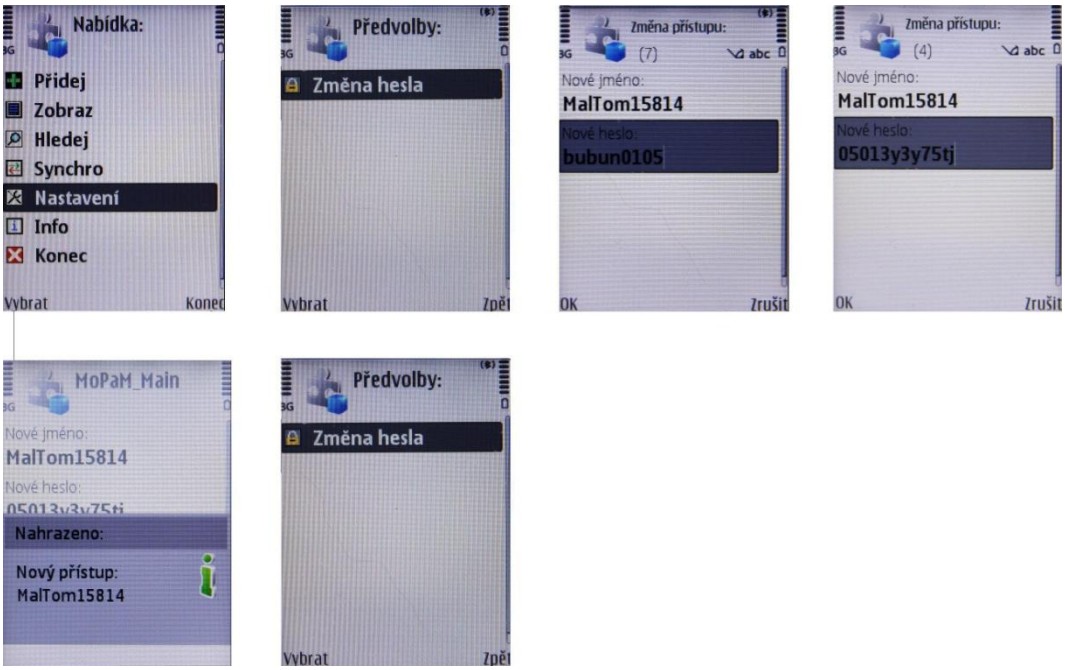
Obrázek 29 - Export záznamů do XML a import po reinstalaci

5.4 Možnosti nastavení

V menu nastavení byla zprovozněna pouze možnost změny přístupových údajů. Původní záměr byl takový, že aplikace nabídne uživateli i možnost změnit šifrovací klíč pro algoritmus AES. Nakonec od toho bylo upuštěno, protože touto změnou by se existující záznamy staly nečitelnými. Teoretickým řešením by byla implementace metody, která by měla na starosti automatické dešifrování všech záznamů za pomoci původního klíče a jejich následné zašifrování klíčem novým. Rizikem by však bylo, že „slabší“ mobilní telefony by takovou operaci nemuseli být schopné provést. Mohlo by tak dojít k fatálním následkům v podobě poškození nebo ztráty uložených dat. Z tohoto důvodu byl záměr implementovat tuto funkci anulován.

5.4.1 Změna přístupových údajů

Jako hlavní možnost nastavení aplikace byla implementována změna nastavení přístupových údajů (zadaných po instalaci a vytvoření hlavního účtu aplikace). K prvnímu záznamu v RMS nelze běžně (prohlížením ani vyhledáváním) přistupovat. Proto byla tato možnost zařazena do nastavení. Po zvolení této volby je na displeji zobrazen formulář s původními údaji (původním uživatelským jménem a heslem). Po úpravě lze nové údaje uložit na místo těch původních. Tím dojde k přepsání prvního záznamu v RMS a poté se lze do aplikace přihlásit pouze pod nově zadaným uživatelským jménem a s novým heslem. Pokud by byla volba uložení vyvolána s původními údaji, dojde k jejímu zrušení a vyvolání hlášení, že údaje nebyly upraveny (jako při úpravě ostatních běžných záznamů). Úprava záznamu je anulována a platné zůstávají původní přihlašovací údaje. Obrázek 30 demonstruje změnu přístupových údajů.



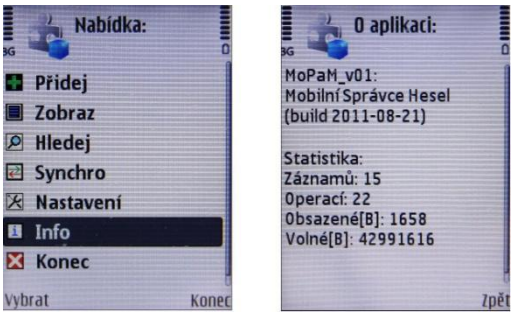
Obrázek 30 - Změna přístupových údajů

Následuje ukázka kódu metody pro úpravu přihlašovacího záznamu.

```
public String upravPristupZaznam(ZaznamMPM nZaznam) {
…
boolean vysledek;
ZaznamMPM puvZaz = vratPristupZaznam();
if ((puvZaz.getLogin().equals(nZaznam.getLogin()))
&& puvZaz.getHeslo().equals(nZaznam.getHeslo()))
        return "Není nutné upravovat – nový záznam je stejný.";
else
vysledek = upravZaznam(1, nZaznam);
…
if (vysledek == true)
        return "Přístupový záznam změněn: " + nZaznam.ToString();
else
        return "Neupraveno";
…
}
```

5.5 Informace o aplikaci

Tato volba vyvolá na obrazovku zařízení formulář s informacemi o aplikaci (verze a datum vytvoření), důležitá data o využití paměti a počtu provedených operací nad používaným RMS. K dispozici jsou údaje o celkovém počtu záznamů (kde ten první je přihlašovací - běžných uživatelských záznamů je o jeden méně). Dále je poskytnuta informace o počtu provedených operací s RMS od nainstalování aplikace (počítají se operace přidání, vymazání i úpravy záznamů). Dalšími dvěma (neméně důležitými) údaji jsou počty obsazených bytů v paměti a počty volných bytů pro uložení dalších záznamů. Z toho lze odhadnout, kolik paměti přibližně zabírá jeden záznam a také kolik jich lze do RMS aplikace ještě vložit. Obrázek 31 ukazuje detailní výpis informací o aplikaci.



Obrázek 31 - Informace o aplikaci

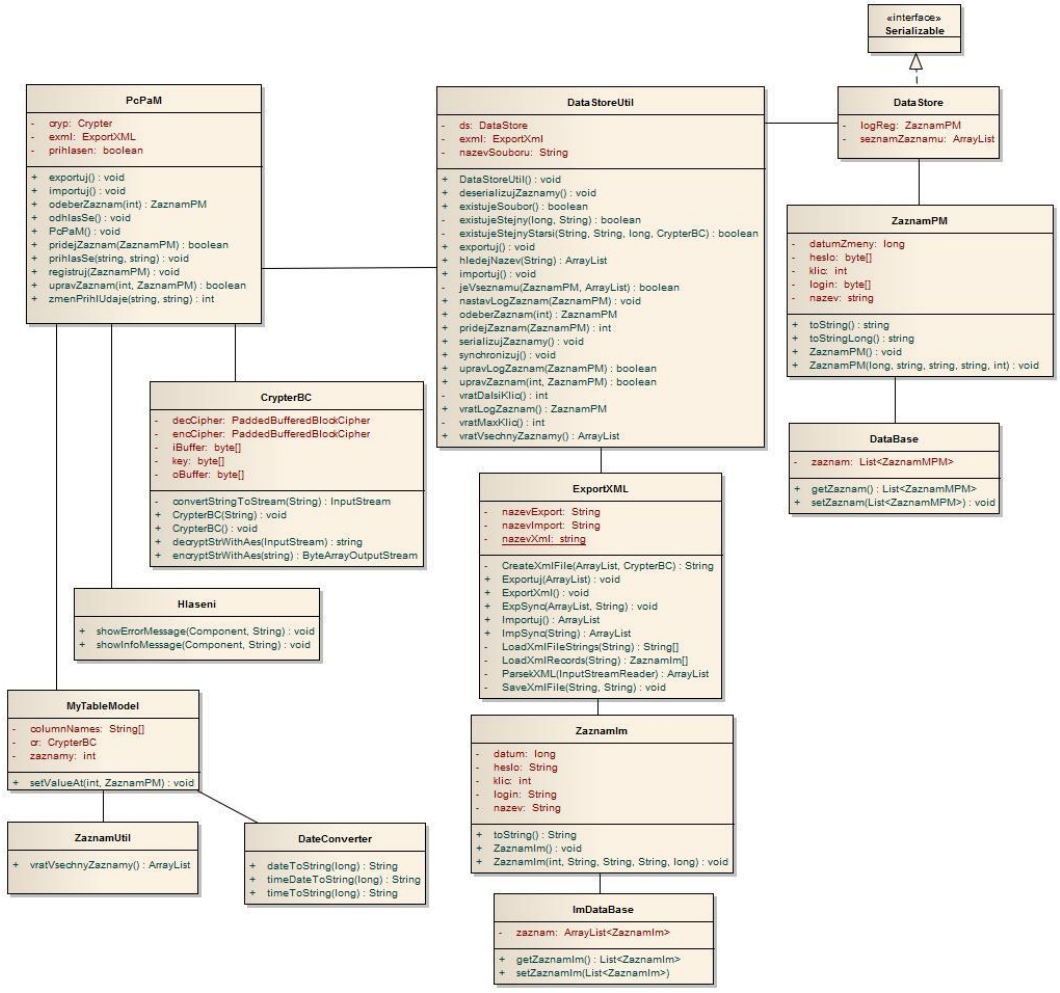
Následuje zkrácená ukázka kódu pro výstup požadovaných informací o stavu paměti.

```
public int[] vratStavDat() {          // kde rs je instance RecordStore
int[] out = new int[3];
…
out[0] = rs.getVersion();             // počet úprav
out[1] = rs.getSize();                // obsazeno bajtů
out[2] = rs.getSizeAvailable();       // ještě možno obsadit
…
return out;
```

6 Aplikace pro PC

V následujícím textu je uveden návrh a základní popis řešení aplikace pro PC. Určená je pro ten samý účel jako aplikace mobilní (uchovávání databáze přístupových údajů v zašifrované podobě a umožnění synchronizace dat s mobilní aplikací).

6.1 Diagram tříd počítačové aplikace



Obrázek 32 - Diagram tříd počítačové aplikace

6.2 Třídy aplikace a jejich důležité metody

V následujících odstavcích byly popsány nejdůležitější třídy počítačové aplikace.

6.2.1 Datová struktura záznamu PC aplikace

Datová struktura záznamu `ZaznamPM` je (oproti datové struktuře záznamu mobilní aplikace) mírně odlišná. Řetězcové datové složky uživatelského jména a hesla byly nahrazeny datovými složkami typu `byte[]`.

6.2.2 Třída CrypterBC

Podobně jako u aplikace pro mobilní zařízení je i u počítačové aplikace třída `CrypterBC` vybavena metodami implementujícími šifrování pomocí algoritmu AES. Na rozdíl od mobilní třídy stejného názvu však převádí vstupní řetězce na pole bytů (vytváří šifrované datové složky ukládané v objektech typu `ZaznamPM`) a naopak. K implementaci byly využity stejné knihovny jako pro mobilní aplikaci. Nádledující ukázka kódu demonstruje použití `Bouncy Castle API`, při implementaci šifrovací metody `encryptStrWithAes()`.

```
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.ShortBufferException;
import org.bouncycastle.crypto.DataLengthException;
import org.bouncycastle.crypto.InvalidCipherTextException;
import org.bouncycastle.crypto.engines.AESEngine;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
...
public class CrypterBC {
...
public ByteArrayOutputStream encryptStrWithAes(String inRetezec) throws Exception{
InputStream is = convertStringToStream(inRetezec);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
try {
       int pvb = 0;   //počet vstupních bajtů
       int pzb = 0;   //počet zpracovaných
       while ((pvb = is.read(iBuffer)) >= 0) {
                pzb = encCipher.processBytes(iBuffer, 0, pvb, oBuffer, 0);
                baos.write(oBuffer, 0, pzb);
                }
       pzb = encCipher.doFinal(oBuffer, 0);
```

```
        baos.write(oBuffer, 0, pzb);
        baos.flush();
        return baos;
} catch (IOException e) {
return null;
}
}
```

6.2.3 Třída MyTableModel

Nutnost implementace této třídy byla dána využitím grafické komponenty `JTable` pro výpis záznamů. Je odvozena od třídy `AbstrackTableModel` a implementuje některé její základní tabulkové metody. Jako hlavní datovou složku má seznam prvků typu `ZaznamPM`. Instance této třídy byla použita pro plnění tabulky hlavního formuláře.

6.2.4 Třída ExportXml

Třída `ExportXml` má za úkol provádět všechny akce spojené s exportem, importem, vytvářením, ukládáním a parsováním XML souborů. Její hlavní využití je pro zálohování a synchronizaci dat s mobilní aplikací.

6.2.5 Třídy ZaznamIm a ImDatabase

Třídy `ZaznamIm` a `ImDatabase` jsou pouze pomocné třídy sloužící k ukládání dat do objektových struktur (při importování záznamů z XML souborů určených pro synchronizaci).

6.3 Ukládání dat

6.3.1 Třídy DataStore a DataStoreUtil

Objekt třídy `DataStore` je datovou strukturou udržující všechna data aplikace. Jako datové složky má v sobě obsažen hlavní přihlašovací záznam a seznam běžných záznamů typu `ZaznamPM`.

Třída `DataStoreUtil` je strukturou, která disponuje implementací všech obslužných metod pro práci s daty uloženými v objektu typu `DataStore`. Obsahuje implementace metod pro přidávání, editaci, odebírání, hledání, exportování a synchronizaci dat aplikace. Ukládání dat bylo vyřešeno použitím binárního souboru a objektové serializace. Následující ukázka kódu naznačuje způsob serializace záznamů do binárního souboru pomocí metod objektové serializace.

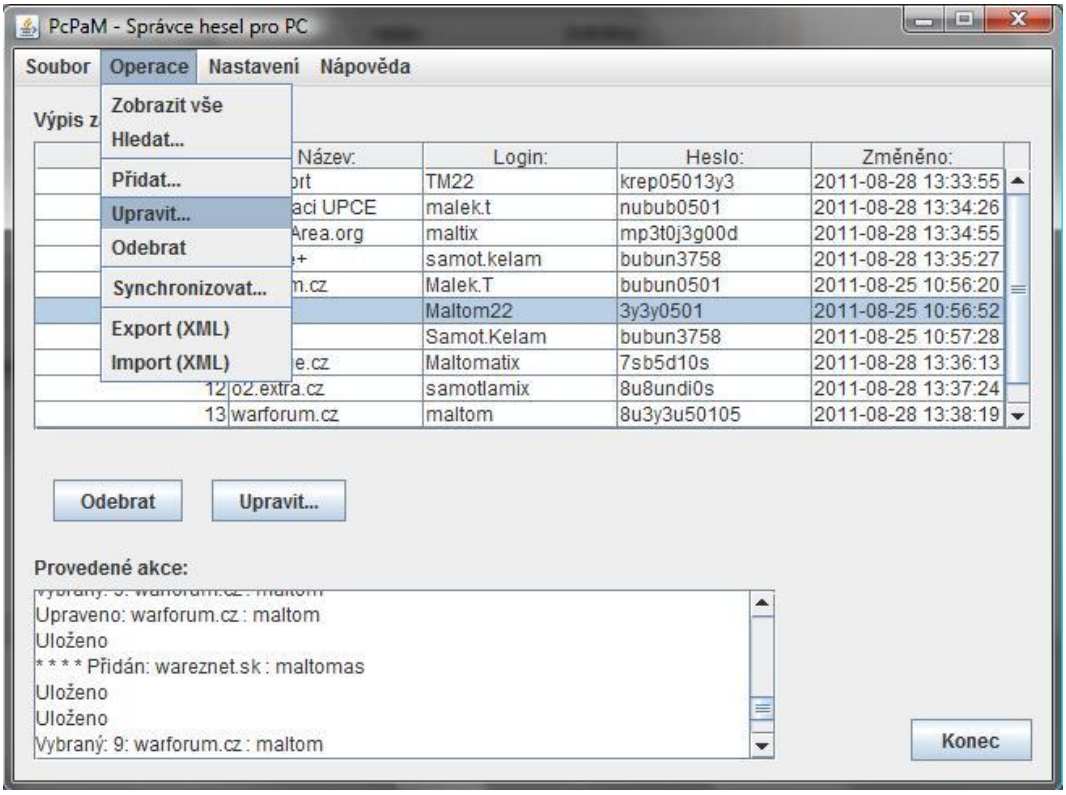
```
public void SerializujZaznamy() throws IOException {
ObjectOutputStream zapis = null;
…
zapis = new ObjectOutputStream(new FileOutputStream(nazevSouboru, false));
zapis.writeObject(ds);        // ds - objekt typu DataStore
…

public void DeserializujZaznamy() throws IOException {
ObjectInputStream cteni = null;
…
cteni = new ObjectInputStream(new FileInputStream(nazevSouboru));
ds = (DataStore) cteni.readObject();
…
```

6.4 Uživatelské rozhraní počítačové aplikace

Hlavní řídící třídou desktopové aplikace je třída `PcPaM`, která je potomkem třídy `JFrame`. Všechny metody této třídy jsou určeny pro ovládání uživatelského rozhraní, vyvolávání uživatelských dialogů pro editaci apod. Uživatelské rozhraní počítačové aplikace bylo realizováno jako klasické formulářové okno o pevně určených rozměrech (maximalizace je zakázána). Po spuštění lze zobrazit pouze

omezená nabídka menu (dokud se uživatel nepřihlásí). Nepřihlášený uživatel má práva otevřít rychlou nápovědu, zobrazit informace o verzi aplikace, přihlásit se nebo aplikaci ukončit. Po přihlášení jsou (díky obsluhující funkci) odemčeny další dvě skupiny menu - `Operace` a `Nastavení`. V nastavení lze změnit přístupové údaje k aplikaci. Menu Operace je obsáhlejší. Horní část uživatelského rozhraní tvoří výpis seznamu záznamů implementovaný pomocí tabulky typu `JTable`. Volby menu a tlačítka pro úpravu i odebrání záznamu jsou aktivována, pouze pokud je v seznamu označen záznam.



Obrázek 33 - Uživatelské rozhraní počítačové aplikace

6.5 Synchronizace s mobilní aplikací

K účelu výměny dat mezi mobilní a počítačovou aplikací je využito propojení pomocí USB kabelu. V počítačové aplikaci jsou metody pro parsování a ukládání exportního XML obsaženy ve třídě `ExportXml`. Základní synchronizace dat byla navržena pomocí využití paměťové karty mobilního telefonu (pro uložení XML).

6.6 Zálohování

Kromě obsluhy synchronizačních operací (k importu a exportu) disponuje třída `ExportXml` i implementací metod pro zálohování. Zálohování a případné další obnovení dat aplikace bylo realizováno formou serializace uložených záznamů do souboru typu XML. Pro serializaci bylo použito třídy `DataBase`, která má (v tomto případě) jako datovou složku seznam záznamů:

```
private List<ZaznamPM> zaznam = new ArrayList<ZaznamPM>();
```

Třída `DataBase` má nastaven následující příznak zajišťující, že ve výsledném dokumentu XML je použit název této třídy jako kořenový element:

```
@XmlRootElement
```

Následující ukázka kódu znázorňuje použití XML serializace a deserializace pro zálohování dat aplikace s využitím metod z balíku `javax.xml.bind`.

```
public void Exportuj(ArrayList seznamZaznamu) throws Exception {
        DataBase db = new DataBase();
        int pocet = seznamZaznamu.size();
        for (int i = 0; i < pocet; i++) {
            db.getZaznam().add((ZaznamPM) seznamZaznamu.get(i));
        }
        JAXBContext con = JAXBContext.newInstance(DataBase.class);
        Marshaller mar = con.createMarshaller();
        mar.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        mar.marshal(db, new FileWriter(getNAZEV_XML()));
    }
    public ArrayList Importuj() throws Exception {
        ArrayList vystSeznam;
        DataBase db = new DataBase();
        try {
            JAXBContext con = JAXBContext.newInstance(DataBase.class);
            Unmarshaller unm = con.createUnmarshaller();
            db = (DataBase) unm.unmarshal(new FileReader(getNAZEV_XML()));
            int pocetVr = db.getZaznam().size();
            vystSeznam = new ArrayList();
            for (int i = 0; i < pocetVr; i++) {
                vystSeznam.add(i, db.getZaznam().get(i));
            }
```

```
            return vystSeznam;
        } catch (Exception e) {
            ArrayList vp = new ArrayList();
            ZaznamPM ch = new ZaznamPM();
            ch.setKlic(1);
            ch.setNazev(e.getLocalizedMessage());
            vp.add(1, ch);
            return vp;
        }
    }
```

Následující ukázka části souboru demonstruje zašifrované řetězce (uživatelské jméno a heslo) uložené ve struktuře záznamů serializovaných do formátu XML. Jako uživatelské jméno tohoto záznamu byl použit řetězec `st15481` a jako fiktivní heslo pak `605bea9d82tr63`.

```
</dataBase>
...
    <zaznam>
        <klic>16</klic>
        <nazev>Intranet UPCE</nazev>
        <login>XQnX+5fgbj7eQ9dVvR38kA==</login>
        <heslo>JV4TJ1tBWmjKbYcAzgYHHQ==</heslo>
        <datumZmeny>1314795697442</datumZmeny>
    </zaznam>
...
</dataBase>
```

7 Závěr

Byla vytvořena aplikace pro mobilní zařízení a počítačová aplikace určená k synchronizaci záznamů. Konečná verze mobilní aplikace byla úspěšně otestována na telefonech značky Nokia - konkrétně na Nokia E51 s operačním systémem Symbian S60 3rd Ed. s rozšířením Feature Pack 1 a Nokia 6720 Classic s operačním systémem Symbian S60 5th Ed. Na obou telefonech byla aplikace stabilní a neprojevil se žádný problém. Počítačová aplikace byla vyvíjena a testována na Notebooku ASUS M50VC s OS MS Windows Vista Home Premium SP2. Dále byla otestována také na stolním počítači s OS MS Windows 7 Profesional. Oba OS byly v 32 bitových verzích. Jediným potřebným doplňkem (který tato aplikace potřebovala pro běh) bylo běhové prostředí Java Runtime Environment ve verzi JRE 6 Update 27 dostupné z [19].

Vzhledem k realizované struktuře záznamu a způsobu použité synchronizace nelze automaticky odebírat z databáze záznamy, které na druhém zařízení již odebrány byly. Synchronizace je tak neúplná. Možným řešením je přidat do struktury záznamů další datovou složku udávající datum odstranění. Potom by však položky zůstávali (i když označené jako neplatné) v databázi zařízení a vzrostly by tak nároky na paměťový prostor.

Jako další zdokonalení mobilní aplikace by bylo možno navrhnout například doplnění midletu o možnosti zvukových či vibračních signálů směrem k uživateli (v případě chyby nebo nutnosti upozornění na jinou událost). Dále by bylo např. vhodné doplnit aplikaci o generátor náhodných silných hesel včetně možnosti vygenerované heslo ihned využít při tvorbě nového vkládaného záznamu. Další užitečné funkcionality by se daly navrhnout i implementovat. Výkonové požadavky na běh aplikace by se však mohly zvýšit. Přece jen se jedná o aplikaci určenou pro běh na zařízení s omezenými prostředky a výkonem.

Z hlediska vylepšení desktopové aplikace by se jistě našla řada možných vylepšení. Vhodné by při nejmenším bylo navrhnout a implementovat další způsob synchronizace s využitím bezdrátové technologie (např. Bluetooth).

Použité zdroje

[1] TOPLEY, Kim. *J2ME v kostce : Pohotová referenční příručka*. 1. Vyd. Praha : Grada Publishing, 2004. 519 s. ISBN 80-247-0426-9.

[2] KISZKA, Bogdan. 1001 tipů a triků pro jazyk Java. Vyd. 2. Brno : Computer Press, 2009. 542 s. ISBN 978-80-251-2467-3.

[3] GROŠEK, Otakar; PORUBSKÝ, Štefan. Šifrovanie - algoritmy, metódy, prax. Praha : Grada, 1992. 268 s. ISBN 80-85424-62-2.

[4] Doseděl, Tomáš, 1980-. Počítačová bezpečnost a ochrana dat. Tomáš Doseděl. Vyd. 1. Brno : Computer Press, 2004. 190 s. ISBN 80-251-0106-1.

[5] Piper, F. C. (Frederick Charles) , 1940-, Murphy, Sean, 1963-. Kryptografie : průvodce pro každého. Fred Piper, Sean Murphy ; [z anglického originálu ... přeložil Pavel Mondschein]. 1. vyd. v českém jazyce. Praha : Dokořán, 2006. 157 s. ISBN 80-7363-074-5 (váz.).

[6] SINGH, Simon; KOUBSKÝ, Petr; ECKHARDTOVÁ, Dita. Kniha kódů a šifer : tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v českém jazyce. Dokořán : Argo, 2009. 382 s. ISBN 978-80-7363-268-7.

[7] *Interval.cz* [online]. 02. 12. 2003 [cit. 2011-08-28]. J2ME pro pokročilé - XML. Dostupné z WWW: <http://interval.cz/clanky/j2me-pro-pokrocile-xml/>.

[8] Advanced Encryption Standard. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 10 November 2001, last modified on 26 August 2011 [cit. 2011-08-26]. Dostupné z WWW: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[9] The Java Community Process (SM) Program [online]. c2011 [cit. 2011-08-27]. Java Specification Requests. Dostupné z WWW: <http://www.jcp.org/en/jsr/platform>.

[10] *Nokia Developer* [online]. March 23rd, 2011 [cit. 2011-08-28]. Forum Nokia Library. Dostupné z WWW: <http://library.developer.nokia.com/>.

[11] *NetBeans IDE* [online]. c2011 [cit. 2011-08-28]. NetBeans Org. Dostupné z WWW: <http://netbeans.org/index.html>.

[12] *Oracle.com* [online]. 2011 [cit. 2011-08-28]. Java SE Downloads. Dostupné z WWW: <http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html>.

[13] *Nokia Developer* [online]. c2011 [cit. 2011-08-28]. Symbian SDKs. Dostupné z WWW: <http://www.developer.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html>.

[14] *The Legion of the Bouncy Castle* [online]. c2011 [cit. 2011-08-28]. Latest Java Releases. Dostupné z WWW: <http://www.bouncycastle.org/latest_releases.html>.

[15] BRUCE, Schneier. Applied Cryptography : Protocols, Algorithms, and Source Code in C. 2nd edition. New York : Wiley Publishing, 1996. 758 s. ISBN 978-0471117094.

[16] FERGUSON, Niels; SCHNEIER, Bruce; KOHNO, Tadayoshi. Cryptography Engineering : Design Principles and Practical Applications. 1 edition. Indianapolis, Indiana : Wiley Publishing, 2010. 384 s. ISBN 978-0-470-47424-2.

[17] PAAR, Christof; PELZL, Jan; PRENEEL, Bart. Understanding Cryptography : A Textbook for Students and Practitioners. 2nd Printing edition. Heidelberg : Springer-Verlag, 2010. 390 s. ISBN 978-3642041006.

[18] FERGUSON, Niels; SCHNEIER, Bruce. Practical Cryptography. 1 edition. Indianapolis, Indiana : Wiley Publishing, 2003. 432 s. ISBN 978-0471223573.

[19] *Oracle.com* [online]. 2011 [cit. 2011-08-29]. Java SE Downloads. Dostupné z WWW: < http://www.oracle.com/technetwork/java/javase/downloads/jre-6u27-download-440425.html>.

[20] Symbian OS. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 24. 2. 2006, last modified on 14. 8. 2011 [cit. 2011-08-30]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Symbian_OS>.