

# Šifrování dat na mobilních zařízeních Android

Data Encryption on Mobile Android Devices

Bc. Milan Oulehla

---

Diplomová práce  
2013



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2012/2013

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Milan Oulehla**  
Osobní číslo: **A11838**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**  
Forma studia: **kombinovaná**

Téma práce: **Šifrování dat na mobilních zařízeních Android**

Zásady pro vypracování:

1. Specifikujte teoretická východiska řešené problematiky.
2. Popište charakteristiku současných technologií kryptografie pro mobilní platformu Android.
3. Specifikujte omezení kryptografie na mobilních platformách.
4. Navrhněte kryptografickou aplikaci pro mobilní platformu Android.
5. Definujte problematiku přenosu šifrovaných souborů mezi mobilními zařízeními s operačním systémem Android.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. MURPHY, M. L. Android 2: Průvodce programováním mobilních aplikací. 1. vyd. Brno: Computer Press, 2011. 375 s. ISBN 978-80-251-3194-7.
2. UJBÁNYAI, M. Programujeme pro Android. 1. vyd. Praha: Grada, 2012. 187 s. ISBN 978-80-247-3995-3.
3. SPELL, B. Java: Programujeme profesionálně. 1. vyd. Praha: Computer Press, 2002. 1022 s. ISBN 80-7226-667-5.
4. VLČEK, K. Teorie informace, kódování a kryptografie. 1. vyd. Ostrava: VŠB-Technická univerzita, 1999. 182 s. ISBN 80-7078-614-0.
5. JAŠEK, R. Informační a datová bezpečnost. 1. vyd. Zlín: Univerzita Tomáše Bati ve Zlíně, 2006. 140 s. ISBN 80-7318-456-7.
6. MARTÍNEK, P. Základy teoretické informatiky. 1. vyd. Olomouc: Univerzita Palackého v Olomouci, 2006. 92 s.
7. BRAGA, A. M., NASCIMENTO, E. N. Portability evaluation of cryptographic libraries on Android smartphones. Cyberspace Safety and Security. Lecture Notes in Computer Science. 7672 volume, 2012, pp 459-469. ISSN 0302-9743 (Print). ISSN 1611-3349 (Online).
8. WANG, Z., MURMURIA, R., STAVROU, A. Implementing and optimizing an encryption filesystem on Android. In: Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management (mdm 2012). Washington, DC: IEEE Computer Society, 2012, pp. 52-62. ISBN 978-0-7695-4713-8.

Vedoucí diplomové práce:

**Ing. David Malaník, Ph.D.**

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

**22. února 2013**

Termín odevzdání diplomové práce:

**22. května 2013**

Ve Zlíně dne 22. února 2013

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



doc. Mgr. Roman Jašek, Ph.D.  
*ředitel ústavu*

## ABSTRAKT

Diplomová práce se zabývá šifrováním dat na mobilních zařízeních Android. V teoretické části práce je uvedena historie šifrování a druhy symetrického a asymetrického šifrování. Pro jednotlivé typy šifrování jsou uvedeny praktické příklady, na kterých je prezentován celý postup šifrování a dešifrování. Dále jsou vysvětleny matematické základy útoku na RSA. Praktická část diplomové práce se věnuje charakteristice a omezením současných technologií kryptografie pro mobilní platformu Android. Následně je v práci navržena originální kryptografická aplikace pro mobilní platformu Android, která respektuje moderní požadavky ochrany dat. Následně jsou definovány požadavky a podmínky přenosu šifrovaných souborů mezi mobilními zařízeními s operačním systémem Android.

Klíčová slova: Android, šifrování, dešifrování, kryptografie, mobilní platforma, kryptografická aplikace

## ABSTRACT

This thesis deals with data encryption on mobile devices Android. The theoretical part deals with the history and types of symmetric and asymmetric encryption. For each type of encryption are given practical examples on which the procedure is presented encryption and decryption. The following explains the mathematical foundations attack on RSA. The practical part describes the characteristics and limitations of current technologies cryptography for mobile platform Android. Subsequently, the work proposed by the original cryptographic applications for the Android mobile platform, which is consistent with the modern requirements of data protection. Following are the terms and conditions defined transfer encrypted files between mobile devices running Android.

Keywords: Android, encryption, decryption, cryptography, mobile platform cryptographic applications

### Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Davidovi Malaníkovi, Ph.D. za odborné vedení, cenné rady a podněty, které mi poskytl při zpracování této práce. V neposlední řadě své partnerce a rodině za trpělivost, zázemí a podporu během studia.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl jsem seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>9</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>10</b>
<b>1 TEORETICKÁ VÝCHODISKA PROBLEMATIKY ŠIFROVÁNÍ</b> .....	<b>11</b>
1.1 HISTORIE.....	11
1.1.1 Steganografie.....	11
1.1.2 Kryptografie .....	12
1.2 NEDOSTATEČNOST HISTORICKÝCH ŠIFER PRO MODERNÍ ÚČELY.....	15
1.2.1 Matematický základ monoalfabetických substitučních šifer .....	15
1.2.2 Frekvenční analýza.....	15
1.2.3 Příklad s reálným textem.....	17
1.3 TEORETICKÝ ZÁKLAD MODERNÍCH ŠIFROVACÍCH ALGORITMŮ .....	22
1.3.1 Počet provedených elementárních kroků .....	23
1.3.2 Řádové porovnání funkcí .....	23
1.3.3 Klasifikace časové složitosti z pohledu delšího období.....	27
1.4 MODERNÍ KRYPTOGRAFIE .....	29
1.4.1 Symetrická kryptografie.....	29
1.4.2 Asymetrická kryptografie.....	34
<b>2 CHARAKTERISTIKA SOUČASNÝCH TECHNOLOGIÍ KRYPTOGRAFIE PRO MOBILNÍ PLATFORMU ANDROID</b> .....	<b>40</b>
2.1 JAVA CRYPTOGRAPHY ARCHITECTURE (JCA) A JAVA CRYPTOGRAPHY EXTENSION (JCE) .....	40
2.2 SYMETRICKÁ KRYPTOGRAFIE.....	41
2.2.1 Třída SecretKeySpec.....	42
2.2.2 Třída Cipher .....	43
2.2.3 Padding neboli dorovnání bajtů .....	47
2.2.4 Režimy blokových šifer .....	51
2.2.5 Symetrická kryptografie založená na heslech .....	55
2.2.6 Rozdíly v šifrování a dešifrování .....	57
2.3 ASYMETRICKÁ KRYPTOGRAFIE .....	59
2.3.1 Výroba veřejného a soukromého klíče.....	59
2.3.2 RSA šifrování.....	61
2.3.3 RSA dešifrování .....	62
2.4 SOLENÍ SOUBORŮ .....	63
2.4.1 Sůl je stejně velká jako soubor k posolení .....	64
2.4.2 Sůl a soubor k posolení jsou různě velké .....	68
<b>II PRAKTICKÁ ČÁST</b> .....	<b>72</b>
<b>3 OMEZENÍ KRYPTOGRAFIE NA MOBILNÍCH PLATFORMÁCH</b> .....	<b>73</b>
3.1 OMEZENÍ DANÉ SOUČASNOU STRUKTUROU HARDWARE NA TRHU .....	73
3.2 OMEZENÍ VYCHÁZEJÍCÍ Z VLASTNOSTÍ OPERAČNÍHO SYSTÉMU ANDROID.....	75
3.3 ÚPRAVA ŠIFROVÁNÍ A DEŠIFROVÁNÍ.....	82
3.4 VÝSLEDNÉ ŘEŠENÍ S UKÁZKOU KÓDU.....	82
<b>4 NÁVRH KRYPTOGRAFICKÉ APLIKACE PRO MOBILNÍ PLATFORMU ANDROID</b> .....	<b>86</b>

4.1	NÁSTROJE PRO VÝVOJ A UKÁZKA VYTVÁŘENÍ PROGRAMŮ .....	86
4.2	SPUŠTĚNÍ A TESTOVÁNÍ VYVÍJENÝCH APLIKACÍ V ZAŘÍZENÍCH S OPERAČNÍM SYSTÉMEM ANDROID.....	90
4.3	GRAFICKÝ NÁVRH APLIKACE.....	94
4.3.1	Způsoby vytváření GUI a jejich problémy.....	94
4.3.2	Návrh GUI kryptografické aplikace.....	99
4.4	VÝVOJ KOMPONENTY PRO GRAFICKOU PRÁCI SE SOUBORY .....	103
4.4.1	Aktuální situace.....	103
4.4.2	Návrh komponenty pro grafickou práci se soubory a adresáři .....	103
4.5	NÁVRH APLIKAČNÍ LOGIKY .....	116
4.5.1	Úvodní obrazovka .....	116
4.5.2	Správa klíčů.....	117
4.5.3	Mazání souborů klíčů .....	117
4.5.4	Vytváření a ukládání klíčů .....	119
4.5.5	Šifrování.....	124
4.5.6	Dešifrování.....	130
<b>5</b>	<b>PŘENOS ŠIFROVANÝCH SOUBORŮ MEZI MOBILNÍMI ZAŘÍZENÍMI S OPERAČNÍM SYSTÉMEM ANDROID.....</b>	<b>134</b>
5.1	JAVA SECURE CHANNEL.....	134
5.2	POUŽITÍ JSCH VE SVÉM PROGRAMU.....	134
5.3	ZAPOUZDŘENÍ FUNKCIONALITY JSCH DO TŘÍD .....	137
5.4	REGISTRACE.....	138
5.5	ODESÍLÁNÍ SOUBORŮ NA SSHD SERVER .....	147
5.6	STAHOVÁNÍ SOUBORŮ Z SSHD SERVERU .....	151
5.6.1	Zobrazování souborů na vzdáleném serveru.....	151
5.6.2	Stahování souborů zip ze vzdáleného serveru .....	155
5.6.3	Mazání souborů zip ze vzdáleného serveru .....	157
	<b>ZÁVĚR .....</b>	<b>160</b>
	<b>ZÁVĚR V ANGLIČTINĚ.....</b>	<b>163</b>
	<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>166</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>171</b>
	<b>SEZNAM TABULEK.....</b>	<b>175</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>176</b>

## ÚVOD

Informační technologie sehrávají v dnešní době klíčovou roli ve všech oblastech lidské činnosti. Od řízení výrobních procesů přes obchod a služby až po státní správu. Současný trend vývoje naznačuje rozvoj a zvyšování podílu informačních technologií ve společnosti. Soukromé a státní instituce mají svá data běžně dostupná v elektronické podobě. Díky nim generují zisky nebo poskytují služby a informace. Data patří k nehmotným aktivům, která tvoří hodnotu a know-how organizací. Náhlá ztráta obchodních kontaktů, účetních systémů, výrobních postupů může způsobit zásadní existenční problémy postižené organizace.

Na tvorbě firemního informačního systému není nejdražší jeho softwarový vývoj nebo pořízení hardware ale data, které informační systém obsahují. Proto se stále více dostává do popředí potřeba ochrany strategických dat. Například odcizením notebooku s plány nových výrobků nevznikne organizaci škoda pouze ve výši cca 25 000 Kč za hardware. Významnější škoda by nastala, kdyby plány získala konkurence. V těchto případech může škoda překročit i miliony korun. Jako vhodný prostředek logické ochrany dat se osvědčila kryptografie.

Trendem posledních pěti let je pak masivní nástup chytrých telefonů a tabletů zejména do firemní sféry. Vždyť bez chytrých telefonů a tabletů se dnes neobejde téměř žádná obchodní schůzka nebo porada vedení. Bohužel je v současné době nepoměr mezi hodnotou dat uložených ve firemních mobilních zařízeních a jejich ochranou.

Cílem diplomové práce je vytvořit návrh mobilní kryptografické aplikace, která by dokázala poskytnout ochranu dat na lokálním úložišti mobilního zařízení jak před ztrátou či odcizením tak před útoky po síti, kde je situace ještě významnější. Důvodem je, že většině zařízení chybí jakákoliv ochrana a jsou přitom neustále napojeny na internet.

Další důležitou problematiku, kterou řeší diplomová práce, je bezpečná výměna dat mezi mobilními zařízeními. Pokud mají být mobilní zařízení prakticky použitelná, jako jeden z efektivních nástrojů komunikace, je nutné zajistit bezpečný přenos dat.

Práce se věnuje operačnímu systému Android, protože mobilní telefony a tablety s tímto operačním systémem zaujímají největší podíl na trhu.

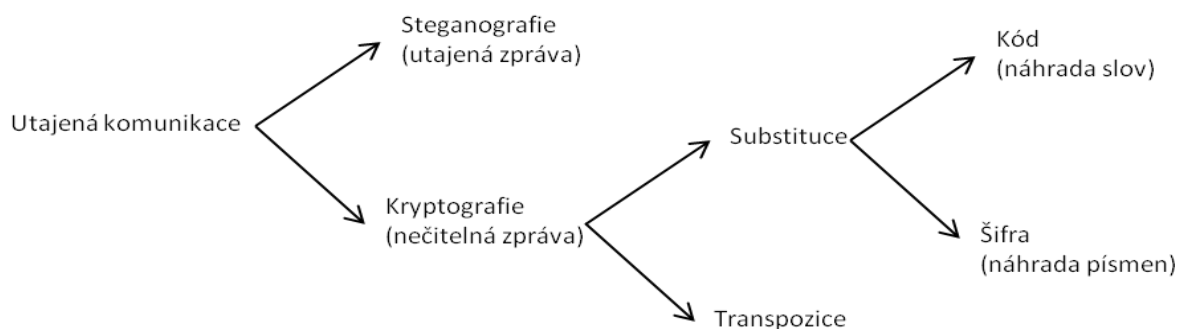
## I. TEORETICKÁ ČÁST

# 1 TEORETICKÁ VÝCHODISKA PROBLEMATIKY ŠIFROVÁNÍ

V této kapitole je uveden historický vývoj a moderní trendy v kryptografii. Pro srozumitelnost je doplněna praktickými příklady šifrování a dešifrování.

## 1.1 Historie

Utajená komunikace není jen doménou moderních telekomunikačních a počítačových sítí, ale je stará jako písmo a posílání zpráv. Neboť vždy bylo potřeba utajit obsah zprávy před nežádoucí osobou. Tou mohl být například velitel nepřátelského vojska nebo král, proti kterému se připravovalo spiknutí. Hlavní podoby utajené komunikace v historickém kontextu jsou zachyceny na obrázku 1.



Obr. 1 Historický vývoj utajené komunikace [1]

Pro komplexní výčet podoborů utajované komunikace je níže uvedena jejich charakteristika.

### 1.1.1 Steganografie

Název steganografie vznikl složením dvou řeckých slov steganós, které znamená skrytý, a gráphein, které znamená psát. Principem je přidání tajné informace skrytým způsobem k otevřené zprávě. Otevřená zpráva může být přečtena kýmkoliv. Pokud je zpráva zachycena neoprávněnou osobou, přečte pouze otevřenou zprávu, aniž by upozorovala, že součástí zprávy je skrytá zpráva. V minulosti byly velmi populární neviditelné inkousty. Aplikace neviditelných inkoustů je založena na napsání standardního dopisu, kdy je mezi řádky dopsán text. Skrytá zpráva je napsána mlékem, cibulovou nebo citrónovou šťávou,

která po zaschnutí není vidět. Při zahřátí dopisu například nad svíčkou skrytý text ztmavne, což umožňuje jeho přečtení. Odtud pochází rčení „číst mezi řádky“.

Je celá řada dalších stenografických technik například mikrotečky. Problematika mikroteček je detailně řešena v literatuře [1]. V dnešní době je digitální verze steganografie populární. Hojně ji využívají třeba špičky organizovaného zločinu, které vkládají utajované zprávy do obrazových souborů. Ty jsou následně posílány jako přílohy e-mailů s nevinným obsahem.

### 1.1.2 Kryptografie

Kryptografie je disciplína, které se zabývá šifrováním a kódováním zpráv. Tento vědní obor vznikl před více než dvěma tisíci lety [2]. Kryptografie se v průběhu své historie vyvíjela a měnila oblasti působení. Od jednoduchých transpozičních a substitučních šifer až k vrcholům analogového období kryptografie jako jsou Vigenèrova šifra či známý šifrovací stroj Enigma. S nástupem výpočetní techniky dochází k revoluci v kryptografii. Do té doby probíhal pozvolný vývoj a kryptografie se týkala úzkých vybraných skupin obyvatelstva, představitelů vojenských a diplomatických elit. Období výpočetní techniky je doprovázeno bouřlivým vývojem ovlivňujícím všechny uživatele počítačů, aniž by o tom někteří věděli. Například automatické aktualizace operačního systému nelze bez kryptografie provádět. Zamezuje se útočníkům možnost vydávat jejich škodlivý kód za aktualizaci operačního systému. Operační systém přijímá pouze aktualizace podepsané důvěryhodnou certifikační autoritou.

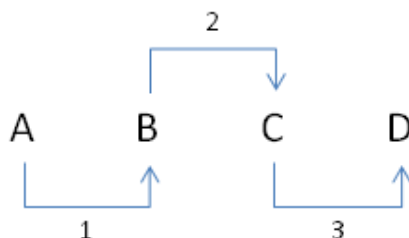
#### Substituční šifrování

Mezi nejpoužívanější historické šifry patří substituční šifry. Při substitučním šifrování se u písmene zachová jeho pozice v textu, ale mění se jeho hodnota. Šifry jsou založené na substituci (nahrazení) množiny písmen otevřené abecedy za jinou množinu symbolů. Symbolem může být písmeno otevřené abecedy, speciální znak navržený pouze pro potřeby šifrování (znaky takzvaných tajných abeced). Šifrovaná abeceda může být tvořena náhodnou změnou pořadí písmen otevřené abecedy. Příklad šifrované abecedy je uveden na obrázku 2.

a	b	c	d	E	f	g	h	i	j	k	l	M	n	o	p	q	r	s	t	u	v	w	x	y	z
j	z	e	R	B	d	f	i	a	y	L	h	C	o	n	m	p	v	g	s	t	u	w	k	q	x

Obr. 2 Příklad šifrované abecedy [zdroj vlastní]

Druhou možností vzniku šifrované abecedy je posunutí vůči otevřené abecedě. Je-li délka posunutí tři, písmeno A je šifrováno na písmeno D. Písmeno B je šifrováno na písmeno E a tak dále. Ukázka šifrování písmene A je zachycen na obrázku 3.



*Obr. 3 Příklad šifrování posunutím písmene A [zdroj vlastní]*

Mezi nejznámější šifrování posunutím patří Césarova šifra, u níž je délka posunutí rovna třem. Všechny výše uvedené metody jsou vytvořeny pomocí jedné šifrovací abecedy a jsou nazývány monoalfabetické substituční šifry. Substituční šifry, ve kterých se kryptogram vytváří pomocí více šifrovacích abeced, jsou označovány jako polyalfabetické. Do této skupiny spadá velice kvalitně sestavená Vigenèrova šifra odolávající útoku hrubou silou (dát do poznámky pod čarou hrubou silu). Avšak pro svou složitost nebyla příliš populární. Poslední skupinou je homofonní šifrování. Jednotlivá písmena otevřené abecedy je možné zašifrovat více písmeny šifrovací abecedy. Tím je znemožněna realizace frekvenční analýzy. Písmenům s velkou relativní četností se přiřazuje větší množství obrazů (šifrovacích písmen) než písmenům s malou relativní četností. Cílem je, aby četnosti všech písmen v zašifrovaném textu byly co nejrovnoměrnější.

### **Transpoziční šifrování**

Další velkou skupinou historických šifer je transpoziční šifrování. Zachovává hodnoty jednotlivých písmen, ale mění jejich pozici podle předem daných pravidel. Nejjednodušší transpoziční šifrou je text napsaný opačně. Dalším často uváděným příkladem je šifra Rail Fence. Jednotlivá písmena se pokládají střídavě mezi řádky, které představují koleje. Způsobů pokládání písmen je více. Na obrázku 4 a 5 jsou zobrazeny dva různé způsoby pokládání písmen na tři koleje.

T				J				J				P				A
	O		O		E		A		N		Z		R		V	
		T				T				A				A		

Obr. 4 Příklad prvního způsobu pokládání písmen na koleje [zdroj vlastní]

T	O	T	N	P	V
O	J	A	A	R	A
T	E	J	Z	A	

Obr. 5 Příklad druhého způsobu pokládání písmen na koleje [zdroj vlastní]

Zápis zašifrovaného textu do zprávy probíhá po řádcích a seskupují se po čtyřech. První způsob: TJJP AOOE ANZR VTAA. Druhý způsob: TOTN PVOJ AARA TEJZ A. Mezi další transpoziční šifry patří Sloupcová transpozice, Dvojitá transpozice a další. Časté je také používání různých mřížek, podle kterých se nejprve zapíše slova otevřeného textu, které se pak doplní libovolným maskovacím textem.

### Kód

Kódování je nahrazování celých slov dohodnutým kódem. Kódy se uchovávají v kódovacích knihách obsahující všechna potřebná slova. Kódové knihy jsou největší slabinou systému. Při jejich kompromitaci se zhroutí celá kódovaná komunikace a musí být vytvořena nová kódová kniha. Je nutné zajistit její distribuci mezi účastníky kódové komunikace. Příklad kódování je uveden na obrázku 6.

...	...
Je	106
...	...
Tajný	502
...	...
Text	520
...	...
Toto	601
...	...

Obr. 6 Příklad kódování [zdroj vlastní]

Kódovaná zpráva vypadá takto: 601\*106\*502\*520.

### Šifra

Při šifrování dochází k nahrazování jednotlivých písmen jinými, jak bylo popsáno výše.

## 1.2 Nedostatečnost historických šifer pro moderní účely

Následující příklad ukazuje, proč se s nástupem výpočetní techniky stala většina starých šifer nepoužitelných.

### 1.2.1 Matematický základ monoalfabetických substitučních šifer

Před samotným řešením příkladu je nutné uvést matematický základ k monoalfabetickým substitučním šifrám.

Dvojice otevřené abecedy a šifrovací abecedy pro jakékoliv monoalfabetické substituční šifrování mohou být obecně chápány jako bijektivní zobrazení z množiny, jejímiž prvky jsou všechna písmena abecedy daného přirozeného jazyka tuto na množinu. Platí  $f: A \rightarrow A$ , kde  $A$  je množina abecedy přirozeného jazyka. Je-li mohutnost množiny  $|A| = a$  pak existuje  $a!$  možných bijekcí. Počet bijekcí byl vypočítán, tak že první znak šifrované abecedy je možné z otevřené abecedy vybrat  $a$  způsoby. Druhý znak šifrované abecedy je možné z otevřené abecedy vybrat  $a - 1$  způsoby, neboť jedno písmeno již bylo vybráno. Třetí znak šifrované abecedy je možné z otevřené abecedy vybrat  $a - 2$  způsoby, neboť dvě písmena již byla vybrána. Shodně se postupuje až k poslednímu znaku, který je možné vybrat pouze jediným způsobem, protože všechna ostatní písmena abecedy již byla vybrána. Použijeme-li pravidlo součinu, dostaneme (1):

$$a (a - 1) (a - 2) (a - 3) \dots \cdot 1 \quad (1)$$

což není nic jiného než permutace a proto platí  $P(a) = a!$ . Pro anglickou abecedu platí, že  $|A| = 26$ , tedy  $a! = 403291461126605635584000000$ . Pro českou abecedu, se všemi diakritickými znaky platí  $|A| = 42$ , tedy  $a! = 1,4050061177528798985431426062445e + 51$  možných šifrovacích abeced. Jistota nelezení správného řešení pro kryptogram, jehož předpokládaným jazykem je anglický jazyk, spočívá v postupném vyzkoušení  $403291461126605635584000000$  možných bijekcí. Tento postup je velice zdlouhavý. Lepším řešením je frekvenční analýza.

### 1.2.2 Frekvenční analýza

Myšlenka frekvenční analýzy je založena na skutečnosti, že relativní četnosti jednotlivých písmen abecedy jsou v přirozeném jazyce dané a příliš se nemění. V anglickém jazyce se nejčastěji vyskytuje písmeno **e**. Pokud písmeno **e** otevřené abecedy bude například nahrazeno písmenem **h** šifrované abecedy, pak bude mít v zašifrovaném textu největší

četnost právě písmeno **h**. V literatuře [3] je uváděna tabulka relativních četností výskytu písmen v anglickém jazyce. Tabulka relativních četností písmen v anglickém jazyce je prezentována v tabulce 1.

*Tab. 1 Tabulka relativních četností písmen v anglickém jazyce [3]*

Písmeno	Relativní četnost anglické abecedy	Písmeno	Relativní četnost anglické abecedy
<b>A</b>	8,2	<b>n</b>	6,7
<b>B</b>	1,5	<b>o</b>	7,5
<b>C</b>	2,8	<b>p</b>	1,9
<b>D</b>	4,3	<b>q</b>	0,1
<b>E</b>	12,7	<b>r</b>	6
<b>F</b>	2,2	<b>s</b>	6,3
<b>G</b>	2	<b>t</b>	9,1
<b>H</b>	6,1	<b>u</b>	2,8
<b>I</b>	7	<b>v</b>	1
<b>J</b>	0,2	<b>w</b>	2,4
<b>K</b>	0,8	<b>x</b>	0,2
<b>L</b>	4	<b>y</b>	2
<b>M</b>	2,4	<b>z</b>	0,1

Tabulka 1 slouží jako základ pro frekvenční analýzu kryptogramů, u kterých se předpokládá, že jsou v anglickém jazyce. Všechny kryptogramy zašifrované pomocí substitučních metod nelze dešifrovat pouze prostým porovnáváním četností. Důvodem může být například:

1. zachycené množství textu je příliš malé, na to aby se projevil statistické charakteristiky;
2. do textu jsou vkládány takzvané nulové znaky, které nepředstavují žádné písmeno;
3. do textu jsou přidávány funkční znaky, které ruší n-tý předchozí nebo n-tý následující znak a narušují četnosti jednotlivých písmen v kryptogramu.

V takových případech je vhodné použít četnosti bigramů (dvojice písmen které jdou po sobě, například slovo ahoj obsahuje bigramy: ah, ho, oj) a trigramů. Počet bigramů je podle pravidla součinu  $a \cdot a = a^2$ , počet trigramů je  $a \cdot a \cdot a = a^3$ . Mezi nejčastější bigramy v anglickém jazyce patří th, he a in. Trigramy jsou nejčastější the, and a ing [4]. V českém jazyce jsou to bigramy st, ní, po a trigramy pro, ost, sta [5].

Tab. 2 Počet bigramů a trigramů v anglickém a českém jazyce [4][5]

Jazyk	Počet bigramů	Počet trigramů
Anglický jazyk	676	17576
Český jazyk	1764	74088

Z tabulky 2 vyplývá, že zanalyzovat zašifrovaný text a ručně spočítat výskyty bigramů a trigramů v delším textu je časově náročné.

### 1.2.3 Příklad s reálným textem

Na příkladu textu článku serveru novin The Telegraph [6] je demonstrována neúčinnost historických šifer při použití výpočetní techniky. Na obrázku 7 je zobrazena část textu novinového článku.

David Cameron: Conservatives will battle for Britain's future

The Conservatives are determined to win the battle for Britain's future, writes David Cameron in the Sunday Telegraph.



'That is what everything this Government does comes back to: the future', writes the PM Photo: Geoff Pugh for the Telegraph

By David Cameron  
9:00PM GMT 02 Mar 2013

There has been a barrage of advice this weekend following the Eastleigh by-election. Some say the Conservative Party should veer right. Some say the Government should abandon the course it's on.

Some say spend more. Some say spend less.

To all of these pieces of advice, my answer is simple. We are engaged in a battle for Britain's future. It is a battle to defeat some of the most dangerous challenges in our history. And it is a battle we will win only if we reject the cynicism, the political calculation and the easy ways out – and stick to the course we are on.

I totally understand the concerns people have. The person reading this who is concerned that when their children leave college they won't be able to find a good job. The parents with frozen wages who feel the bills getting tighter by the month. The father who wants his daughter to grow up in a country where aspiration is real, where she can rise up through

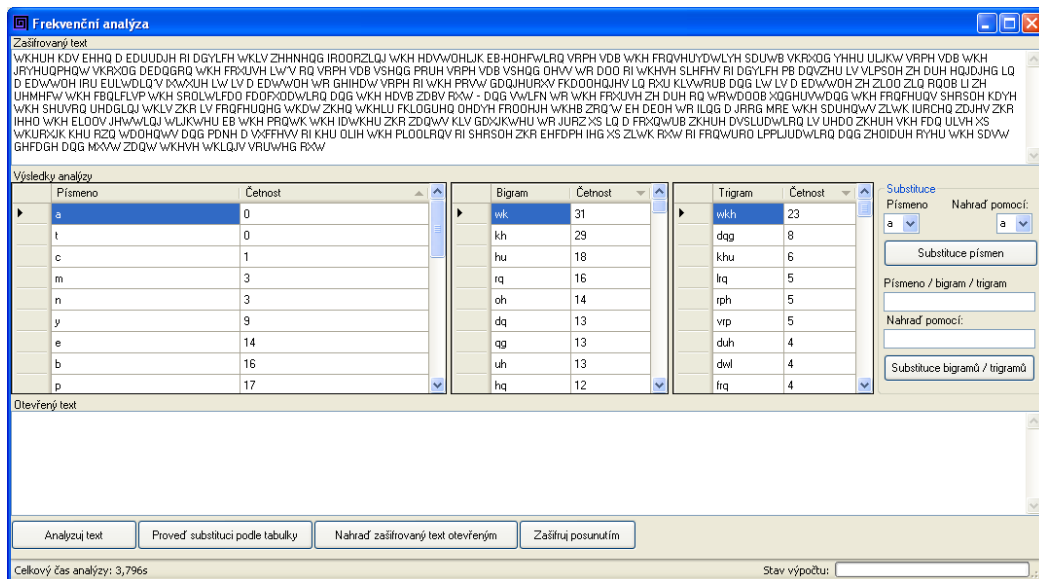
Print this article  
Share 506  
Facebook 391  
Twitter 113  
Email  
LinkedIn 2  
+1 0

David Cameron  
News > Politics >  
Conservative >  
UK News >  
Editor's Picks >

Elsewhere

Obr. 7 Část textu novinového článku [6]

Šifrování celého textu článku je provedeno pomocí programu pro frekvenční analýzu. Program pro frekvenční analýzu byl napsán pro potřeby diplomové práce v jazyce C#. Vzhled programu je zachycen na obrázku 8. Text je zašifrován substituční metodou šifrou posunutí prezentovanou v podkapitole substituční šifrování. Délka posunutí je rovna třem.



Obr. 8 Program pro frekvenční analýzu [zdroj vlastní]

Část textu novinového článku [6]:

THERE HAS BEEN A BARRAGE OF ADVICE THIS WEEKEND FOLLOWING THE EASTLEIGH BY-ELECTION SOME SAY THE CONSERVATIVE PARTY SHOULD VEER RIGHT SOME SAY THE GOVERNMENT SHOULD ABANDON THE COURSE IT'S ON SOME SAY SPEND MORE SOME SAY SPEND LESS TO ALL OF THESE PIECES OF ADVICE MY ANSWER IS SIMPLE WE ARE ENGAGED IN A BATTLE FOR BRITAIN'S FUTURE IT IS A BATTLE TO DEFEAT SOME OF THE MOST DANGEROUS CHALLENGES IN OUR HISTORY AND IT IS A BATTLE WE WILL WIN ONLY IF WE REJECT THE CYNICISM THE POLITICAL CALCULATION AND THE EASY WAYS OUT – AND STICK TO THE COURSE WE ARE ON TOTALLY UNDERSTAND THE CONCERNS PEOPLE HAVE THE PERSON READING THIS WHO IS CONCERNED THAT WHEN THEIR CHILDREN LEAVE COLLEGE THEY WON'T BE ABLE TO FIND A GOOD JOB THE PARENTS WITH FROZEN WAGES WHO FEEL THE BILLS GETTING TIGHTER BY THE MONTH THE FATHER WHO WANTS HIS DAUGHTER TO GROW UP IN A COUNTRY WHERE ASPIRATION IS REAL WHERE SHE CAN RISE UP THROUGH HER OWN TALENTS AND MAKE A SUCCESS OF HER LIFE THE MILLIONS OF PEOPLE WHO BECAME FED UP WITH OUT OF CONTROL IMMIGRATION AND WELFARE OVER THE PAST DECADE AND JUST WANT THESE THINGS SORTED OUT.

Zašifrovaná podoba části článku:

WKHUH KDV EHHQ D EDUUDJH RI DGYLFH WKLV ZHHNHQG IROORZLQJ  
WKH HDVWOHLJK EB-HOHFWLRQ VRPH VDB WKH FRQVHUYDWLYH  
SDUWB VKRXOG YHHU ULJKW VRPH VDB WKH JRYHUQPHQW VKRXOG  
DEDQGRQ WKH FRXUVH LW'V RQ VRPH VDB VSHQG PRUH VRPH VDB  
VSHQG OHVV WR DOO RI WKHVH SLHFHV RI DGYLFH PB DQVZHU LV  
VLPSOH ZH DUH HQJDJHG LQ D EDWWOH IRU EULWDLQ'V IXWXUH LW LV  
D EDWWOH WR GHIHDW VRPH RI WKH PRVW GDQJHURXV FKDOOHQJHV  
LQ RXU KLVWRUB DQG LW LV D EDWWOH ZH ZLOO ZLQ RQOB LI ZH  
UHMHFV WKH FBQLFLVP WKH SROLWLFDO FDOFXODWLRQ DQG WKH  
HDVB ZDBV RXW – DQG VWLFN WR WKH FRXUVH ZH DUH RQ WRWDOOB  
XQGHUVWDQG WKH FRQFHUQV SHRSOH KDYH WKH SHUVRQ UHDLQJ  
WKLV ZKR LV FRQFHUQHG WKDW ZKHQ WKHLU FKLOGUHQ OHDYH  
FROOHJH WKHB ZRQ'W EH DEOH WR ILQG D JRRG MRE WKH SDUHQVW  
ZLWK IURCHQ ZDJHV ZKR IHHO WKH ELOOV JHWLQJ WLJKWHU EB WKH  
PRQWK WKH IDWKHU ZKR ZDQWV KLV GDXJKWHU WR JURZ XS LQ D  
FRXQWUB ZKHUH DVSLUDWLRQ LV UHDO ZKHUH VKH FDQ ULVH XS  
WKURXJK KHU RZQ WDOHQVW DQG PDNH D VXFFHVV RI KHU OLIH WKH  
PLOOLRQV RI SHRSOH ZKR EHFDPH IHG XS ZLWK RXW RI FRQWURO  
LPPLJUDWLRQ DQG ZHOIDUH RYHU WKH SDVW GHFDGH DQG MXVW  
ZDQW WKHVH WKLQJV VRUWHG RXW.

Kroky realizace frekvenční analýzy pomocí výpočetní techniky.

V prvním kroku byl zašifrovaný text článku vložen do vytvořeného programu pro frekvenční analýzu. Byly provedeny výpočty absolutních četností písmen, bigramů a trigramů.

Proces zahrnoval výpočet četností pro 1100 znaků, 676 bigramů a 17576 trigramů zašifrovaného textu. Na běžném počítači (procesor Intel® Core™2 Duo, 2 GB RAM) trvala analýza pouhé 3,796 sekundy. Ve výpočtových algoritmech nebyla provedena žádná optimalizace pro urychlení výpočtu.

Ve druhém kroku byly vypočítané četnosti přidány do relativních četností tabulky 3.

Tab. 3 Vypočtená četnost programem pro frekvenční analýzu [3][zdroj vlastní]

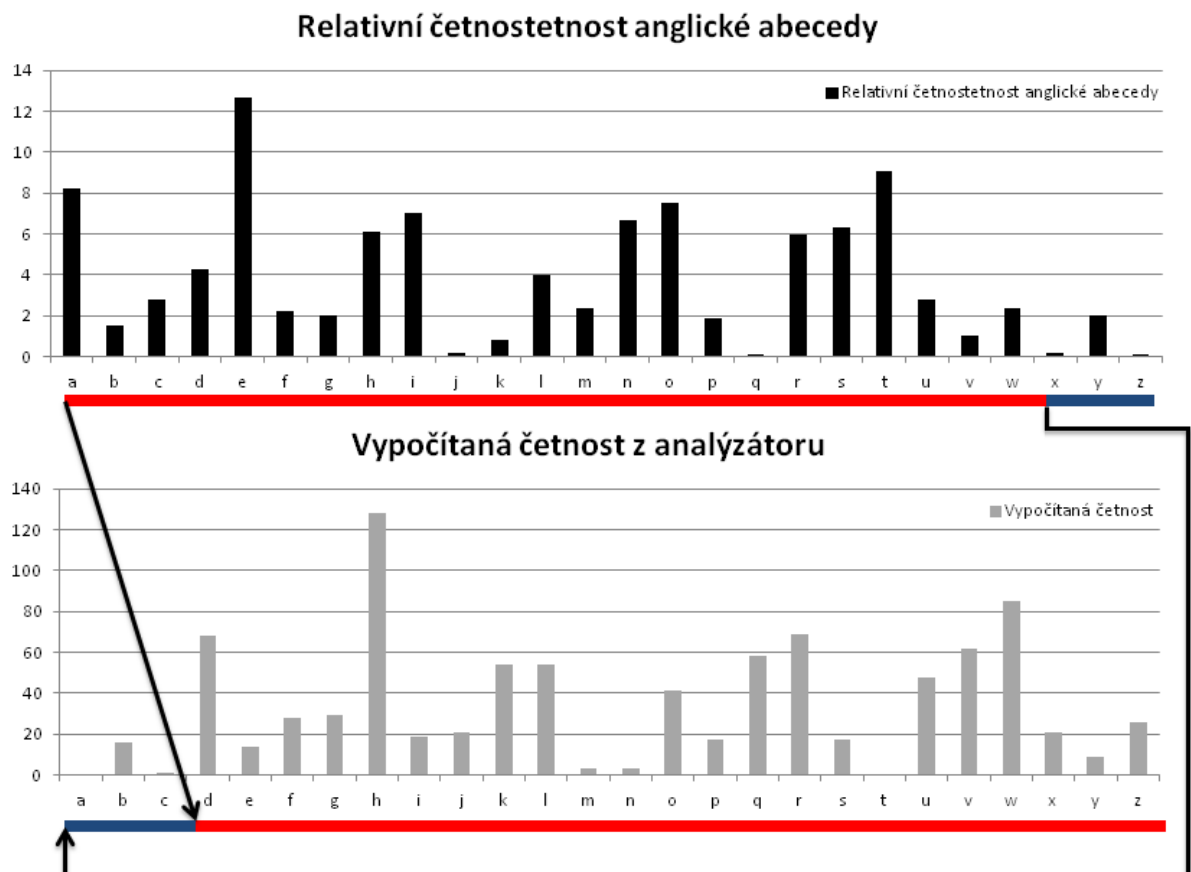
Písmeno	Relativní četnost anglické abecedy	Vypočítaná četnost
<b>a</b>	8,2	0
<b>b</b>	1,5	16
<b>c</b>	2,8	1
<b>d</b>	4,3	68
<b>e</b>	12,7	14
<b>f</b>	2,2	28
<b>g</b>	2	29
<b>h</b>	6,1	128
<b>i</b>	7	19
<b>j</b>	0,2	21
<b>k</b>	0,8	54
<b>l</b>	4	54
<b>m</b>	2,4	3
<b>n</b>	6,7	3
<b>o</b>	7,5	41
<b>p</b>	1,9	17
<b>q</b>	0,1	58
<b>r</b>	6	69
<b>s</b>	6,3	17
<b>t</b>	9,1	0
<b>u</b>	2,8	48
<b>v</b>	1	62
<b>w</b>	2,4	85
<b>x</b>	0,2	21
<b>y</b>	2	9
<b>z</b>	0,1	26

Ve třetím kroku byly z tabulky 3 vykresleny grafy, jak je vidět na obrázku 9.

Při porovnání tvaru červeně označené části horního histogramu na obrázku 7 se shodně potvrzenou částí dolního histogramu je vidět jejich tvarová podobnost. Lze předpokládat, že písmeno **a** je šifrováno pomocí písmene **d**. Písmeno **b** je šifrováno pomocí písmene **e** atd. Porovnání tvaru modře označené části horního histogramu na obrázku 7 se stejně potvrzenou částí v dolním histogramu je vidět jejich tvarová podobnost. Lze předpokládat, že písmeno **x** je šifrováno pomocí písmene **a**. Písmeno **y** je šifrováno pomocí písmene **b** a písmeno **z** je šifrováno pomocí písmene **c**. Z celkového

tvaru histogramů vyplývá, že se jedná o šifrování posunutím, kde délka posunutí je rovna třem. Jde o známou Césarovu šifru.

Provedení analýzy textu článku o tisíci sto znacích a odhalení substituce takto jednoduchého zašifrování lze pomocí výpočetní techniky a vhodně zvolené metody provést za pár minut. Výsledek analýzy je dobrý a nebyly potřeba bigramy a trigramy.

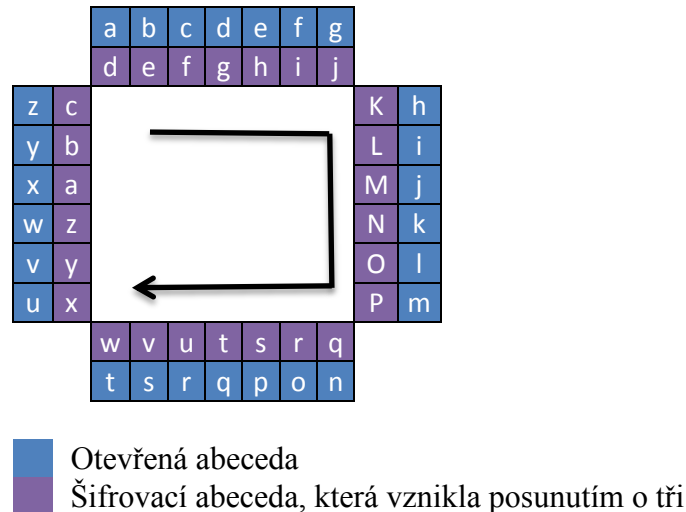


Obr. 9 Histogram výsledků frekvenční analýzy [zdroj vlastní]

Ve své době byly substituční šifry chytře vymyšleny. Ruční procházení a zkoušení 403291461126605635584000000 možných šifrovacích abeced je časově nere realizovatelné. V dnešní době jsou však již nepoužitelné. Jejich tvůrci nepočítali s obrovskou výpočetní silou, kterou disponují dnešní počítače. Z toho vyplývá, že metody/algoritmy používané v moderní kryptografii musí být odolné vůči výpočetnímu výkonu současných i budoucích počítačů.

Na obrázku 10 je vidět příkladem odhalená otevřená a šifrovací abeceda. Obrázek 10 může sloužit jako pomůcka pro vytváření šifrovacích abeced o libovolné délce posunutí. Stačí posouvat vnitřním čtvercem, jak naznačuje šipka. Je zřejmé, že posunutí je možné pouze

v rozsahu 0 (žádné posunutí) až 25. Posunutí 26 je rovno posunutí 0. Výpočet skutečného posunutí  $i$  pro délky větší než je 25 je funkcí modula. Platí tedy  $x \bmod 26$ , kde  $x$  je počet posunutí, pro které platí  $x \in \{z \mid z \in \mathbb{Z} \wedge z \geq 0\}$ .



Obr. 10 Schéma otevřené a šifrovací abecedy [zdroj vlastní]

### 1.3 Teoretický základ moderních šifrovacích algoritmů

Při konstrukci moderních šifrovacích algoritmů se vychází s disciplíny, která se nazývá složitost. Disciplína zvaná složitost se zabývá algoritmicky řešitelnými problémy, u nichž zkoumá výpočetní obtížnost jejich řešení [7]. Složitost se zabývá časovou a paměťovou náročností. Paměťová náročnost není pro účely kryptografie důležitá, proto se bude tento text zabývat pouze časovou náročností, které se také říká časová složitost. Uvažujme nyní tento problém. Je dán algoritmus, u kterého je potřeba posoudit jeho časová náročnost. Je zkoumaný algoritmus časově náročný, pokud jeho vykonání zabere dvacet minut v běžném notebooku? Nebo je časově náročný, pokud poběží dvacet minut na superpočítači či chytrém telefonu? Situace je ještě nepřehlednější, jestliže se vezme v úvahu výpočetní technika, která bude k dispozici například za deset let. Bude nutné pravidelně přehodnocovat časové složitosti všech známých algoritmů? Vyřazovat algoritmy, které se již nepovažují za časově náročné a tedy vhodné pro kryptografické úlohy?

Z výše uvedeného problému je zřejmé, že posuzování časové náročnosti měřením délky běhu algoritmu v časových jednotkách na konkrétním hardwaru je velmi nepraktické

a nevedlo by k dobrým výsledkům. Klasifikace časových složitostí algoritmů musí splňovat požadavky:

1. Klasifikace musí být nezávislá na konkrétních výpočetních prostředcích.
2. Klasifikace musí být stálá v čase. To znamená, že ať se v budoucnu objeví jakkoliv výkonné počítače neovlivní to klasifikaci.

Na první pohled se zdá, že uvedené požadavky nebude možné splnit. Výpočetní prostředky a algoritmy/programy spolu nerozlučně souvisí. Jak by bylo možné je posuzovat nezávisle? To vede k domněnce, že vyřešit tento problém bude velmi náročné a výsledná klasifikace bude složitá a nepřehledná. Opak je však pravdou. Klasifikace časové složitosti je elegantní, přehledná a založena na dvou jednoduchých myšlenkách:

- Počet provedených elementárních kroků.
- Řádové porovnání funkcí.

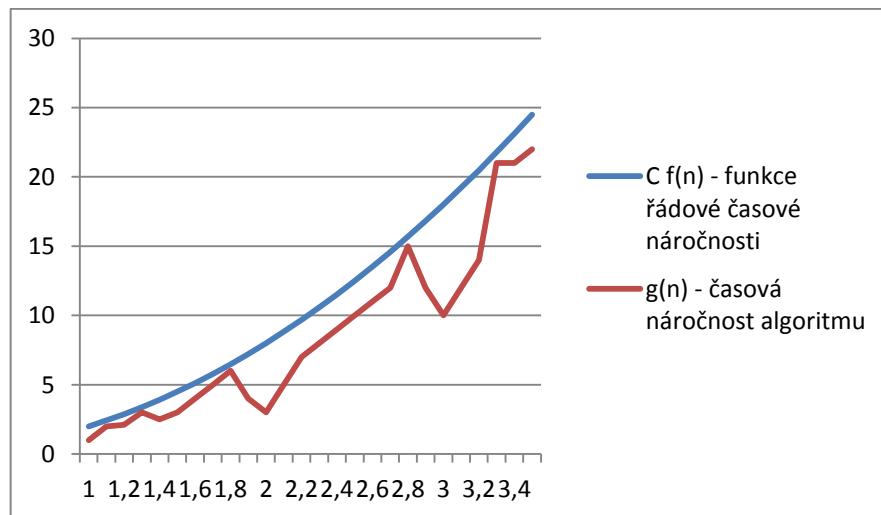
### 1.3.1 Počet provedených elementárních kroků

Pro hodnocení časových složitostí se nepoužívají časové jednotky, ale počet provedených elementárních kroků potřebných pro vykonání výpočtu [7]. Pro účely tohoto textu je pomínuta vágnost termínu elementární krok a problémy které z toho plynou (např. že existuje jen Churchova–Turingova teze a nikoliv Churchova–Turingova věta). Výpočet je prováděn tak, že jsou procházeny zkoumané algoritmy a počítány jejich jednotlivé elementární kroky, a tak je vyjadřována časová složitost přímo v jejich počtech. Výsledkem jsou složitosti například složitosti  $3n^3 + 2n^2 + n$  nebo  $5n^3 + 2n + 150$  atd. Je to nejen pracné ale i nepřehledné a nepraktické. Z hlediska návrhu nebo implementace algoritmu není příliš důležité, zda je uvažována složitost  $5n^3 + 2n + 150$  nebo jen složitost  $n^3$ . Proto nás zajímá pouze řádová složitost. Časová složitost algoritmu, který vykoná  $3n^3 + 2n^2 + n$  elementárních kroků, a časová složitost algoritmu, který vykoná  $5n^3 + 2n + 150$  elementárních kroků, jsou řádově stejné a to  $n^3$ . Jsou vynechávány koeficienty a konstanty a výsledek je zaokrouhlován.

### 1.3.2 Řádové porovnání funkcí

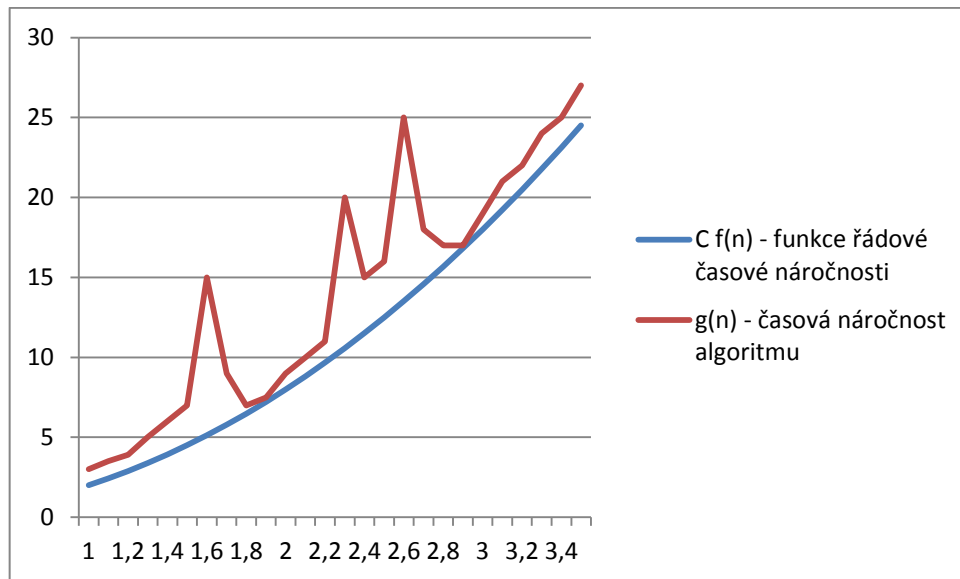
Funkce  $g$  roste řádově nejvýše jako funkce  $f$ , pokud platí:  $g(n) \leq C f(n)$ , kde  $C > 0$  [7] (aby součin  $C f(n)$  nebyl nulový). Hodnota funkce  $g$  nebude pro žádné  $n$  větší hodnoty než součin funkce  $f$  a konstanty  $C$ , kde  $n$  je délka vstupního slova. Délka vstupního slova

můžeme chápana jako délka/velikost dat vstupujících do algoritmu. Pokud si představíme, že funkce  $g$  je časová náročnost nějakého algoritmu představujícího například vyhledávání v databázi a funkce  $f$  je funkce řádové časové náročnosti, která shora ohraničuje funkci  $g$ , pak máme k dispozici efektivní nástroj pro klasifikaci algoritmů.



Obr. 11 Ohraničení funkce  $g(n)$  shora [zdroj vlastní]

V případě zachyceném na obrázku 9 je složitost  $n^2$ . Ohraničení shora se používá spíše u klasických výpočtových algoritmů, kdy nás zajímá chování algoritmu v nejhorším možném případě. Uživatel klikne na tlačítko v nějaké aplikaci a provede se daný algoritmus. Uživatel bude čekat nejdéle  $f(n)$ , pokud datový vstup do algoritmu byl velikosti  $n$ . Pro oblast šifrování je spíše typické ohraničení zdola. Zajímáme se o nejlepší možný případ, tedy nejkratší možnou dobu potřebnou k rozluštění kryptogramu. Je to velmi důležitý údaj jak pro samotný návrh šifrovacích algoritmů, tak pro jejich zavádění do systémů. Obecně platí, že čím více jsou algoritmy komplikované, tím více zpomalují systém jako celek a jejich zavádění je většinou dražší. Pro systémy, ve kterých je relevance informace v řádu sekund, bude požadována po šifrovacím algoritmu menší časová náročnost jeho prolomení. Je akceptovatelné jednodušší řešení (tj. levnější, rychlejší, realizované například samostatným hardwarovým čipem). Jiná situace je u strategických vojenských systémů, kde je relevance dat i několik let. Definice řádového porovnání funkcí upravené pro potřeby šifrování zní: funkce  $g$  roste řádově aspoň jako funkce  $f$ , pokud platí  $g(n) \geq C f(n)$ , kde  $C > 0$  pro délku vstupního slova  $n$  [7]. Popsaná situace je zachycena na obrázku 12.



Obr. 12 Ohraničení funkce  $g(n)$  zdola [zdroj vlastní]

Jestliže skupina algoritmů pro délku vstupních dat  $n$  provede nejvýše/nejméně například  $n^3$  elementárních kroků (instrukcí) můžeme říci, že skupina má časovou složitost  $n^3$ . Tím je zajištěno, že navržená klasifikace je nezávislá na hardwaru.

Tabulka 4 naznačuje dobu potřebnou pro výpočet algoritmů, jejichž časová složitost je dána funkcí  $g(n)$ , velikost dat vstupujících do algoritmu je  $n$ . Algoritmy běží na teoretickém výpočetním prostředku, na kterém každý elementární krok trvá právě 1 ns.

Tabulka 4 má velkou vypovídací schopnost. Vyvozené závěry z tabulky 4 zná každý programátor ze své praxe. Pracovat má smysl jen s algoritmy, které mají časovou složitost nejhůře  $n^3$ . Jedná se o skutečnou hranici reálné použitelnosti. Představme si situaci: algoritmus vyhledávání v databázi má časovou složitost  $n^4$ . Počet záznamů v databázi je 10000, což pro dnešní databáze není mnoho. Jeden dotaz do databáze, který by měl projít všechny záznamy databáze, by trval na teoretickém počítači odvozeného z tabulky 4 - 115 dní, 17 h, 46 min a 40 s. Výpočet tvrzení: pro 10000 záznamů provede algoritmus  $10000^4 = 10000\ 0000\ 0000\ 0000$  instrukcí. Každá instrukce trvá stejně dlouho a to 0,000000001 s. Celkový čas výpočtu je  $10000\ 0000\ 0000\ 0000 \cdot 0,000000001 = 10000000$  s, to je 115 dní, 17 h, 46 min a 40 s. Vypočtený čas není pro praktické použití přijatelný. Uživatel není ochoten tak dlouho čekat, poté co stiskl tlačítko.

Tab. 4 Porovnání délek výpočtů u algoritmů s různou časovou náročností [7]

	n						
$g(n)$	20	40	60	80	100	500	1 000
$n$	20 ns	40 ns	60 ns	80 ns	0,1 $\mu$ s	0,5 $\mu$ s	1 $\mu$ s
$n^2$	0,4 $\mu$ s	1,6 $\mu$ s	3,6 $\mu$ s	6,4 $\mu$ s	10 $\mu$ s	0,25 ms	1 ms
$n^3$	8 $\mu$ s	6,4 $\mu$ s	0,22 ms	0,5 ms	1 ms	0,125 s	1 s
$n^5$	3,2 ms	0,1 s	0,8 s	3,3 s	10 s	9 hodin	12 dní
$2^n$	1 ms	18 min	37 let	14·109 let			
$n!$	77 let						

Pro komplexnější pohled na problematiku je potřeba si uvědomit, že délka trvání vypočtených kroků je pouze teoretická. V praxi se může doba jednotlivých výpočetních velmi lišit. Modelový příklad:

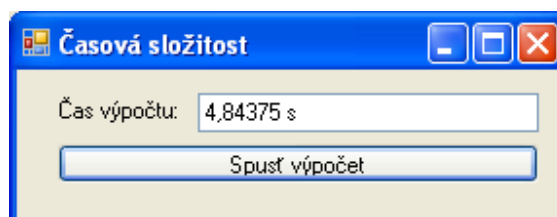
Kód 1:

```
private void button1_Click(object sender, EventArgs e)
{
    // ANALYZA ZACATEK
    DateTime zacatek = DateTime.Now;
    string prom = "mo";

    for (int i = 0; i < 1000000; i++)
    {
        textBox1.Text = prom;
    }

    // ANALYZA KONEC
    DateTime konec = DateTime.Now;
    string dobaVypoctu = ((konec - zacatek).TotalSeconds).ToString() +
    " s";
    textBox1.Text = dobaVypoctu;
}
```

Na začátku kódu se zjistí aktuální čas. Postupně se do `textBox1` milionkrát zapíše hodnota proměnné `prom`. Po ukončení cyklu se opět zjistí aktuální čas. Rozdíl časů je přibližná doba výpočtu. Výsledek je 4,84375 sekund a je zachycen na obrázku.



Obr. 13 Doba výpočtu kódu 1 [zdroj vlastní]

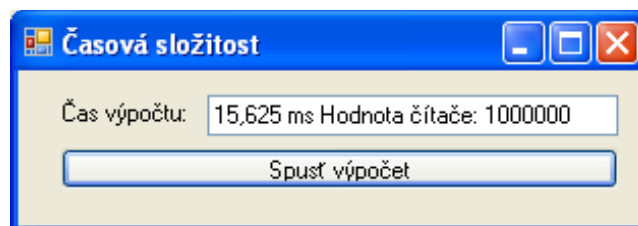
Kód 2:

```
private void button1_Click(object sender, EventArgs e)
{
    // ANALYZA ZACATEK
    DateTime zacatek = DateTime.Now;
    int citac = 0;

    for (int i = 0; i < 1000000; i++)
    {
        citac += 1;
    }

    // ANALYZA KONEC
    DateTime konec = DateTime.Now;
    string dobaVypoctu = ((konec - zacatek).TotalMilliseconds).ToString()
+ " ms" + " Hodnota čítače: " + Convert.ToString(citac);
    textBox1.Text = dobaVypoctu;
}
```

Druhý kód je prakticky identický, místo výpisu hodnoty do `textBox1` se inkrementuje hodnota čítače. Výsledek je 15,625 milisekund a je zachycen na obrázku 14.



Obr. 14 Doba výpočtu kódu 2 [zdroj vlastní]

V kódu 1 trval jeden krok 0,00000484375 sekundy, zatímco v kódu 2 0,000000015625 sekundy. Je vidět, že inkrementace čítače je 310 krát rychlejší. Důvod výsledku je zjevný. Daný příklad demonstruje fakt, že v praxi zdaleka ne všechny výpočtové kroky trvají 1 nanosekundu. Lze předpokládat, že by algoritmus vyhledávání v databázi pracující s časovou složitostí  $n^4$  dosáhl ve skutečnosti ještě horších výsledků. Z výše uvedeného lze odvodit tvrzení, že klasické výpočetní algoritmy by nikdy neměly pracovat s časovou složitostí větší než je  $n^3$ . Moderní šifry by měly být navrženy tak, aby jejich dešifrování bez znalosti hesla pracovalo s časovou složitostí větší než  $n^4$ , ideálně s exponenciální nebo  $n!$  složitostí.

### 1.3.3 Klasifikace časové složitosti z pohledu delšího období

Předchozím textem bylo ověřeno, že navržená klasifikace je nezávislá na konkrétních výpočtových prostředcích. Obstojí uvedená klasifikace v čase? Při hledání odpovědi využijeme tabulku 5.

Tab. 5 Zvětšení rozsahu zpracovatelných dat při 100krát a 1000krát rychlejších počítačích [7]

$g(n)$	zrychlení výpočtu		
	1 x	100 x	1000 x
$n$	100	10 000	100 000
$n^2$	100	1 000	3 162
$n^3$	100	464	1 000
$n^5$	100	251	398
$2^n$	100	106	109
$n!$	100	100	101

Podle Moorova zákona se počet tranzistorů v integrovaných obvodech/procesorech zdvojnásobí každých osmnáct měsíců [8]. Ztotožníme-li dvojnásobení počtu tranzistorů s dvojnásobením výpočetního výkonu, tak za 15 let (180 měsíců) bude 1024 krát vyšší, jak ukazuje tabulka 6. Což přibližně odpovídá poslednímu sloupci tabulky 5. Jestliže dešifrování bez znalosti hesla pracuje se složitostí  $2^n$ , dojde za 15 let ke zrychlení o pouhých 9%. V případě, že by dešifrování pracovalo se složitostí  $n!$ , dojde za 15 let ke zrychlení o jediné procento!

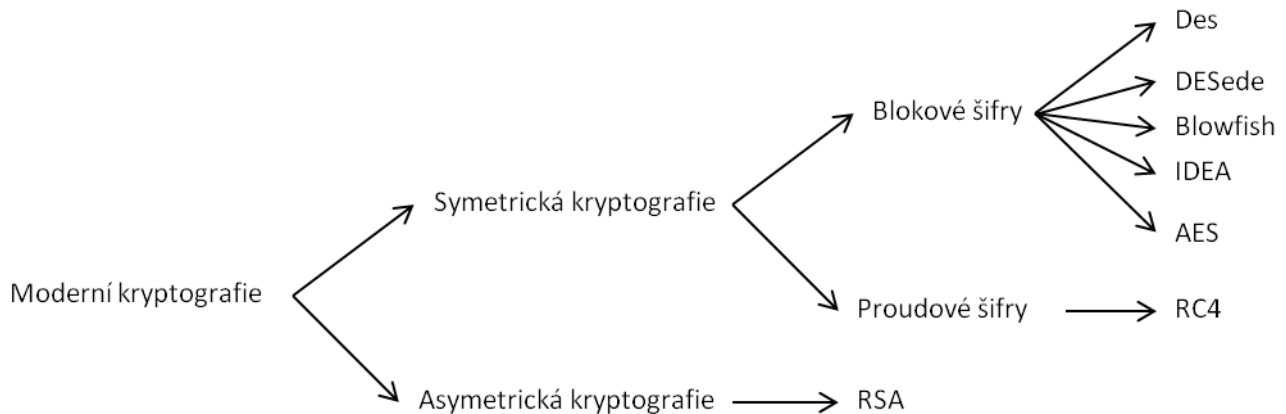
Tab. 6 Nárůst počtu tranzistorů podle Moorova zákona [zdroj vlastní]

Počet měsíců	Znásobení počtu tranzistorů
18	2
36	4
54	8
72	16
90	32
108	64
126	128
144	256
162	512
180	1024

Na základě uvedených faktů lze prohlásit, že daná klasifikace uspěje i v delším časovém horizontu. Následující kapitoly se budou věnovat kryptografickým algoritmům splňujícím tato kritéria.

## 1.4 Moderní kryptografie

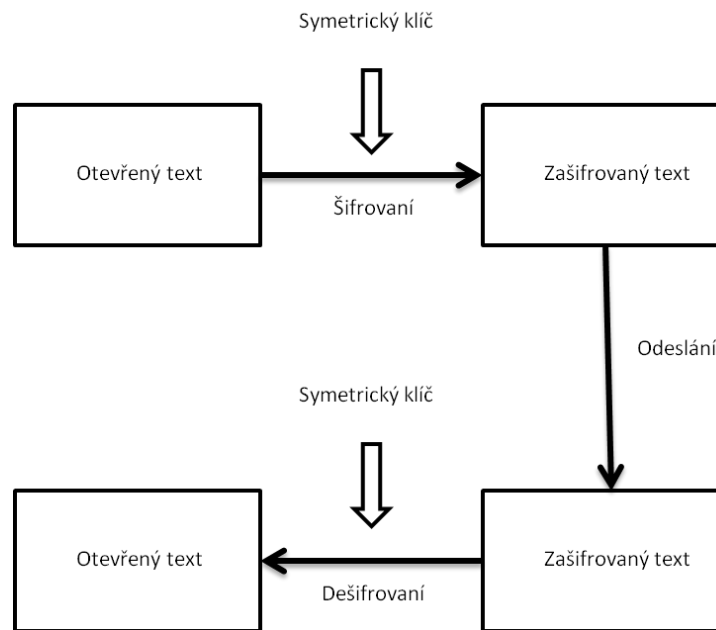
Moderní kryptografii je možné rozdělit na dva podobory podle způsobu provádění šifrování a dešifrování. Schematické znázornění moderní kryptografie je zobrazeno na obrázku 15.



Obr. 15 Moderní kryptografie - typy šifer [zdroj vlastní]

### 1.4.1 Symetrická kryptografie

Symetrická kryptografie provádí šifrování a dešifrováním pomocí jediného klíče označovaného jako symetrický (obrázek 16). Výhodou je rychlé šifrování i dešifrování. Naopak velkým nedostatkem je problematická distribuce symetrických klíčů. V případě zachycení klíče neoprávněnou osobou během přenosu, dojde ke kompromitaci celé šifrované komunikace. Symetrické šifry se dělí podle způsobu práce s bajty souboru, který má být zašifrován.



Obr. 16 Šifrování a dešifrování pomocí symetrického klíče [zdroj vlastní]

### Blokové šifry

Blokové šifry nešifrují všechny bajty souboru naráz ale po menších částech, které se nazývají bloky, a mají pevnou velikost. Šifry mohou mít rozdílnou velikost bloků. Například Advanced Encryption Standard (AES) pracuje s bloky o délce 16 bajtů, Data Encryption Standard (DES) pracuje s bloky o délce 8 bajtů. Málo kdy se stane, že šifrovaný soubor má velikost rovnu n násobku velikosti bloku. Co se zbytkem bajtů, který je menší než velikost bloku? Zbytek je doplněn chybějícími bajty tak, aby měl zbytek bajtů opět velikost bloku. Zarovnání se říká padding (v českém jazyce vycpávka). Problematice paddingu je věnována podkapitola 2.2.3.

### Feistelova šifra

Feistelova šifra slouží jako základ mnoha blokových šifer. Mezi nejznámější patří DES a Lucifer. DES v současnosti není považována za bezpečnou, protože používá krátké 56 bitové klíče. DES byla nahrazena šifrou TripleDES, která používá 168 bitové klíče ( $3 \cdot 56$  bitů).

Popis algoritmu Feistelovi šifry:

Nechť  $x$  je otevřená zpráva, která lze rozdělit na dva bloky stejné délky  $x = (L_0, R_0)$ .

Bloky  $L_0, R_0$  mají stejnou délku:  $|L_0| = |R_0| = d$ , takže délka zprávy  $x$  je vždy sudá.

Počet kol je  $n$ .

Princip šifrování:

Krok 0:

$$i = 1$$

Krok 1 až  $n$  (jednotlivá kola)

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus f(R_i, K_i)$$

Funkce  $f(R_i, K_i)$  je permutace množiny klíče  $K_i$ , tedy

$$f \begin{pmatrix} R_i \\ K_i \end{pmatrix} = \begin{pmatrix} R_{i_0} & R_{i_1} & \dots & \dots & R_{i_{d-1}} \\ K_{i_0} & K_{i_1} & \dots & \dots & K_{i_{d-1}} \end{pmatrix}$$

Princip dešifrování:

Krok 0:

$$j = n$$

Krok  $n$  až 1 (jednotlivá kola)

$$R_{j-1} = L_j$$

$$L_{j-1} = R_j \oplus f(L_j, K_{j-1})$$

Funkce  $f(L_j, K_{j-1})$  je permutace množiny klíče  $K_{j-1}$ , tedy

$$f \begin{pmatrix} L_j \\ K_{j-1} \end{pmatrix} = \begin{pmatrix} L_{j_0} & L_j & \dots & \dots & L_{j_{d-1}} \\ K_{j-1_0} & K_{j-1_1} & \dots & \dots & K_{j-1_{d-1}} \end{pmatrix}$$

Následující příklad projde krok za krokem algoritmus Feistelovi šifry a prezentuje výpočet nejen numericky ale i graficky.

Příklad:

$$x = 00101000, K_0 = 2130, K_1 = 3021$$

Šifrování:

$$L_0 = 0010, R_0 = 1000$$

Kolo  $i = 1$

$$L_1 = R_0 = 1000$$

$$R_1 = L_0 \oplus f(R_0, K_0) \quad f(R_0, K_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 \end{pmatrix} = 0010$$

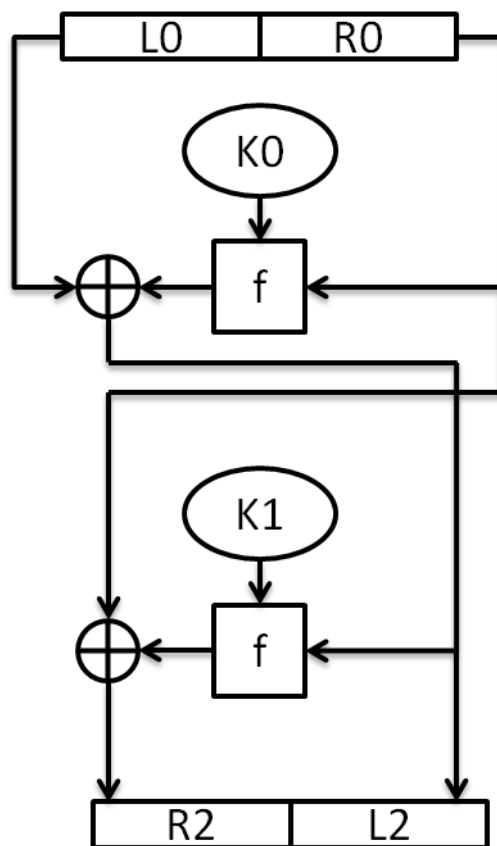
$$R_1 = 0010 \oplus 0010 = 0000$$

Kolo  $j = 2$

$$L_2 = R_1 = 0000$$

$$R_2 = L_1 \oplus f(R_1, K_1) \quad f(R_1, K_1) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 1 \end{pmatrix} = 0000$$

$$R_2 = 1000 \oplus 0000 = 1000$$



Obr. 17 Šifrování pomocí Feistelovi šifry

[zdroj vlastní]

Dešifrování:

Kolo  $j = 2$

$$R_1 = L_2 = 0000$$

$$L_1 = R_2 \oplus f(L_2, K_1) \quad f(L_2, K_1) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 1 \end{pmatrix} = 0000$$

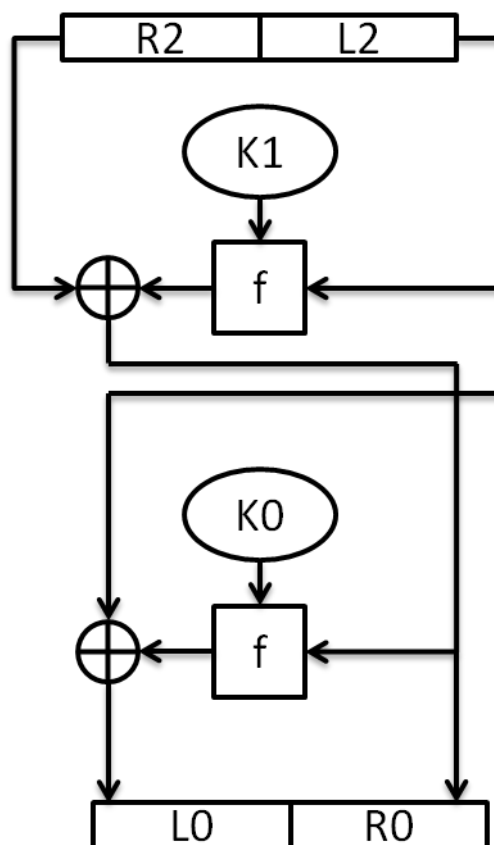
$$L_1 = 1000 \oplus 0000 = 1000$$

Kolo  $j = 1$

$$R_0 = L_1 = 1000$$

$$L_0 = R_1 \oplus f(L_1, K_0) \quad f(L_1, K_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 \end{pmatrix} = 0010$$

$$L_0 = 0000 \oplus 0010 = 0010$$



Obr. 18 Dešifrování pomocí Feistelovi šifry [zdroj vlastní]

### Proudové šifry

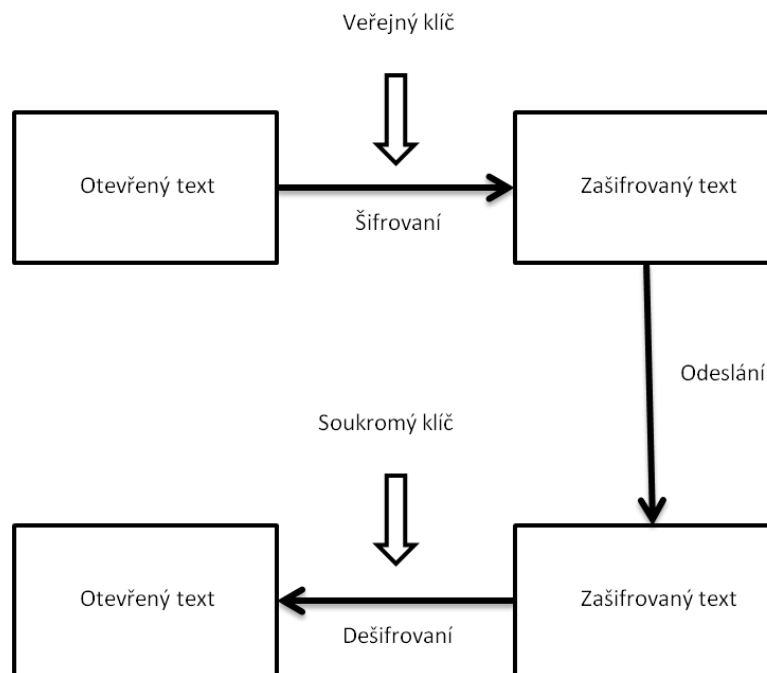
Proudové šifry provádí šifrování po jednotlivých bajtech. Šifrování je rychlejší než blokové, protože se nemusí čekat, až bude zašifrován celý blok. Často se používá pro šifrované spojení v reálném čase, například pro šifrování bezdrátového spojení mezi

koncovým počítačem a přístupovým bodem. Proudové šifry nejsou vhodné pro šifrování souborů v operačním systému Android, protože mnoho zařízení běžících pod operačním systémem Android má stále ještě omezenou operační paměť. Při šifrování nedochází k odesílání proudu ale k jeho ukládání do operační paměti. Jakmile je překročena velikost přidělené operační paměti, dojde k pádu šifrovací aplikace.

#### 1.4.2 Asymetrická kryptografie

Asymetrická kryptografie provádí šifrování a dešifrování pomocí různých klíčů. Šifrovacímu klíči se říká veřejný klíč. Na rozdíl od symetrického klíče může být veřejný klíč volně vystaven například na webové stránce. Kdokoliv chce majiteli klíče poslat šifrovanou zprávu, může ho použít (šifrovat může každý). Pouze majitel soukromého klíče může zprávu rozšifrovat. Jediným držitelem soukromého klíče je jeho majitel (obrázek 19). Veřejný a soukromý klíč spolu matematicky souvisí, ale odvození soukromého klíče z veřejného klíče je prakticky nemožné (pro dostatečně dlouhé klíče). Asymetrická kryptografie odstraňuje největší slabinu symetrické kryptografie, kterou je sdílení klíče. Výhody:

- šifrovaná komunikace může probíhat i mezi lidmi, kteří si nedůvěřují natolik, aby vlastnily společný klíč.
- odpadá nutnost utajené distribuce klíče a s tím spojené riziko jeho zachycení.



Obr. 19 Šifrování a dešifrování pomocí veřejného a soukromého klíče [zdroj vlastní]

## RSA

Nejnámějším šifrovacím algoritmem asymetrické kryptografie je algoritmus RSA. Byl vytvořen pány Rivest, Shamir, Adleman v Massachusetts Institute of Technology. Z jejich počátečních písmen je odvozen název algoritmu. Publikován byl v roce 1978 [9]. Algoritmus RSA je založen na matematickém problému, který byl v té době již dlouho znám: problém rozkladu na prvočinitele neboli faktorizace. Jde o geniální využití problému ve prospěch kryptografie. Vynásobit prvočíslo  $p$ , které má  $u$  dekadických cifer, a prvočíslo  $q$ , které má  $v$  dekadických cifer, lze pomocí  $\log_2 u \cdot \log_2 v$  bitových operací [10]. Operace  $n = p \cdot q$  má vynikající časovou složitost. Ovšem opačný postup, tj. nalezení prvočinitelů  $p$  a  $q$  není pro dostatečně velké  $n$  v polynomiálním čase.

### Příklad RSA

Vytvoření soukromého a veřejného klíče.

Náhodně vyber dvě prvočísla:

$$p = 19$$

$$q = 23$$

Proveď součin čísel  $p$  a  $q$ .

$$n = p \cdot q = 19 \cdot 23 = 437$$

Vypočti Eulerovu funkci čísla  $n$

$$\varphi(n) = \varphi(pq) = (p - 1)(q - 1) = (19 - 1)(23 - 1) = 396$$

Náhodně vyber číslo  $c$  takové, že největší společný dělitel čísla  $c$  a  $\varphi(n)$  je 1.

Ověř číslo  $c$ , že splňuje výše uvedenou podmínku. Ověření lze provést například Euklidovým algoritmem:

$$\frac{396}{31} = 12 \text{ zbytek } 24$$

$$\frac{31}{24} = 1 \text{ zbytek } 7$$

$$\frac{24}{7} = 3 \text{ zbytek } 3$$

$$\frac{7}{3} = 2 \text{ zbytek } 1$$

$$\frac{3}{1} = 3 \text{ zbytek } 0$$

Největší společný dělitel čísla  $c$  a  $\varphi(n)$  je 1, lze pokračovat.

Proveď inverzi čísla  $c$  v  $\mathbb{Z}_{\varphi(n)}$ :

		Zbytek	Koeficienty	
		396	1	1
		31	0	0
	Násobek	31	0	0
396 / 31	12	24	$1 - 12 \cdot 0 = 1$	$0 - 12 \cdot 1 = -12$
31 / 24	1	7	$0 - 1 \cdot 1 = -1$	$1 - 1 \cdot (-12) = 13$
24 / 7	3	3	$1 - 3 \cdot (-1) = 4$	$-12 - 3 \cdot 13 = -51$
7 / 3	2	<b>1</b>	$-1 - 2 \cdot 4 = -9$	$13 - 2 \cdot (-51) = \mathbf{115}$

↑
↑  
 Inverze existuje Výsledek

Obr. 20 Výpočet inverze čísla  $C$  [zdroj vlastní]

Poznámka: Výpočet zobrazený na obrázku 20 byl proveden Rozšířeným Euklidovým algoritmem.

Veřejný klíč: (437, 31). Soukromý klíč: (437, 115).

Zpráva určená k zašifrování  $x = 25$ ;

**Šifrování:**

$$y = 25^{31} \bmod 437 = 2,1684043449710088680149056017399e + 43 \bmod 437 \\ = 213$$

Dešifrování:

$$x = 213^{115} \bmod 437 = 5,8030243840553968945662293365048e + 267 \bmod 437 \\ = 25$$

Bezpečnost RSA

Útok na RSA 1: Prolomení soukromého klíče hrubou silou (nazývaný jako faktorizace modula)

Mezi laiky ale i některými IT odborníky panuje mýtus, že šifrování RSA je tak dokonalé, že i krátké klíče jsou bezpečné. Je to pravda? Jak silný je soukromý klíč (437, 115)? Následuje pokus o jeho prolomení.

Z veřejného klíče (437, 31) útočník získá  $n$  a provede na něj útok hrubou silou. Při útoku vždy vezme  $n$  a jako modulátor se postupně budou zkoušet prvočísla s cílem najít takové, že platí:  $437 \bmod v = 0$ . Pak byl nalezen jeden činitel součinu  $n = p \cdot q$ . Druhý činitel lze triviálně dopočítat.

$$437 \bmod 3 = 2$$

$$437 \bmod 5 = 2$$

$$437 \bmod 7 = 3$$

$$437 \bmod 11 = 8$$

$$437 \bmod 13 = 8$$

$$437 \bmod 17 = 12$$

$437 \bmod 19 = 0$  Bylo nalezeno  $p$ , zbývá dopočítat  $q$ .

$$n = p \cdot q, q = \frac{n}{p} = \frac{437}{19} = 23$$

Výpočtem zobrazeným na obrázku 20 se dopočítá 115. Soukromý klíč je prolomen.

Útok na soukromé RSA klíče je účinný pouze tehdy, pokud jsou zvoleny dostatečně krátké klíče. Pro dlouhé klíče nejsou výsledky útoku hrubou silou očekávatelné v polynomiálním čase. Z toho důvodu jsou dlouhé klíče velmi bezpečné. Příklad ilustruje, že pokud budou uživatelé a programátoři používat krátké klíče, nebudou systémy a data dostatečně zabezpečeny, přestože algoritmus RSA je bezpečný. Příklad potvrzuje známou skutečnost, že nejslabším článkem zabezpečení je člověk.

Útok na RSA 2: Zachycení  $\varphi(n)$

Představme si další situaci: útočník zachytil  $\varphi(n)$  a  $n$  si zjistil veřejného klíče.

$$p = 17077$$

$$q = 10177$$

$$n = p \cdot q = 173792629$$

$$\varphi(n) = (p - 1)(q - 1) = (17077 - 1)(10177 - 1) = 173765376$$

Víme, že  $n = p \cdot q$ , vyjádříme  $p = \frac{n}{q}$  a dosadíme do  $\varphi(n)$ :

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

$$\varphi(n) = pq - p - q + 1$$

$$\varphi(n) = \frac{nq}{q} - p - q + 1$$

$$\varphi(n) = n - p - q + 1$$

$$\varphi(n) = n + 1 - (p + q)$$

Nyní dosadíme za  $\varphi(n)$  a  $n$ :

$$173765376 = 173792630 - (p + q)$$

$$p + q = 173792630 - 173765376$$

$$p + q = 27254$$

Podle věty 3.5.1 o bezpečnosti RSA [11] vypočteme:

$x^2 - 2b + n$ ,  $2b = p + q$  tedy hledáme kořeny kvadratické rovnice:

$$x^2 - 27254 + 173792629$$

Výpočet kořene p

$$p = \frac{2b + \sqrt{4b^2 - 4n}}{2}$$

$$p = \frac{27254 + \sqrt{742780516 - 4 \cdot 173792629}}{2}$$

$$p = \frac{27254 + \sqrt{47610000}}{2}$$

$$p = \frac{27254 + 6900}{2}$$

$$p = 17077$$

Výpočet kořene q

$$q = \frac{2b - \sqrt{4b^2 - 4n}}{2}$$

$$q = \frac{27254 - 6900}{2}$$

$$q = 10177$$

Příklad druhého útoku na RSA ukazuje na potřebu opatrné implementace RSA. V případě, že aplikace používající RSA ukládala do operační paměti  $\varphi(n)$ , a útočník by její hodnotu zjistil, klíč by byl opět prolomen. Byť sama metoda RSA je vynikající.

## 2 CHARAKTERISTIKA SOUČASNÝCH TECHNOLOGIÍ KRYPTOGRAFIE PRO MOBILNÍ PLATFORMU ANDROID

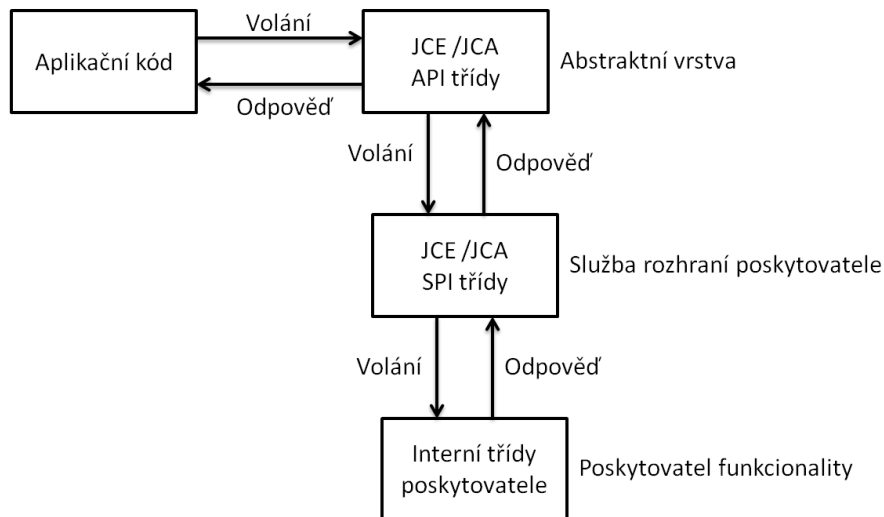
Vývoj aplikací pro mobilní zařízení s operačním systémem Android se provádí prostřednictvím jazyka Java, rozšíření jazyka Java o rysy specifické pro operační systém Android a XML návrh grafického uživatelského rozhraní [12]. Existují i jiné možnosti vývoje aplikací pro operační systém Android. Například se jedná o rozšíření Visual Studia nebo Xamarin Studio, které umožňují programování v jazyce C#. Uvedené programy mají v současné době menšinové postavení bez oficiální podoby společnosti Google, která je výrobcem operačního systému Android. Z tohoto důvodu se práce zabývá pouze plně profesionálním způsobem programování, který je uveden výše. Kryptografie na mobilní platformě Android má svá specifika, jimž se bude věnovat kapitola 3.

### 2.1 Java Cryptography Architecture (JCA) a Java Cryptography Extension (JCE)

Pokud se při vývoji aplikací pro operační systém Android vyskytne potřeba vytvořit specializovaný šifrovací nástroj nebo jen potřeba dílčího zabezpečení aplikace či komunikace aplikace, vždy se využije služeb JCA a JCE. JCA je kryptografická architektura jazyka Java. Zabývá se certifikáty a digitálními podpisy. JCE je kryptografické rozšíření. Dávají jazyku Java možnosti šifrování a dešifrování.

JCA a JCE poskytují pro vývojáře aplikací abstraktní vrstvu. Vývojář se nemusí detailně zabývat tím, jak kryptografický algoritmus pracuje nebo dokonce vytvářet vlastní implementace šifrovacích algoritmů. O vše se stará sada tříd, metod a rozhraní, které poskytují JCA a JCE. Metody JCA a JCE obsahují pečlivě provedené implementace všech potřebných šifrovacích algoritmů, to umožňuje rychlé a kvalitní psaní kryptografických částí programu. Každý programátor je může použít. Ve světě Java je používáno označení „provider-based architecture“, v českém překladu architektura založená na poskytovateli. Architektura má i své nevýhody. Například z důvodu výpočetního omezení mobilního telefonu je potřeba měnit chování tříd poskytovaných JCA nebo JCE. Řešení problémů jsou kostrbatá, ne-li nemožná.

Na obrázku 21 je vidět spolupráce jednotlivých částí JCE/JCA. Kód aplikace zavolá příslušné JCE/JCA API třídy. Ty zavolají SPI třídy rozhraní poskytovatele. Třídy SPI zavolají interní kód poskytovatele, který dodá požadovanou funkcionalitu.



Obr. 21 Architektura JCE/JCA [autorem přeložen zdroj 13]

## 2.2 Symetrická kryptografie

Symetrická kryptografie je jedním z hlavních pilířů sloužící k zabezpečení dat v Javě. Symetrická kryptografie je vhodná zejména k šifrování velkých objemů dat. Je možné používat jak šifrování založené na klíčích, tak šifrování založené na heslech. Podkapitola se zabývá:

- Symetrickou kryptografií založenou na klíčích.
- Paddingem neboli dorovnáním bajtů.
- Režimy blokových šifer.
- Symetrickou kryptografií založenou na heslech.

Symetrická kryptografie v jazyce Java je vysvětlena na ukázce zdrojového kódu. Prezentovaná ukázka volně vychází z kódu uvedeného v literatuře [13]. Podobné ukázky lze nalézt i ve webové vědomostní bázi Stack Overflow [14], která je určena profesionálním programátorům. Jedná se o nejjednodušší ukázku symetrického šifrování pomocí klíče. Není mnoho způsobů jak jinak naprogramovat šifrovací kód, aby byl stále korektní a jednoduchý.

Ukázka zdrojového kódu:

```

public void zasifruj (View view)
{
    byte[] vstup = new byte[] { (byte)1, (byte)2, (byte)3, (byte)4,
        (byte)5, (byte)6, (byte)7};
  
```

```
int delkaPBPK = 16;

byte[] poleBajtuProKlic = new byte[delkaPBPK];

for(int i = 0; i < delkaPBPK; i++)
{
    poleBajtuProKlic[i] = (byte)i;
}

try
{
    SecretKeySpec symetrickyKlic =
    new SecretKeySpec(poleBajtuProKlic, "AES");

    Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding", "BC");

    byte[] zasifrovanePoleBajtu = new byte[vstup.length];

    cipher.init(Cipher.ENCRYPT_MODE, symetrickyKlic);

    int delkaZPB =
    cipher.update(vstup, 0, vstup.length, zasifrovanePoleBajtu,
    0);

    delkaZPB += cipher.doFinal(zasifrovanePoleBajtu, delkaZPB);

    tv1.setText(this.byteNaString(zasifrovanePoleBajtu));
}
catch (Exception e)
{
    tv1.setText("Při šifrování se vyskytka chyba: " +
    e.toString());
}
}
```

Jednotlivé třídy a metody tříd jsou vysvětleny v následujícím textu.

### 2.2.1 Třída SecretKeySpec

Nejjednodušší způsob v jazyce Javě, jak vytvořit symetrický klíč, je použít třídu SecretKeySpec. Před samotným použitím je potřeba provést import javax.crypto.spec.SecretKeySpec; do zdrojového souboru. Jsou dva konstruktory třídy SecretKeySpec:

Verze 1: SecretKeySpec(byte[] key, int offset, int len, String algorithm) [15]

key – Klíč se vytváří z dodaného pole bajtů. Jeho obsah je nakopírován do vytvářené instance třídy SecretKeySpec a je chráněn proti následným úpravám.

len – Při vytváření klíče se nemusí použít celé pole bajtů, ale jen prvních  $n$  bajtů pole, kde  $n = len$ .

algorithm – Název šifrovacího algoritmu, pomocí kterého bude vytvářený klíč šifrovat/dešifrovat. Těmto názvům se říká AlgorithmParameters a jsou specifikovány ve zdroji [16].

Verze 2: SecretKeySpec(byte[] key, String algorithm) [17].

Parametry key a algorithm jsou stejné. Rozdíl je, že konstruktor použije celé pole bajtů key. Literatura [13] varuje před vytvářením slabých (krátkých klíčů). Přesto není možné vytvořit klíč, který by byl delší než 256 bitů. Pokud bychom chtěli vytvořit klíč například o délce 1024 bitů. V ukázce na straně 42 by se změnila proměnná delkaPBPK na 128 (128B = 1024b). Skončí běh programu výjimkou, jak ukazuje obrázek 22.



Obr. 22 Výjimka vyvolaná dlouhým klíčem [zdroj vlastní]

### 2.2.2 Třída Cipher

Třída Cipher se stará o šifrování i dešifrování. Její instance se nevytváří klasicky konstruktorem, ale pomocí metody getInstance (). Vytváření objektu Cipher je potřeba dát do bloku try-catch, jinak skončí překlad programu chybami: Unhandled exception type NoSuchPaddingException a Unhandled exception type NoSuchAlgorithmException.

### Metoda `Cipher.getInstance ()`

Instance JCA a JCE objektů jsou vytvářeny pomocí `getInstance ()`. Programátor může zadat poskytovatele, který se bude používat. V tomto případě je jedná o instance třídy `Cipher`. Metoda se použije ve tvaru: `getInstance(String transformation, String provider)` [18]. Pokud chce aplikační programátor například používat poskytovatele „Bouncy Castle“, do zdrojového kódu zapíše:

```
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding" , "BC");
```

Pokud programátora nezajímá, který poskytovatel bude použit pro vytváření objektů, stačí ho neuvést:

```
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding" );
```

V takovém případě provede výběr poskytovatele samo běhové prostředí jazyka Java (Java Runtime). Vybere prvního poskytovatele schopného splnit požadavek podle seznamu poskytovatelů v souboru `java.security`. Přičemž platí, že poskytovatelé s nižším preferenčním číslem mají přednost před těmi s vyšším.

Parametr transformace je trochu zvláštní. Nahrazuje obvyklý parametr `algorithm`. Je to vlastně složenina tří položek oddělených lomítkem: `algorithm /mode/padding`. Položka `algorithm` je povinná, zbývající dvě jsou nepovinné [19]. Nicméně kniha [13] doporučuje všechny tři položky uvádět. Z důvodu, že by programátor po nějaké době mohl změnit poskytovatele, nový poskytovatel může mít jiný výchozí mód či padding, než s jakými se v programátorově kódu počítá. To by mohlo vést ke zbytečným chybám.

Upozornění: metoda `getInstance` třídy `Cipher` se používá jako konstruktor, musí být použita jako první a všechny ostatní metody dané třídy se používají až po `getInstance`. V opačném případě skončí běh aplikace výjimkou `IllegalStateException` [20].

### Metoda `Cipher.init()`

Existuje osm verzí metody `init` [21]. Je popsána nejpoužívanější:

```
init(int opmode, Key key) [22]
```

`opmode` – představuje operační režim, ve kterém bude instance třídy `Cipher` pracovat: `ENCRYPT_MODE` (pro šifrování), `DECRYPT_MODE`(pro dešifrování), `WRAP_MODE`, `UNWRAP_MODE` [23].

### Metoda `Cipher.update()`

Jakmile je objekt `Cipher` pomocí metod `getInstance` a `init` nastaven, lze mu předkládat data. Objekt s daty provede činnost nastavenou v parametru `opade` metody `init`. Samotná práce (šifrování/dešifrování) se provádí pomocí několika metod, z nichž nejpoužívanější je `update`. Činnost metody probíhá následovně:

```
int update(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset) [24]
```

`input` – vstupní pole bajtů

`inputOffset` – posunutí začátku vstupu. Vstupní pole bajtů se nebude číst od začátku ale od hodnoty specifikované zde.

`inputLen` – délka vstupu

`output` - výstupní pole bajtů

`outputOffset` – specifikuje, kde se má pokračovat ve výstupu. Provádí se pomocí návratové hodnoty, viz níže.

Metoda `update` pracuje tak, že si objekt ze vstupního pole bajtů (v příkladu výše je to `vstup`) vezme blok dat. Data zpracuje a výsledek uloží do výstupního pole bajtů. V příkladu výše je to `zasifrovanePoleBajtu`. Následně vezme další blok dat a provede to samé, pak další atd. Není jisté, kolik dat se při aktualizaci skutečně zapíše do výstupního pole. Může to být od nuly až po počet bajtů specifikovaném v `inputLen`. Kolik bajtů se skutečně zapsalo, je potřeba sledovat pomocí návratové `int` hodnoty metody `update`. Údaj se potom uvádí v `outputOffset`, kromě prvního `update`, protože se na výstup zapisuje poprvé (na začátek) a je tam hodnota 0.

### Metoda `Cipher.doFinal()`

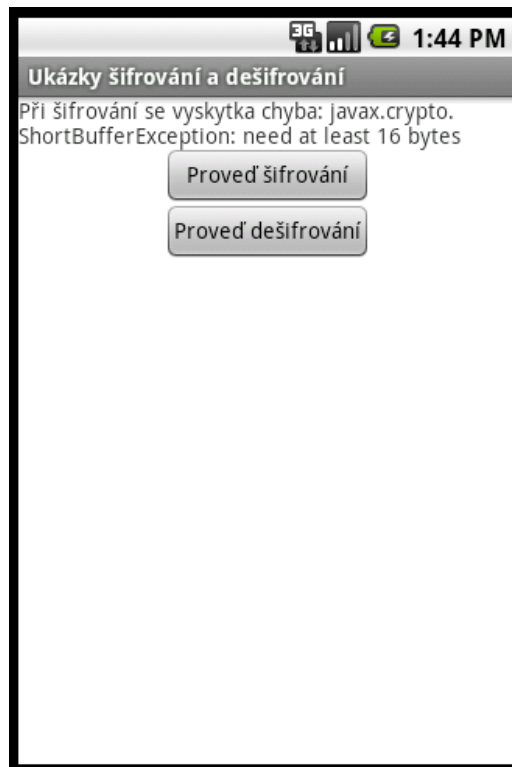
Stejně jako `update` je metoda `doFinal` pracovní metodou třídy `Cipher`. Stejně jako u `update` není, jisté kolik bajtů bude zapsáno na výstup. Je nutné sledovat počet zapsaných bajtů pomocí návratové hodnoty metody. Jinak se nelze vyhnout `NullPointerException` výjimkám.

```
int doFinal(byte[] output, int outputOffset) [25]
```

`output` - výstupní pole bajtů

`outputOffset` – specifikuje, kde se má pokračovat ve výstupu. Je to stejné jako u metody `update`.

Nyní je program kompletní. Pokud by byl spuštěn, skončil by přesto výjimkou zachycenou na obrázku 23. Důvodem je, že šifra AES má pevně danou velikost bloku dat, se kterým může pracovat a to je 16 bajtů. Uvedeno v kapitole 1 blokové šifry. Pro šifrování dat velikosti menší než 16 bajtů musí být použita vycpávka dorovnávací chybějící bajty, aby data měla 16 bajtů.

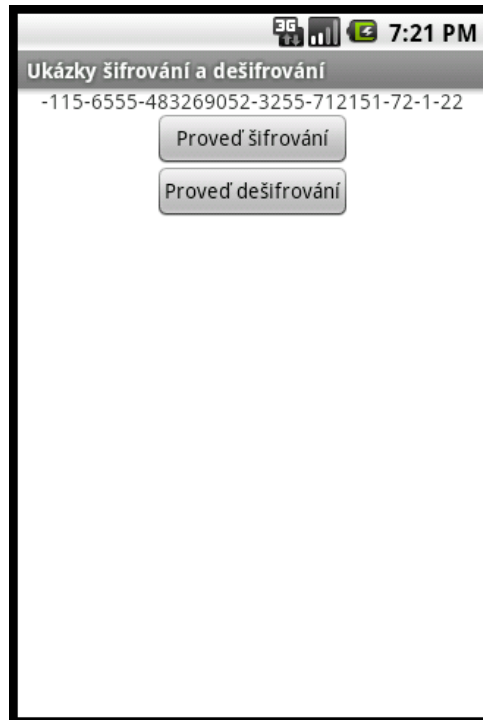


Obr. 23 Výjimka *ShortBufferException*  
[zdroj vlastní]

Doplnění chybějících bajtů, například:

```
byte[] vstup = new byte[] { (byte) 1, (byte) 2, (byte) 3, (byte) 4,  
                             (byte) 5, (byte) 6, (byte) 7, (byte) 9,  
                             (byte) 9, (byte) 9, (byte) 9, (byte) 9,  
                             (byte) 9, (byte) 9, (byte) 9, (byte) 9};
```

Ukázka proběhne bez problému, jak je zachyceno na obrázku 24.



Obr. 24 Výsledek šifrování [zdroj vlastní]

### 2.2.3 Padding neboli dorovnání bajtů

Padding neboli dorovnávání bajtů se používá, jestliže je šifrovaný soubor/pole bajtů menší než velikost bloku, nebo není velikost souboru  $n$  násobkem velikosti bloku dané šifry. Chybějící bajty se musí dorovnat na velikost bloku. Potom je velikost dat připravených k šifrování  $n$  násobkem velikosti bloku dané šifry.

Data před paddingem (55 B)



Data po paddingu (64 B)



Obr. 25 Dorovnávání bajtů [zdroj vlastní]

### Ruční padding

Potřebuje-li programátor řídit padding ručně, je potřeba o tom informovat instanci třídy Cipher prostřednictvím položky NoPadding.

```
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding", "BC");
```

#### Ruční padding při šifrování

Nejprve je potřeba zjistit kolik bajtů šifrovaného souboru chybí. Například  $p \bmod b = z$ , kde  $p$  je počet bajtů šifrovaného souboru a  $b$  je velikost šifrovacího bloku. Velikost bloku lze zjistit pomocí Cipher.getBlockSize (). V příkladě na obrázku 25 to bude  $55 \bmod 16 = 7$ , nyní se vypočítá, kolik bajtů bude potřeba dorovnat:  $b - z = 16 - 7 = 9$ . Do zbývajících devíti bajtů se umístí hodnota 9. Jak je vidět na obrázku 26.

#### Poslední blok dat k zašifrování (16 B)

DATA 1B	DATA 1B	DATA 1B	DATA 1B
DATA 1B	DATA 1B	DATA 1B	HODNOTA 9
HODNOTA 9	HODNOTA 9	HODNOTA 9	HODNOTA 9
HODNOTA 9	HODNOTA 9	HODNOTA 9	HODNOTA 9

Obr. 26 Doplnění chybějících bajtů [zdroj vlastní]

#### Ruční padding při dešifrování

Nejprve se přečte hodnota posledního bajtu zašifrovaného souboru, hodnota 9. Následně se ze souboru odstraní posledních devět bajtů. Odstraní se paddingové bajty.

#### Speciální případ $p \bmod b = 0$

Jak probíhá ruční padding, pokud je počet bajtů šifrovaného souboru  $n$  násobkem velikosti bloku dané šifry? Na konec souboru je přidán takový počet bajtů, který odpovídá velikosti šifrovacího bloku. Situace je znázorněna na obrázku 27.

### Data před paddingem (48 B)



### Data po paddingu (64 B)



*Obr. 27 Padding souboru, který má velikost rovnou  $n$  násobku velikosti bloku šifry [zdroj vlastní]*

Při dešifrování je poslední celý blok odebrán.

#### Automatický padding

Padding PKCS # 5 byl původně vyvinut pro starší blokové šifry, jako je například DES. Počítal pouze s velikostí bloku 8 bajtů. Později byl vytvořen PKCS #7, který umožňuje dorovnání až na velikost bloku 255 bajtů. Pokud se programátor nechce o padding starat, stačí si vybrat jeden z automatických paddingů. A oznámí to instanci třídy Cipher pomocí:

```
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
```

Příklad používání paddingu na kódu vychází z literatury [13].

```
public void zasifruj (View view)
{
    byte[] vstup = new byte[] { (byte)1, (byte)2, (byte)3, (byte)4,
                                (byte)5, (byte)6, (byte)7 };

    int delkaPBPK = 32;

    byte[] poleBajtuProKlic = new byte[delkaPBPK];

    for(int i = 0; i < delkaPBPK; i++)
    {
        poleBajtuProKlic[i] = (byte)i;
    }

    try
    {
        SecretKeySpec symetrickyKlic =
            new SecretKeySpec (poleBajtuProKlic,
                               "AES");

        Cipher cipher =
            Cipher.getInstance("AES/ECB/PKCS7Padding",
                               "BC");
```

```
cipher.init(Cipher.ENCRYPT_MODE, symetrickyKlic);

byte[] zasifrovanePoleBajtu =
    new byte[cipher.getOutputSize(vstup.length)];

int delkaZPB =
cipher.update(vstup, 0, vstup.length, zasifrovanePoleBajtu, 0);

delkaZPB += cipher.doFinal(zasifrovanePoleBajtu, delkaZPB);

tv1.setText("Kryptogram:\n" +
this.byteNaString(zasifrovanePoleBajtu) + "\n\n" +
"Velikost vstupního pole: " +
String.valueOf(vstup.length) + "\n" +
"Velikost zašifrovaného pole: " +
String.valueOf(zasifrovanePoleBajtu.length) +
"\n\n");
}
catch (Exception e)
{
    tv1.setText("Při šifrování se vyskytka chyba: " +
e.toString());
}
}
```

Je důležité, aby výstupní pole (`zasifrovanePoleBajtu`) bylo vytvořeno až po zavolání metody `Cipher.init ()`. Délka výstupního pole je zjištěna zavoláním metody `Cipher.getOutputSize ()`. Metoda jako parametr používá délku vstupního pole (`vstup.length`). `Cipher.getOutputSize ()` vrátí délku, jakou bude mít výstupní (šifrované / dešifrované) pole. Může být stejná jako délka vrácená metodami `Cipher.getOutputSize ()` a `Cipher.doFinal ()`, ale může být větší, což se často stává!

Důvodem rozdílu je, že objekt třídy `Cipher` nemá žádnou možnost, jak by mohl zjistit kolik bajtů paddingu pole obsahuje, dokud neproběhne jeho rozšifrování. Jde o rozdíl mezi odhadem metody `Cipher.getOutputSize ()` a skutečnou velikostí vrácenou metodami `Cipher.getOutputSize ()` a `Cipher.doFinal ()`.



*Obr. 28 Automatický padding [zdroj vlastní]*

Na obrázku 28 je vidět, že velikost vstupního pole je 7, ale velikost výstupního pole je 16. Padding proběhl zcela automaticky, aniž by byla velikost vstupního pole ručně zarovnáována. Což je pohodlné.

#### **2.2.4 Režimy blokových šifer**

Pro účely blokových šifer byla vytvořena celá řada režimů. První byl vytvořen ECB (Electronic Code Book). Ostatní režimy například CBC, CTS, CTR a další staví na ECB a snaží se odstranit nedostatky ECB.

##### **ECB (Electronic Code Book)**

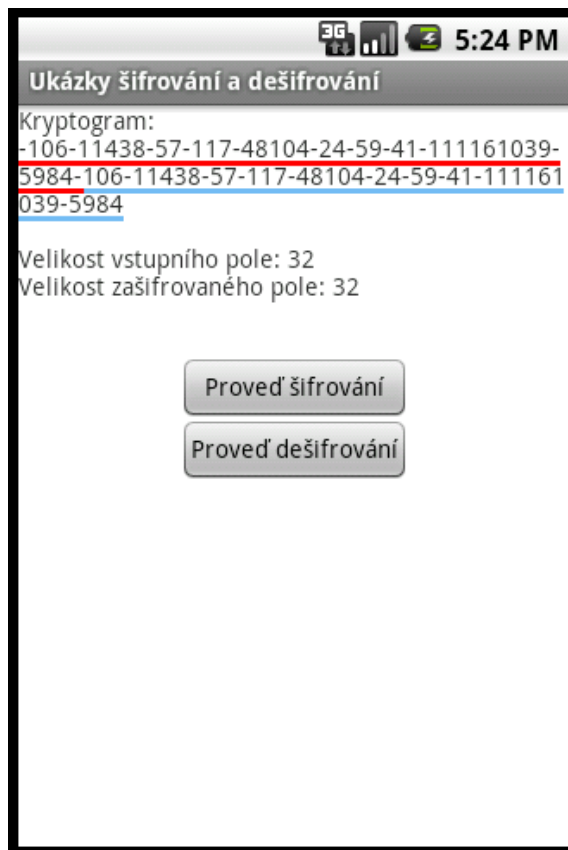
ECB pouze rozděljuje vstupní pole na bloky. Každý blok samostatně zašifruje symetrickým klíčem. To má za následek, že pokud se data v souboru k zašifrování opakují, tak se budou opakovat i v zašifrovaném souboru. Otevřená a zašifrovaná data budou mít velmi podobnou strukturu. V dnešní době, kdy je většina dat strukturována to může útočníkovi velmi usnadnit práci. Proto se v dnešní době téměř nepoužívá.

Následující obrázek 29 ukazuje, jak vypadá zašifrované 32 bitové pole obsahující dva identické 16 bitové bloky: `byte[] vstup = new byte[] {(byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8, (byte)9, (byte)10, (byte)11, (byte)12, (byte)13, (byte)14, (byte)15, (byte)16, (byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8, (byte)9, (byte)10, (byte)11, (byte)12, (byte)13, (byte)14, (byte)15, (byte)16}`;

Blok 1: (byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8, (byte)9, (byte)10, (byte)11, (byte)12, (byte)13, (byte)14, (byte)15, (byte)16,

Blok2: (byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8, (byte)9, (byte)10, (byte)11, (byte)12, (byte)13, (byte)14, (byte)15, (byte)16

Použitý šifrovací klíč byl třiceti dvou bajtový.



Obr. 29 32 bitové pole, které obsahuje dva identické 16 bitové bloky [zdroj vlastní]

Situace je ještě, horší bude-li použita šifra DES a osmi bajtový klíč. Vstupní pole použité v ukázce je: `byte[] vstup = new byte[] {(byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8, (byte)1, (byte)2, (byte)3, (byte)4, (byte)5, (byte)6, (byte)7, (byte)8}`; Jak je



```
        (byte) 5, (byte) 6, (byte) 7, (byte) 8,
        (byte) 9, (byte) 10, (byte) 11, (byte) 12,
        (byte) 13, (byte) 14, (byte) 15, (byte) 16};

byte[] inicializacniVektor = new byte[] {(byte) 8, (byte) 15,
        (byte) 20, (byte) 1, (byte) 2, (byte) 8,
        (byte) 7, (byte) 1, (byte) 5, (byte) 3,
        (byte) 6, (byte) 7, (byte) 1, (byte) 2,
        (byte) 5, (byte) 3};

int delkaPBPK = 32;

byte[] poleBajtuProKlic = new byte[delkaPBPK];

for(int i = 0; i < delkaPBPK; i++)
{
    poleBajtuProKlic[i] = (byte) i;
}

try
{
    SecretKeySpec symetrickyKlic = new
        SecretKeySpec(poleBajtuProKlic,
            "AES");

    Cipher cipher =
        Cipher.getInstance("AES/CBC/NoPadding",
            "BC");

    IvParameterSpec ips =
        new IvParameterSpec(inicializacniVektor);

    cipher.init(Cipher.ENCRYPT_MODE, symetrickyKlic, ips);

    byte[] zasifrovanePoleBajtu =
        new
        byte[cipher.getOutputSize(vstup.length)];

    int delkaZPB =
    cipher.update(vstup, 0, vstup.length, zasifrovanePoleBajtu,
    0);

    delkaZPB += cipher.doFinal(zasifrovanePoleBajtu, delkaZPB);

    tv1.setText("Kryptogram:\n" +
        this.byteNaString(zasifrovanePoleBajtu) +
        "\n\n" +
        "Velikost vstupního pole: " +
        String.valueOf(vstup.length) +
        "\n" + "Velikost zašifrovaného pole: " +
        String.valueOf(zasifrovanePoleBajtu.length) +
        "\n\n");
}
catch (Exception e)
{
    tv1.setText("Při šifrování se vyskytka chyba: " +
        e.toString());
}
}
```

Pojďme se podívat, co se změnilo oproti předchozí ukázce kódu. Přibylo pole inicializačního vektoru (`inicializacniVektor`). Dále přibyla instance třídy `IvParameterSpec`, která jako parametr konstrukturu akceptuje pole bajtů inicializačního vektoru. Poslední změnou je rozšíření konstrukturu objektu `Cipher`, o již zmiňovanou instanci třídy `IvParameterSpec`:

```
cipher.init(Cipher.ENCRYPT_MODE, symetrickyKlic, ips);
```

V knize [13] je uvedeno upozornění, že se často zapomíná na inicializační vektor. Chyba se projevuje tak, že první blok je rozšifrován na nesmyslnou posloupnost bajtů, zatím co ostatní bloky jsou rozšifrovány v pořádku.



Obr. 31 Pole zašifrované pomocí AES a CBC bez opakování [zdroj vlastní]



Obr. 32 Pole zašifrované pomocí DES a CBC bez opakování [zdroj vlastní]

### 2.2.5 Symetrická kryptografie založená na heslech

V jazyce Java existuje více způsobů, jak provozovat symetrickou kryptografii založenou na heslech. Tento text se bude zabývat způsobem, který je ve světě profesionálních

programátorů v jazyce Java nejpoužívanější. Způsob je založen na tzv. Message-Digest algoritmu. Názvem Message-Digest algoritmy se nazývá množina hašovacích funkcí. Ty umí z libovolného počtu bajtů daných na vstup vytvořit výstupní posloupnost bajtů, která má vždy stejnou velikost. Výstupní posloupnost může být chápána jako jakási zkratka, miniatura vstupní posloupnosti. Mezi důležité vlastnosti hašovacích funkcí patří:

- I malá změna vstupní posloupnosti se zásadním způsobem projeví na výstupní posloupnosti.
- Z výstupní posloupnosti nelze odvodit vstupní posloupnost.

### Způsob vytvoření klíče z hesla

Pomocí hašovací funkce se ze zadaného hesla vytvoří pole bajtů, z kterého je vytvořen jednorázový klíč použitelný jen pro aktuální šifrování a neukládá se do souboru. Při dešifrování se stejným způsobem vytvoří jednorázový klíč pro dešifrování. Je pochopitelně stejný jako jednorázový klíč, který byl vytvořen při šifrování.

### Postup

Za prvé je potřeba od uživatele získat heslo v proměnné typu String například pomocí AlertDialogu. V ukázce je takto připravené heslo uloženo v proměnné `heslo`. Proměnná `heslo` je převedena na pole bajtů. To je vstupní posloupnost hašovací funkce. V dalším kroku je pomocí metody `getInstance` vytvořena instance třídy `MessageDigest`. Instance uvedené třídy poskytují vytvářeným programům funkcionalitu hašovacích funkcí. Metoda `getInstance` má dvě varianty:

- `getInstance(String algorithm)` [26]. V ukázce bylo instanci třídy `MessageDigest` sděleno, že má být použita 256 bitová verze SHA-2 hašovací funkce. Vytvářený klíč bude 256 bitový. Jedná se o ideální řešení.
- `getInstance(String algorithm, Provider provider)` [27]. Tato verze metody `getInstance` umožňuje specifikovat poskytovatele požadované funkcionality, stejně jako to umožňuje metoda `getInstance` třídy `Cipher`.

Metoda `digest` vrací pole bajtů, které bude mít v tomto případě 32 bajtů (tj. 256 b).

Symetrický klíč je vytvořen stejně jako v předchozích ukázkách. Jen je na jeho tvorbu použito výstupní pole z hašovací funkce (`klíčPripravaZHesla`).

Java používá na správu paměti garbage collector. Z bezpečnostních důvodů je nutné provést destrukci polí `hesloPoleBajtu` a `klicPripravaZHesla` v okamžiku, kdy nejsou pro chod programů potřeba tedy hned po vytvoření klíče.

Ukázka vytvoření klíče pomocí hesla:

```
// VYTVORENI KLICE POMOCI HESLA //////////////////////////////////////  
byte[] hesloPoleBajtu = (heslo).getBytes("UTF-8");  
  
MessageDigest sha = MessageDigest.getInstance("SHA-256");  
  
byte[] klicPripravaZHesla = sha.digest(hesloPoleBajtu);  
  
Key symetrickyKlic = new SecretKeySpec(klicPripravaZHesla, "AES");  
  
hesloPoleBajtu = null;  
klicPripravaZHesla = null;
```



Obr. 33 Ukázka šifrování s heslem [zdroj vlastní]

## 2.2.6 Rozdíly v šifrování a dešifrování

Díky používání popsané architektury JCA a JCE v úvodu kapitoly, jsou rozdíly mezi šifrováním a dešifrováním opravdu minimální. Pokud se v předchozí ukázce změní operační režim metody `Cipher.init()` z `ENCRYPT_MODE` na `DECRYPT_MODE`, je dešifrovací kód hotov. Samozřejmě je potřeba si připravit pole bajtů, které bude

zašifrované stejným klíčem. Pro lepší srovnání šifrování a dešifrování je připravena funkční ukázka:

```
public void desifruj (View view)
{
    byte[] inicializacniVektor = new byte[] {(byte)8, (byte)15,
        (byte)20, (byte)1, (byte)2, (byte)8,
        (byte)7, (byte)1, (byte)5, (byte)3,
        (byte)6, (byte)7, (byte)1, (byte)2,
        (byte)5, (byte)3};

    int delkaPBPK = 32;

    byte[] poleBajtuProKlic = new byte[delkaPBPK];

    for(int i = 0; i < delkaPBPK; i++)
    {
        poleBajtuProKlic[i] = (byte)i;
    }

    try
    {
        SecretKeySpec symetrickyKlic =
            new SecretKeySpec(poleBajtuProKlic, "AES");

        Cipher cipher =
            Cipher.getInstance("AES/CBC/NoPadding",
                "BC");

        IvParameterSpec ips =
            new IvParameterSpec(inicializacniVektor);

        cipher.init(Cipher.DECRYPT_MODE, symetrickyKlic, ips);

        byte[] zasifrovanePoleBajtu =
            zasifrujPole(inicializacniVektor,
                symetrickyKlic);

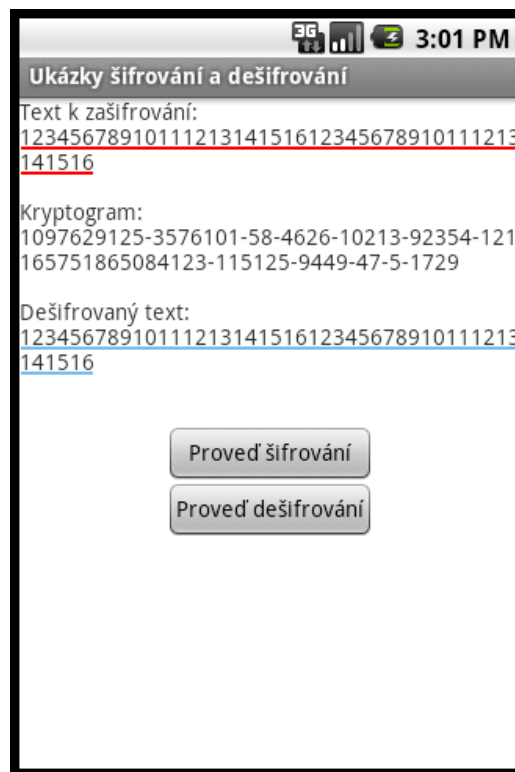
        byte[] desifrovanePoleBajtu =
            new byte[cipher.getOutputSize(zasifrovanePoleBajtu.length)];

        int delkaZPB =
            cipher.update(zasifrovanePoleBajtu, 0,
                zasifrovanePoleBajtu.length, desifrovanePoleBajtu, 0);

        delkaZPB += cipher.doFinal(desifrovanePoleBajtu, delkaZPB);

        tv1.setText(tv1.getText() + "Dešifrovaný text:\n" +
            this.byteNaString(desifrovanePoleBajtu) +
            "\n\n");
    }
    catch (Exception e)
    {
        tv1.setText("Při šifrování se vyskytka chyba: " +
            e.toString());
    }
}
```

Na obrázku 34 je vidět text, který má být zašifrován, jeho zašifrovaná podoba a dešifrovaný text.



Obr. 34 Zašifrovaný text, kryptogram a dešifrovaný text [zdroj vlastní]

## 2.3 Asymetrická kryptografie

Symetrická a asymetrická kryptografie se od sebe velmi algoritmičtě liší, jak bylo popsáno v první kapitole. JCA a JCE jsou navrženy tak, že poskytují vývojářům programujícím v jazyce Java abstraktní vrstvu. Abstraktní vrstva představuje bariéru, za kterou jsou před aplikačními programátory skryty implementace jednotlivých šifrovacích algoritmů. To je důvodem malých rozdílů mezi symetrickou a asymetrickou kryptografií v aplikačním kódu.

### 2.3.1 Výroba veřejného a soukromého klíče

Prakticky jediným rozdílem je výroba klíčů. Odlišnost je dána množstvím používaných klíčů. Symetrická kryptografie používá jediný klíč, zatím co asymetrická kryptografie používá klíče dva. Klíče lze vyrobit více způsoby. Výroba klíčů se vždy provádí pomocí třídy KeyFactory nebo jejich potomků.

## Třída KeyFactory

Třída KeyFactory je poskytovatelem abstraktní vrstvy. Umožňuje převést klíče, které byly vytvořeny mimo prostředí daného poskytovatele po podoby použitelné v prostředí daného poskytovatele. Následně lze převedené klíče vyexportovat. Třída KeyFactory umí i vytvářet nové veřejné a soukromé klíče podle zadaných specifikací. Nové instance třídy KeyFactory se nevytvářejí pomocí konstruktoru, ale pomocí metody getInstance () stejně jako ve třídě Cipher. Podrobnější informace o metodě getInstance () jsou uvedeny ve zdroji [28].

### Metoda generatePrivate

Metoda generatePrivate vytváří objekty třídy PrivateKey (soukromé klíče) podle specifikace (materiálu na výrobu klíče) předaných v parametru keySpec , který je instancí třídy KeySpec [29].

### Metoda generatePublic

Metoda generatePublic vytváří objekty třídy PublicKey (veřejné klíče) podle specifikace (materiálu na výrobu klíče) předané v parametru keySpec , který je instancí třídy KeySpec [30].

### Metoda getKeySpec

Metoda getKeySpec vrací specifikaci (materiál na výrobu klíče) objektu klíče, která byla metodě předána jako parametr key. Návrat specifikace probíhá prostřednictvím druhého parametru, kterým je keySpec [31].

### Metoda translateKey

Klíče jednoho poskytovatele mohou být pro jiné poskytovatele neznámé nebo potenciálně nedůvěryhodné. V uvedených případech je nutné provést překlad klíče pomocí metody translateKey [32].

V ukázce jsou klíče vyrobeny následujícím způsobem:

1. pomocí metody getInstance je vytvořena instance třídy KeyPairGenerator. Jako parametr je metodě getInstance předán typ šifrovacího algoritmu a poskytovatel [33].

2. metoda initialize provede explicitní inicializaci instance třídy KeyPairGenerator. Nastaví délku klíčů na 2048 bitů. Pokud by inicializace nebyla provedena, poskytovatel by nastavil výchozí délku klíče [34].
3. instance třídy KeyPairGenerator se používá k vygenerování instance třídy KeyPair (páru klíčů).
4. instance třídy KeyPair prostřednictvím metod getPublic a getPrivate (zdeděných z KeyFactory) vytvoří soukromý a veřejný klíč.

```
private Boolean vygenerujKlice()
{
    Boolean vysledek;

    try
    {
        KeyPairGenerator kpg =
            KeyPairGenerator.getInstance("RSA", "BC");

        kpg.initialize(2048);

        KeyPair keyPair = kpg.genKeyPair();

        verejnyKlic = keyPair.getPublic();

        soukromyKlic = keyPair.getPrivate();

        vysledek = true;
    }
    catch (Exception e)
    {
        vysledek = false;
        tvl.setText("Při vytváření klíčů se vyskytka chyba: " +
            e.toString());
    }

    return vysledek;
}
```

### 2.3.2 RSA šifrování

Na níže uvedené ukázce šifrování je patrné, že se asymetrické šifrování jen velmi málo liší od symetrického šifrování.

```
public void zasifrujRSA(View view)
{
    try
    {
        if(verejnyKlic == null)
        {
            vygenerujKlice();
        }

        Cipher cipher = Cipher.getInstance("RSA");
```

```

cipher.init(Cipher.ENCRYPT_MODE, verejnyKlic);

zasifrovanePole = cipher.doFinal(vstup);

tv1.setText("Otevřený text: " + byteNaString(vstup) +
           "\n\nKryptogram: " +
           this.byteNaString(zasifrovanePole));
}
catch (Exception e)
{
    tv1.setText("Při šifrování se vyskytka chyba: " +
              e.toString());
}
}

```



Obr. 35 Ukázka RSA šifrování [zdroj vlastní]

### 2.3.3 RSA dešifrování

Z níže prezentované ukázky dešifrování je opět patrné, že se asymetrické dešifrování jen velmi málo liší od symetrického dešifrování.

```

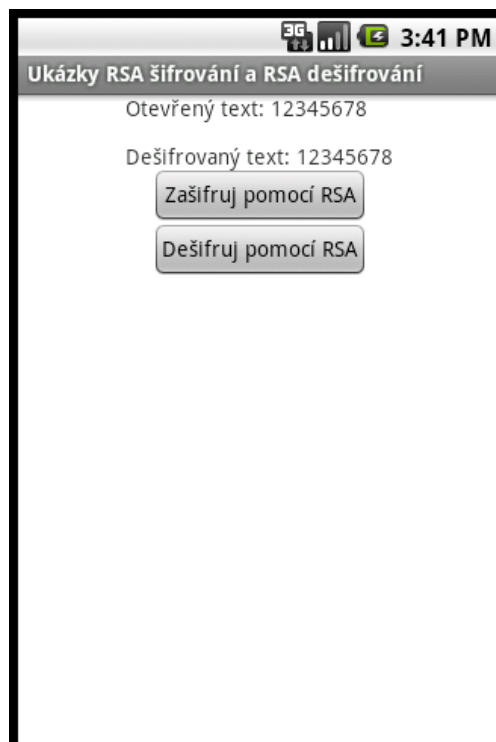
public void desifrujRSA(View view)
{
    try
    {
        if(zasifrovanePole != null)
        {
            if(soukromyKlic == null)

```

```
        {
            vygenerujKlice();
        }

        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, soukromyKlic);
        desifrovanePole = cipher.doFinal(zasifrovanePole);

        tv1.setText("Otevřený text: " +
            byteNaString(vstup) +
            "\n\nDešifrovaný text: " +
            this.byteNaString(desifrovanePole));
    }
}
catch (Exception e)
{
    tv1.setText("Při šifrování se vyskytka chyba: " +
        e.toString());
}
}
```



Obr. 36 Ukázka RSA dešifrování  
[zdroj vlastní]

## 2.4 Solení souborů

Představme si následující situaci: Aplikace volně stažitelná z internetu ke své práci potřebuje soubor, ve kterém jsou citlivá data. Například se může jednat o autentizační údaje pro spojení s registračním serverem, které aplikace musí znát ale uživatel/útočník

nikoliv. Když je soubor zavřený, je zašifrovaný. Co když program potřebuje soubor otevřít a pracovat s ním (číst ale i zapisovat)? Nejjednodušší je soubor rozšifrovat a uložit dočasně na lokálním úložišti. Po skončení práce je soubor opět zašifrován. Co když aplikace během práce s otevřeným (rozšifrovaným) souborem spadne? Například vlivem neošetřené výjimky nebo zásahem útočnicka, který provozuje danou aplikaci v emulátoru telefonu a sleduje virtuální úložiště. Soubor by zůstal rozšifrovaný a každý by si mohl jeho obsah přečíst. Právě v takových případech je vhodné použít solení souborů. Solení je založeno na exkluzivní disjunkci. Než se rozšifrovaný soubor uloží do lokálního úložiště, tak se posolí. Existují dva způsoby solení.

#### 2.4.1 Sůl je stejně velká jako soubor k posolení

Způsob solení, kdy je sůl stejně velká jako solený soubor, je považován za nejlepší. Ovšem pro aplikace běžící v telefonu s omezenou operační pamětí lze tento způsob solení použít jen pro soubory, u kterých se předpokládá, že nebudou větší než 4 MB. Solení je vhodné aplikovat například pro autentizaci a servisní komunikaci telefonu se serverem, protože jsou jen málo kdy větší než 1 MB na relaci.

Příklad osolení dvou bajtového souboru.

Rozšifrovaný soubor má dva bajty: 000110010011010.

Sůl byla zvolena: 0011010011011001.

Soubor k posolení

první bajt      druhý bajt

00011001	00110100
----------	----------

Sůl

první bajt      druhý bajt

00110100	11011001
----------	----------

*Obr. 37 Soubor určený k posolení  
a sůl [zdroj vlastní]*

Osolení souboru:

000110010011010 XOR 0011010011011001 = 0010110111101101

Soubor k posolení	00011001	00110100
Sůl	00110100	11011001
Posolený soubor	00101101	11101101

*Obr. 38 Osolení souboru [zdroj vlastní]*

Výsledek 0010110111101101 je uložen do souboru místo rozšifrovaného 000110010011010. Jestliže aplikace potřebuje s daty pracovat, načte je z posoleného souboru 0010110111101101 a opět provede operaci XOR se zvolenou solí:

0010110111101101 XOR 0011010011011001 = 0001100100110100

Tímto způsobem má aplikace k dispozici původní nezašifrovaná data. Pokud by aplikace spadla v okamžiku, kdy je soubor otevřený, útočník se k datům stejně nedostane, protože jsou posolená.

Ukázka kódu solení

```
try
{
    File souborKPosoleni =
        new File("/mnt/sdcard/aaa/souborKPosoleni.txt");

    FileInputStream fis = new FileInputStream(souborKPosoleni);

    byte[] poleKPosoleni = new byte[fis.available()];

    fis.read(poleKPosoleni);

    fis.close();

    // ABY SE POUZIVALA STALE STEJNA SUL
    if(sul == null)
    {
        // SUL MUSI BYT STEJNE DLOUHA JAKO poleKPosoleni.length
        VygenerujSul vs = new VygenerujSul(poleKPosoleni.length);

        sul = vs.getSul();
    }
}
```

```

PosolPoleBajtu ppb = new PosolPoleBajtu(poleKPosoleni, sul);

byte[] polePosolene = ppb.getPolePosolene();

File souborPosoleny =
    new File("/mnt/sdcard/aaa/souborKPosoleni.txt");

FileOutputStream fos = new FileOutputStream(souborPosoleny);

fos.write(polePosolene);
fos.flush();
fos.close();

tv1.setText(txtSouborNaString(souborPosoleny.getAbsolutePath()));
}
catch (Exception e)
{
    tv1.setText(e.toString());
}

```

V hlavní části programu se otevře soubor ["/mnt/sdcard/aaa/souborKPosoleni.txt"](#) představující rozšifrovaný soubor. Vytvoří se vstupní proud, pomocí kterého se načtou bajty otevřeného souboru do pole bajtů (poleKPosoleni). Vstupní proud již není potřeba, proto je uzavřen. Sůl musí být stejná jak pro posolení, tak pro odsolení. Nová sůl je vygenerována jen, když ještě žádná neexistuje. Sůl je generována prostřednictvím instance třídy VygenerujSul. Posolení se provede pomocí instance třídy PosolPoleBajtu. Výsledné posolené pole bajtů (polePosolene) je zapsáno výstupním proudem na místo původního souboru. Po dokončení zápisu je výstupní proud korektně ukončen.

Ukázka třídy VygenerujSul

```

import java.util.Random;

public class VygenerujSul
{
    // PROM
    int delka;

    // KONSTRUKTOR
    public VygenerujSul(int delkaSoli)
    {
        delka = delkaSoli;
    }

    public byte[] getSul()
    {
        byte[] sul = new byte[delka];
        new Random().nextBytes(sul);
    }
}

```

```

        return sul;
    }
}

```

Instance třídy `VygenerujSul` se starají o vytváření solí, kterým se v konstruktoru sdělí délka požadované soli. Nová sůl je vygenerována pomocí metody `getSul()`.

```

public class PosolPoleBajtu
{
    // PROM
    byte[] sul;
    byte[] poleKPosol;
    byte[] polePosolene;

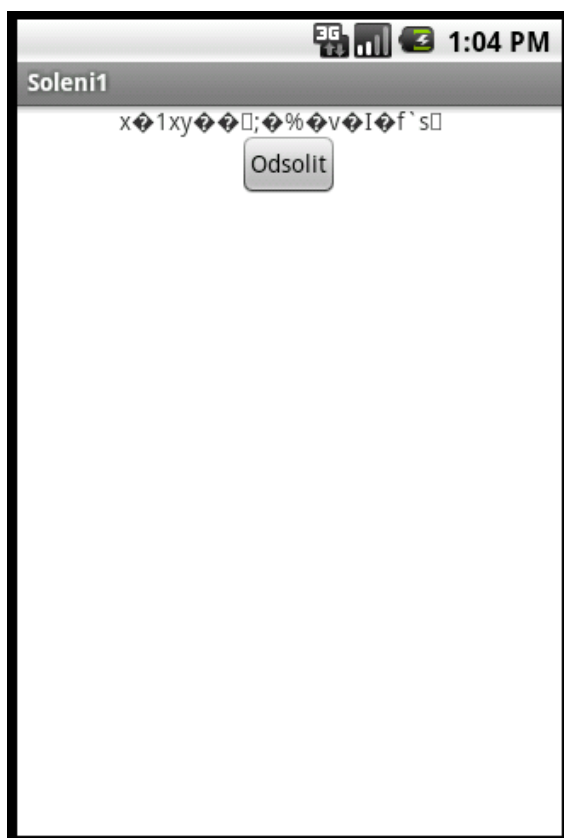
    // KONSTRUKTOR
    public PosolPoleBajtu(byte[] poleKPosoleni, byte[] poleSul)
    {
        sul = poleSul;
        poleKPosol = poleKPosoleni;
        polePosolene = new byte[poleKPosol.length];
        provedSoleni();
    }

    private void provedSoleni()
    {
        if(sul.length == poleKPosol.length)
        {
            for(int i = 0; i < poleKPosol.length; i++)
            {
                polePosolene[i] = (byte)((poleKPosol[i] ^ sul[i])
                    & 0x000000ff);
            }
        }
    }

    public byte[] getPolePosolene()
    {
        return polePosolene;
    }
}

```

Třída `PosolPoleBajtu` slouží k posolení pole bajtů zvolenou solí. Obojí se instancí třídy `PosolPoleBajtu` předá v konstruktoru. Samotné solení je provedeno privátní metodou `provedSoleni`, která je zavolána konstruktorem. Metoda `provedSoleni` proběhne jen, když má pole bajtů určených k posolení a sůl stejnou délku. Pokud mají různou délku, je vráceno místo osoleného pole původní neosolené pole. Je to ochrana před chybou v aplikačním programu používající třídy `VygenerujSul` a `PosolPoleBajtu`. Jestliže by programátor předložil konstruktoru pole bajtů určených k posolení a sůl různých délek, skončila by metoda `provedSoleni` chybou. Na obrázku 39 je vidět obsah osoleného souboru a na obrázku 40 je vidět obsah odsoleného souboru.



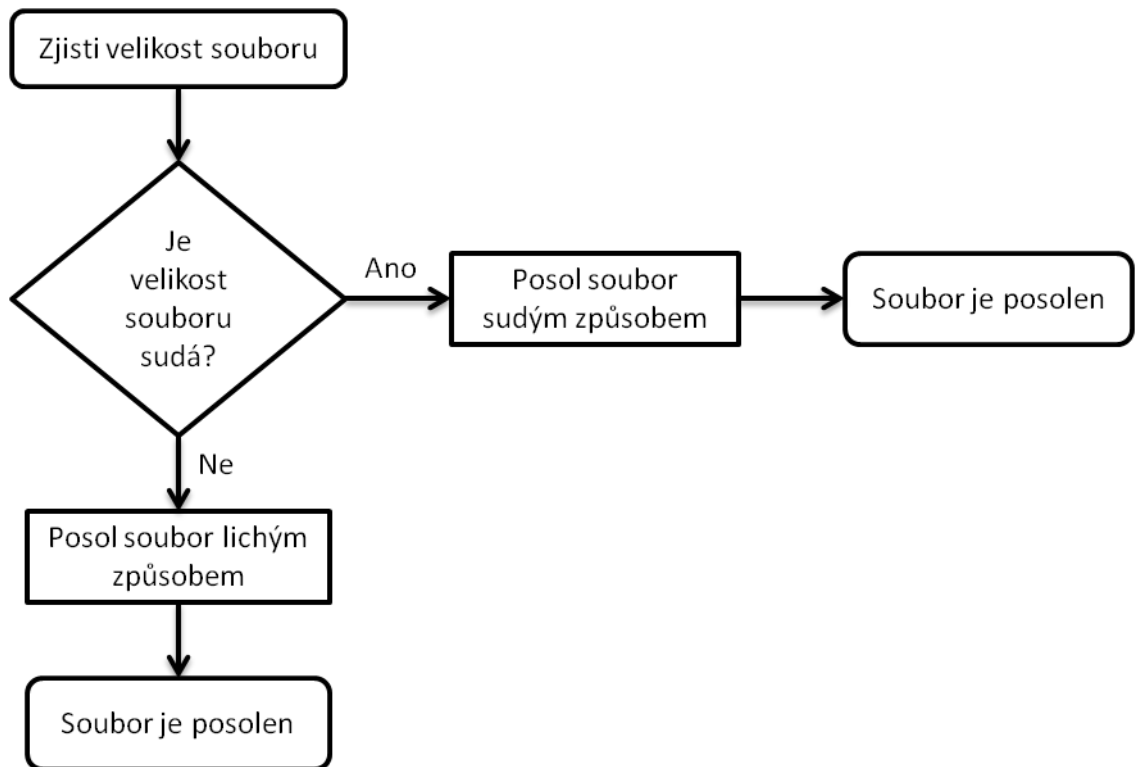
Obr. 40 Obsah osoleného souboru [zdroj vlastní]



Obr. 39 Obsah odsoleného souboru [zdroj vlastní]

#### 2.4.2 Sůl a soubor k posolení jsou různě velké

V programátorské praxi často nastane případ, že je potřeba posolit soubor větší než 4 MB. Pro takové případy je vhodné posolení souboru tzv. sudou solí. Postup je zachycen na obrázku 41.



Obr. 41 Soubor k posolení a sůl jsou různě velké [zdroj vlastní]

Naznačené řešení:

```

try
{
    File souborKPosoleni = new
File(Environment.getExternalStorageDirectory().getAbsolutePath().toString
() + "/souborKPosoleni.txt");

    FileInputStream fis = new FileInputStream(souborKPosoleni);

    String.valueOf(fis.available());

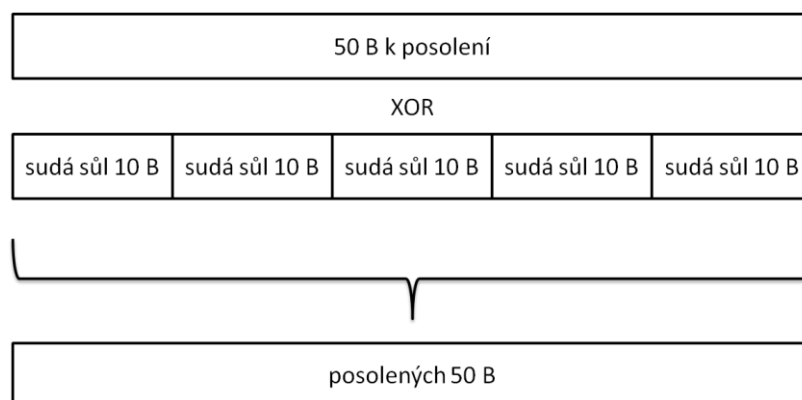
    if(fis.available() % 2 == 0)
    {
        // posol soubor sudým způsobem
    }
    else
    {
        // posol soubor lichým způsobem
    }

    fis.close();
}
catch(Exception e)
{
    tv1.setText(e.toString());
}
  
```

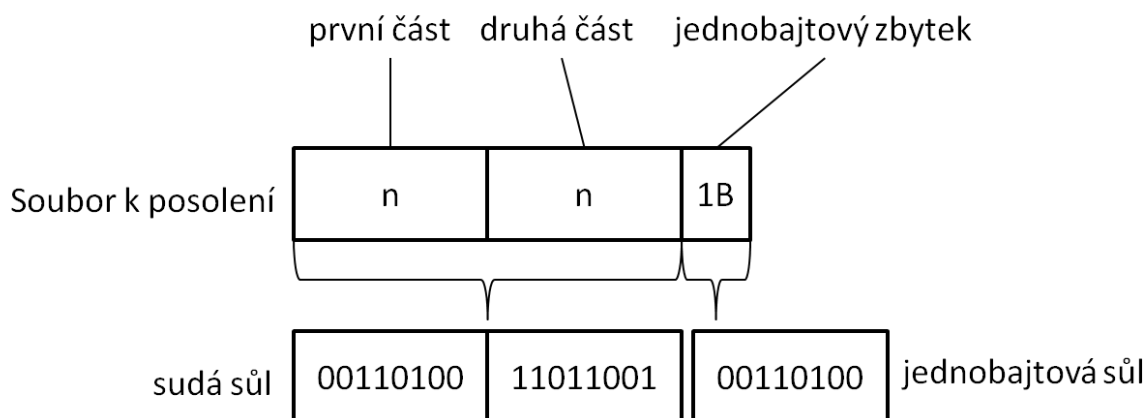
Nejprve se otevře soubor k posolení, ze kterého je vytvořen vstupní proud. Pokud je počet bajtů souboru dělitelný dvěma beze zbytku, pak je počet bajtů sudý. Soubor se bude solit sudým způsobem. Pokud není počet bajtů souboru dělitelný dvěma beze zbytku, pak je počet bajtů lichý. Soubor se bude solit lichým způsobem.

### Lichý způsob solení

Počet bajtů souboru lze rozdělit na dvě stejné části o  $n$  bajtech a jednobajtový zbytek. Přičemž  $n$  může být sudé číslo nebo liché číslo. V případě, že se budou solit obě části o  $n$  bajtech dohromady, bude se solit  $2n$  bajtů, tedy vždy sudý počet bajtů. Blok  $2n$  bajtů lze po částech posolit, jak je vidět na obrázku 42. Soubor o velikosti 51 B, lze rozdělit na blok o velikosti 50 B a jednobajtový zbytek. Padesáti bajtový blok je možné posolit postupným pětinasobným aplikováním téže soli o velikosti 10 B. Jednobajtový zbytek je nakonec osolen speciální jednobajtovou solí. Obě části se spojí dohromady a zapíše se do souboru.



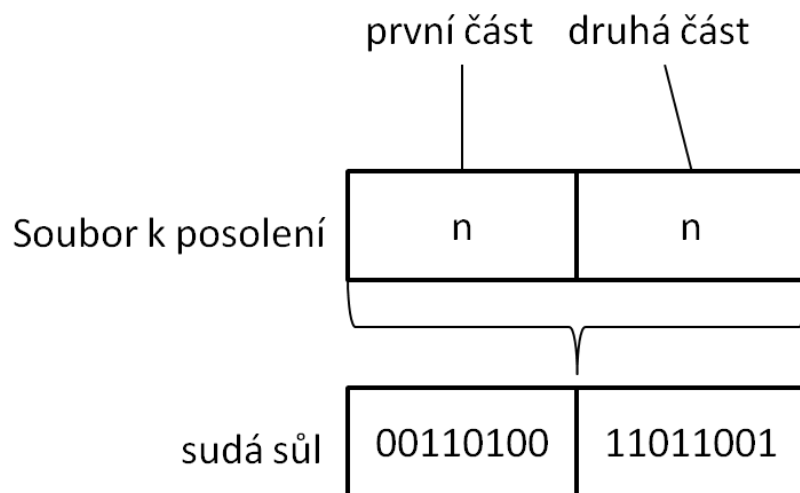
Obr. 42 Posolení 50 B solí o velikosti 10 B po částech



Obr. 43 Lichý způsob solení [zdroj vlastní]

**Sudý způsob solení**

Sudý blok se dá posolit podobným způsobem. Jediný rozdíl je, že odpadá solení jednobajtového zbytku.



*Obr. 44 Sudý způsob solení*

## **II. PRAKTICKÁ ČÁST**

### 3 OMEZENÍ KRYPTOGRAFIE NA MOBILNÍCH PLATFORMÁCH

#### 3.1 Omezení dané současnou strukturou hardware na trhu

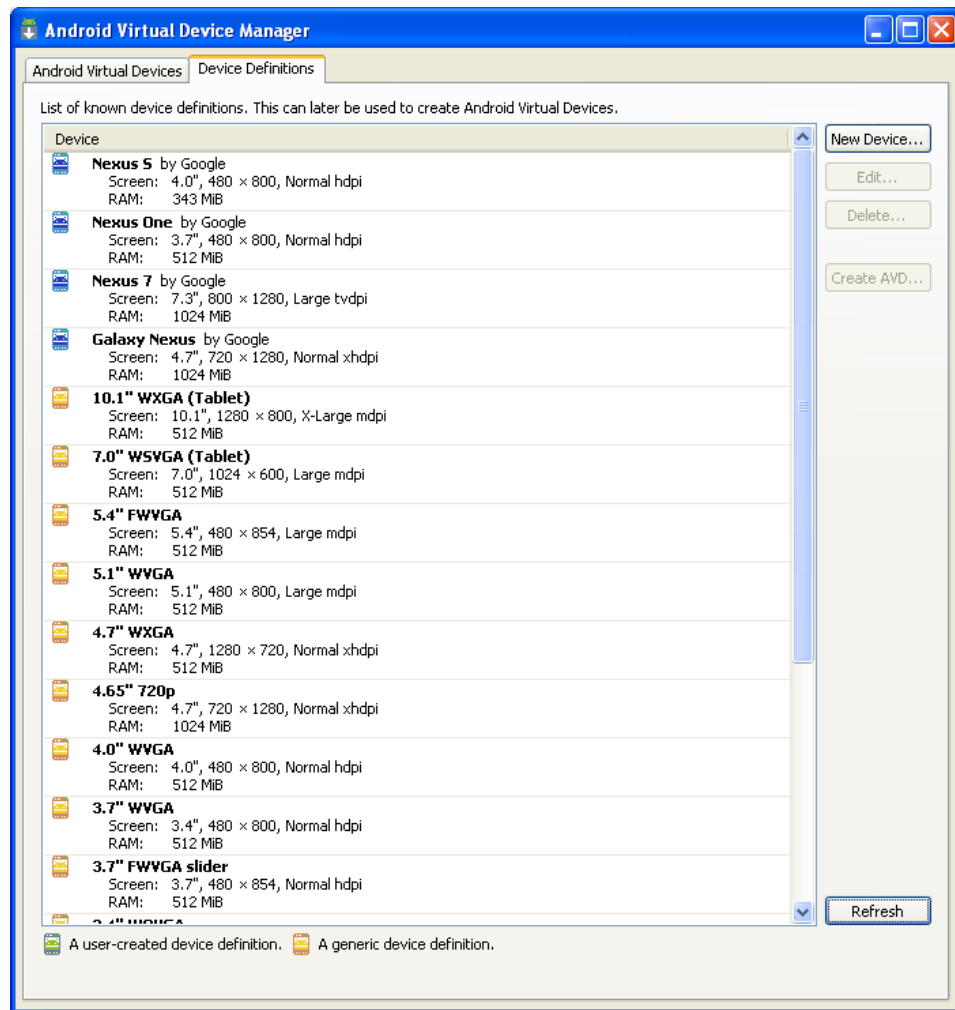
Operační systém Android běží na celé řadě mobilních zařízení, která jsou velmi různorodá. Na jedné straně se jedná o výkonné telefony typu Samsung Galaxy S III a Nexus 4, na druhé straně jsou to levné a málo výkonné telefony jako Huawei Ideos X1 a Gigabyte GSmart G1342. Operační systém Android je také provozován na velkém množství starších zařízení jako například LG Optimus One. Všechna uvedená zařízení se od sebe liší:

- Počtem a výkonem jader procesorů. Od jedno vláknových procesorů pracujících na frekvenci 800 MHz, u nichž použití vláken nepřinese navýšení výpočetního výkonu, až po procesory které mají čtyři standardní pracovní jádra ARM Cortex-A9 r4 pracující na frekvenci 2,3 GHz a další páté jádro určené pro běh nenáročných aplikací a šetřící baterii telefonu [35].
- Velikost operační paměti je v rozsahu od 156 MB do 2 GB.
- Velikost obrazovky je od 2,7 palce do 5,5 palce pro telefony a od 7 palců do 11,6 palců pro tablety.
- Verze operačního systému Android je od 2.1 Eclair až po 4.2 Jelly Bean.

Chce-li programátor napsat aplikaci, která bude komerčně úspěšná a bude správně fungovat na většině zařízení, pak musí počítat z výše uvedenými úzkými místy. Již při návrhu aplikace by měl vývojář definovat omezení a počítat s tím, že by aplikace měla bezproblémově fungovat na telefonu o následujících parametrech:

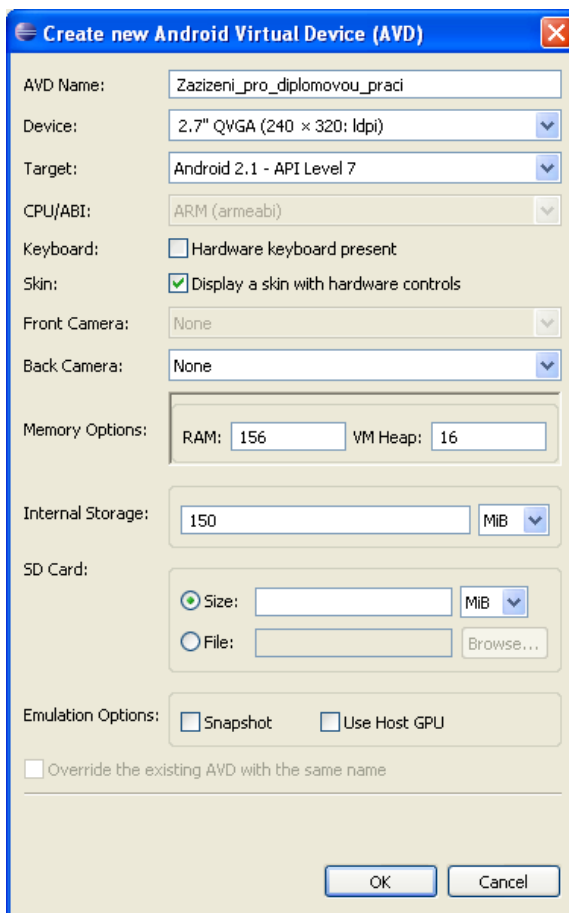
- CPU 1 jádro, frekvence 800 MHz;
- 156 RAM;
- 2,7 palcovým displejem;
- Operačním systémem Android 2.1 Eclair.

Každý vývojář nemá k dispozici telefon uvedených parametrů, na kterém by mohl své aplikace testovat. Firma Google poskytla řešení tohoto problému v podobě programu Android Virtual Device Manager. V programu Android Virtual Device Manager lze vytvářet virtuální telefony požadovaného hardware nebo využít nabídky připravených telefonů a tabletů. Na obrázku 45 je vidět program Android Virtual Device Manager. Vybraná zařízení je možné virtualizovaně provozovat k testování na počítači, na kterém jsou mobilní aplikace vytvářeny.



Obr. 45 Program Android Virtual Device Manager [zdroj vlastní]

Na obrázku 46 je vidět vytvoření virtuálního telefonu, který v roce 2013 představuje virtuální referenční telefon s minimálními hardwarovými požadavky. Prezentovaná omezení jsou závazná i pro mobilní kryptografické aplikace.



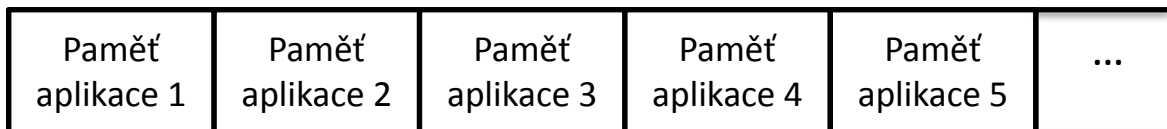
Obr. 46 Vytváření virtuálního zařízení  
[zdroj vlastní]

### 3.2 Omezení vycházející z vlastností operačního systému Android

Operační systém Android byl původně provozován na telefonech. Teprve od verze 3.0 Honeycomb byla přidána podpora velkých obrazovek a přidaly se tablety [36]. I v současné době je stále nejvíce instalací operačního systému Android v mobilních telefonech. Prioritou telefonu je telefonování. Uživatelé telefonních přístrojů očekávají, že bude možné telefonovat či přijmout telefonní hovor za všech okolností. Operační systém určený pro mobilní telefony by se nikdy nestal populární, kdyby aplikacím umožňoval vytěžovat procesor takovým způsobem, že by nebylo možné přijmout telefonní hovor. Případně by aplikace zabíraly operační paměť tak, že by se telefon neustále restartoval. Proto je operační systém navržen tak, aby špatně napsané aplikace nebo výpočetně či paměťově náročné aplikace držel v přesně daných mezích. U kryptografických aplikací se problémy týkají hlavně operační paměti.

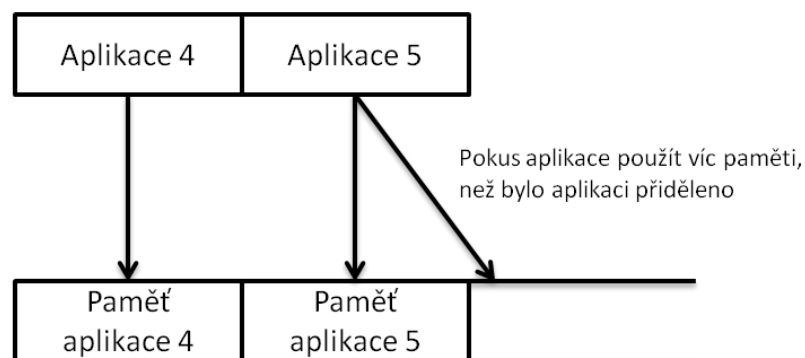
Operační systém Android přiděluje aplikacím přesně daný prostor. Velikost přiděleného prostoru závisí na fyzické velikosti operační paměti daného zařízení. Je dobré si uvědomit, že ještě v dnešní době má většina zařízení s operačním systémem Android velikost operační paměti ve srovnání s desktopey značně omezenou.

#### Část operační paměti, přidělené aplikacím



Obr. 47 Přidělování operační paměti aplikacím [zdroj vlastní]

Požaduje-li aplikace více prostoru než má přiděleno a pokusí se obsadit další paměť, je aplikace operační systém okamžitě ukončena. Viz obrázek 48.



Obr. 48 Pokus aplikace o použití další paměti [zdroj vlastní]

Pro ukázky šifrování v kapitole 2 byl použit jazyk Java. Jazyk funguje na desktopech i v mobilních zařízeních. Šifrování probíhá načtením souboru do pole bajtů, které se následně od začátku do konce zašifruje. Java běžící na desktopu nemá s tímto postupem problém. Protože například operační systém Windows odstránkovává nepoužívané části operační paměti do stránkovacího souboru. Při přerušení nazývaném výpadek stránky se načte požadovaná stránka ze stránkovacího souboru zpět do operační paměti. Každý soubor pocít, že má celou operační paměť jen pro sebe. V operačním systému Android se z výše uvedených důvodů přiděluje operační paměť jinak.

Z tohoto důvodu bude kód ukázek z kapitoly 2 fungovat jen pro soubory do určité velikosti. Velikost je u každého zařízení jiná v závislosti na velikosti operační paměti. Například pro telefonu LG Optimus One je mezní velikost 9 MB. Do této velikosti funguje

šifrování bezchybně, jak je vidět v následující ukázce. Soubor k zašifrování má velikost 4410036 B.



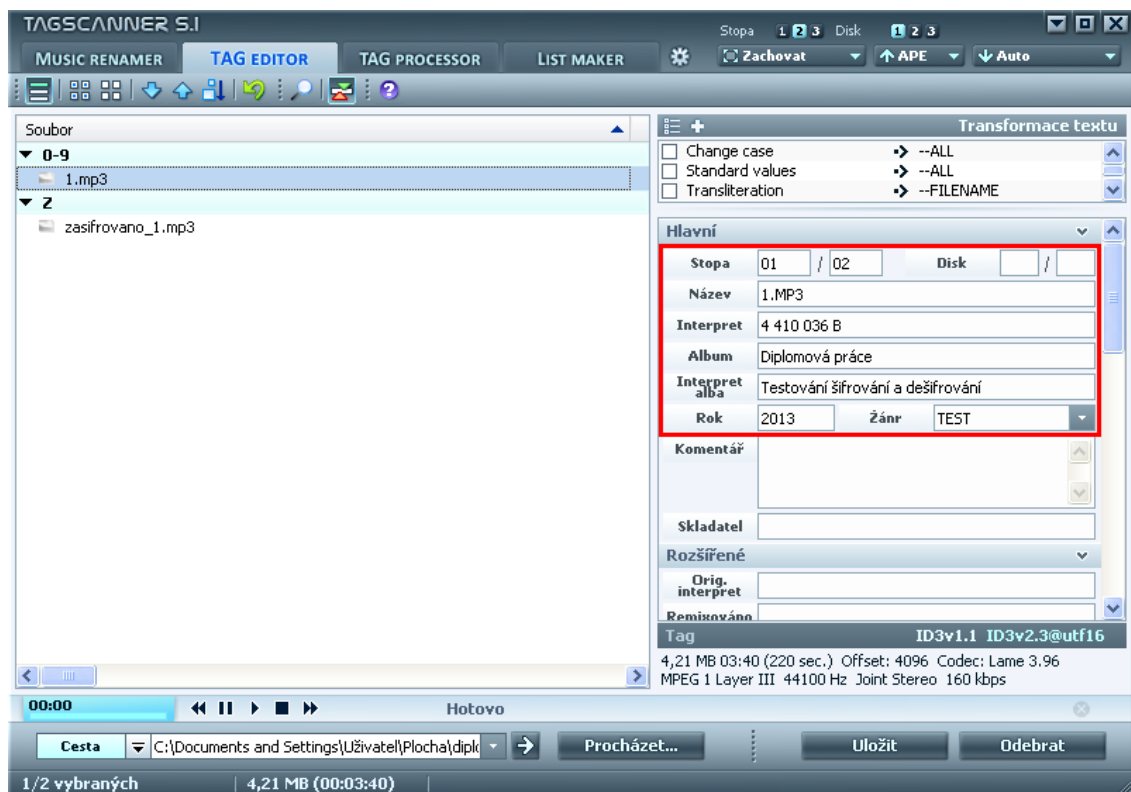
*Obr. 49 Ukázka zašifrování souboru o velikosti 4410036 B [zdroj vlastní]*

Po kliknutí na tlačítko „Proved' šifrování“ proběhlo zašifrování a uložení souboru. Na obrázku 50 je vidět, že zašifrovaný soubor zasifrovano\_1.mp3 má velikost 441048 B. Proběhl tedy padding posledního neúplného bloku, neboť  $4410036 \bmod 16 = 4$ . Byla použita AES šifra s velikostí bloku 16 B. Z toho plyne, že je potřeba zarovnat poslední blok dvanácti bajty:  $4410036 + 12 = 4410048$ .

Name	Size	Date	Time	Permissions	Info
klíce		2013-02-24	16:47	d---rwxr-x	
media		2011-10-14	13:41	d---rwxr-x	
musicdownload		2011-10-14	13:43	d---rwxr-x	
osmdroid		2011-10-21	23:27	d---rwxr-x	
ringtones		2011-10-14	13:43	d---rwxr-x	
smazat		2012-12-11	18:24	d---rwxr-x	
temp		2012-07-27	21:53	d---rwxr-x	
test_sifrovani		2013-03-15	16:13	d---rwxr-x	
1.mp3	4410036	2013-03-15	16:08	----rwxr-x	
klíčAES.aes	16	2013-03-14	20:24	----rwxr-x	
zasifrovano_1.mp3	4410048	2013-03-15	16:13	----rwxr-x	
tmp		2012-07-10	14:22	d---rwxr-x	
widget		2011-10-14	13:43	d---rwxr-x	
wiglewifi		2011-10-21	23:38	d---rwxr-x	
zipTest		2013-03-01	18:35	d---rwxr-x	
secure		1980-01-06	01:00	drwx-----	
proc		1970-01-01	01:00	dr-xr-xr-x	
root		2011-09-19	05:33	drwx-----	
sbin		1970-01-01	01:00	drwxr-x---	
sdcard		1980-01-06	01:00	lrwxrwxrwx	-> /mnt/sd...

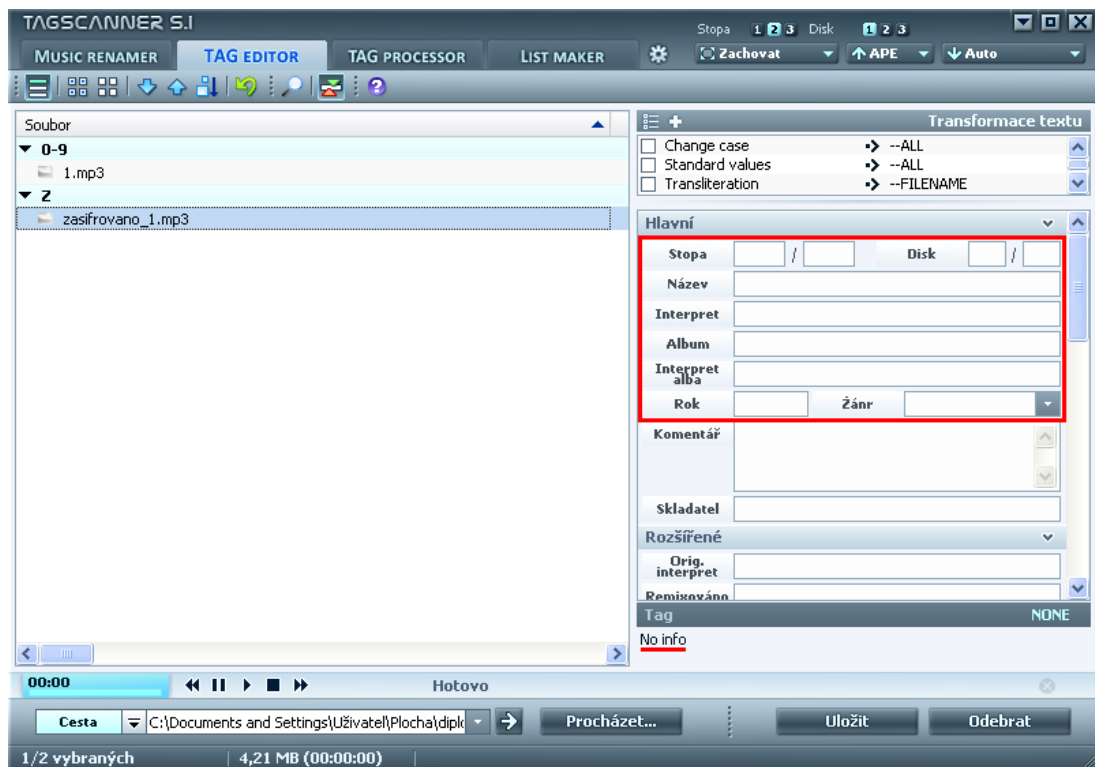
Obr. 50 Velikost souboru před a po zašifrování [zdroj vlastní]

Soubory 1.mp3 a zasifrovano\_1.mp3 byly staženy do počítače. Na obrázku 51 je vidět, že hlavička nezašifrovaného souboru 1.mp3 je čitelná.



Obr. 51 Hlavička nezašifrovaného souboru 1.mp3 je čitelná [zdroj vlastní]

Z obrázku 52 je patrné, že hlavička zašifrovaného souboru není čitelná. To je důkaz, že byl souboru korektně zašifrován.



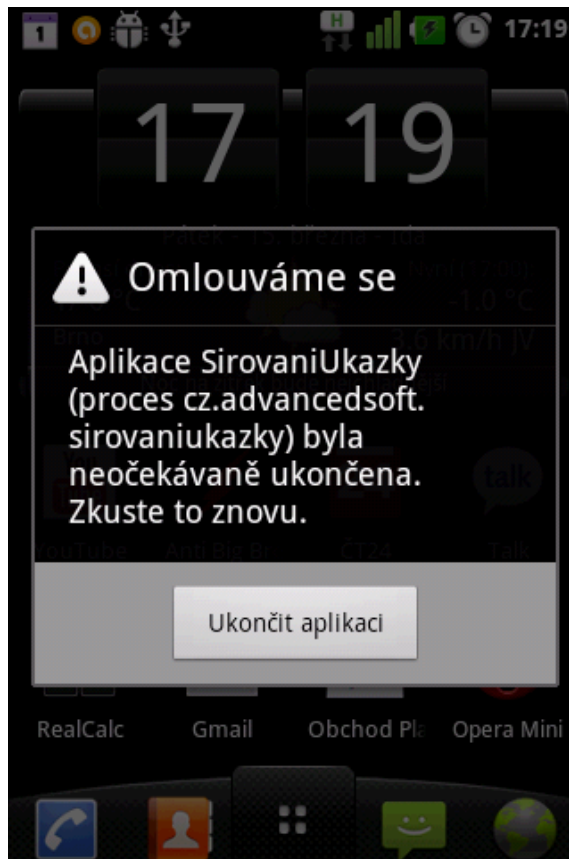
Obr. 52 Hlavička zašifrovaného souboru není čitelná [zdroj vlastní]

Na další ukázce je proveden pokus o zašifrování souboru 2.mp3, který má velikost 15932229 B. Zašifrování je provedeno stejným programem za shodných podmínek. Jediný rozdíl je ve velikostech souborů 1.mp3 a 2.mp3.



*Obr. 53 Ukázka zašifrování souboru  
o velikosti 15932229 B [zdroj vlastní]*

Na obrázku 54 je vidět, že byla šifrovací aplikace bez jakéhokoliv varování ukončena. Je nutné se podívat do výpisu LogCat, co bylo příčinou.



Obr. 54 Pád aplikace při šifrování souboru o velikosti 15932229 B [zdroj vlastní]

Obrázek 55 představuje výpis z LogCat. Z výpisu je zřejmé, že aplikace požadovala více operační paměti, než jí mohl operační systém přidělit (chyba OutOfMemoryError). Aplikace byla striktně ukončena.

```
at android.view.View$1.onClick(View.java:2139)
... 11 more
Caused by: java.lang.OutOfMemoryError
at org.bouncycastle.jce.provider.JCEBlockCipher.engineDoFinal(JCEBlockCipher.java:698)
at javax.crypto.Cipher.doFinal(Cipher.java:1090)
```

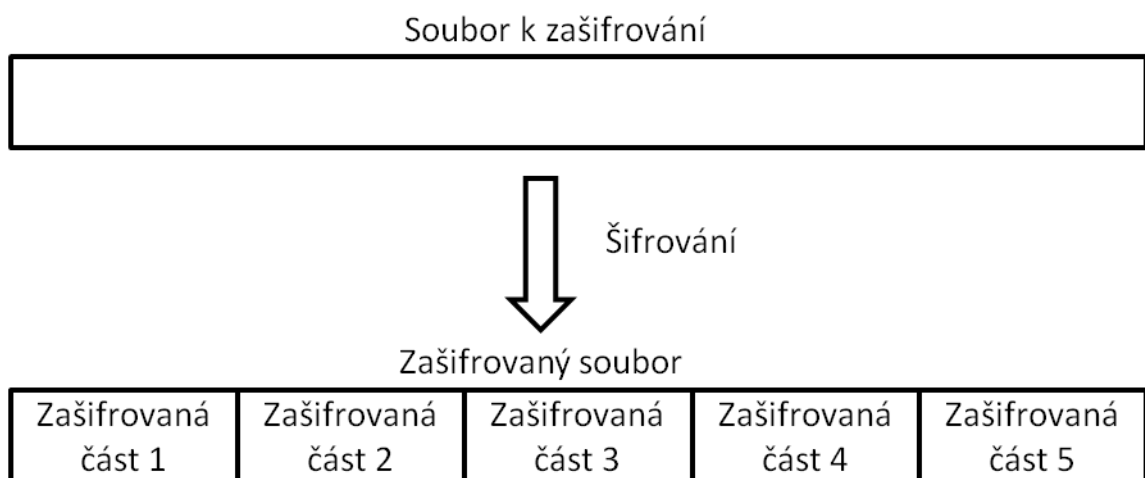
Obr. 55 Výpis z LogCat [zdroj vlastní]

### 3.3 Úprava šifrování a dešifrování

Z výše uvedených ukázek je patrné, že je potřeba řešení fungující nezávisle na velikosti operační paměti mobilních zařízení (od 156 MB do 2 GB). Řešení spočívá v tom, že nešifrují soubory jako celek ale po jednotlivých částech.

Nejprve načte třída `FileInputStream` ze souboru určeného k zašifrování malou část dat do pole bajtů. Po-li bajtů, které postupně načítá části nějakého objektu, se ve světě Javy říká buffer. Třída `Cipher` se postará o zašifrování obsahu bufferu. V posledním kroku je na konec zašifrovaného souboru připsán třídou `FileOutputStream` obsah bufferu. Postup se opakuje v cyklu, dokud má třída `FileInputStream`, co číst ze vstupního souboru. Obsah bufferu je v každém kroku přepsán nově načteným obsahem. Takže v operační paměti je jen malá část šifrovaného souboru o konstantní velikosti. Je důležité, aby velikost bufferu byla  $n$  násobkem bloku používané šifry.

Výsledný zašifrovaný soubor je složen z malých samostatně zašifrovaných částí, jak je vidět na obrázku 56.



Obr. 56 Soubor šifrovaný po částech [zdroj vlastní]

### 3.4 Výsledné řešení s ukázkou kódu

Výše uvedené řešení se zdá jasné, přesto jeho realizace nebyla jednoduchá a byla časově náročná, protože neexistuje řešení publikované v českém jazyce. Řešení prezentované

v anglické literatuře se zabývá pouze problematikou kryptografie na tzv. velké Javě<sup>1</sup> a nebrala v úvahu specifika operačního systému Android. Řešení naznačená v profesionální programátorské bázi stackoverflow [14] však neposkytovala uspokojivá řešení za všech okolností. V některých případech se objevily výjimky typu `NullPointerException`, jindy problémy s paddingem při dešifrování.

Navržené řešení je zcela unikátní a patří mezi hlavní přínosy diplomové práce. Navržené řešení bylo otestováno na celé řadě konkrétního hardwaru. Od telefonů s minimální konfigurací jako je LG Optimus One až po super výkonné tablety jako je Acer A511 s Tegrou 3. Řešení bylo otestováno na malých souborech, které byly menší než velikost bloku šifry stejně jako na velkých video souborech. Ve všech případech se navržené řešení osvědčilo.

V průběhu řešení diplomové práce byly zjištěny skutečnosti:

- Neřídít se údaji o zbývajících bajtech k zašifrování na základě informací od metody `Cipher.getOutputSize` ale ani na základě informací od metod `Cipher.update` a `Cipher.doFinal`. Pro bezproblémový chod šifrovacího programu je vhodné, aby si sám programátor napsal vlastní sledování zbývajících počtu bajtů.
- Rozdělit šifrování na dvě části.
  - V první části šifrovat část souboru, která je větší než blok šifry a je jeho  $n$ -násobkem. První část musí probíhat bez paddingu.
  - Druhá část je zašifrována zvlášť. Zašifruje poslední část souboru, která je menší nebo rovna velikosti bloku šifry. Nejlépe pomocí instance třídy, která by byla pro tento účel vytvořená.

```
// MOŽNÁ IMPLEMENTACE ČÁSTI 1
Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, symetrickyKlic);

int velikostBloku = cipher.getBlockSize();

double pocetBajtu = souborKZasifrovani.length();

byte[] buffer = null;

byte[] bufferZasif = null;
```

---

<sup>1</sup> Velká Java je Java provozovaná na osobních počítačích.

```
while(pocetBajtu > 0)
{
    if(pocetBajtu > velikostBloku)
    {
        buffer = new byte[velikostBloku];

        bufferZasif = new byte[velikostBloku];

        fis.read(buffer, 0, buffer.length);

        cipher.update(buffer, 0, velikostBloku, bufferZasif);

        pocetBajtu -= velikostBloku;
    }
    else if(pocetBajtu <= velikostBloku)
    {
        buffer = new byte[(int)pocetBajtu];

        fis.read(buffer, 0, buffer.length);

        ZasifrujAESPADDING zap =
            new ZasifrujAESPADDING(buffer, symetrickyKlic);

        bufferZasif = zap.zasifruj();

        hlaska += zap.getHlaska() + "\n";

        pocetBajtu = 0;
    }

    fos.write(bufferZasif);
}
```

Při vytváření instance třídy `ZasifrujAESPADDING` se v konstruktoru předá klíč a zbytek bajtů, které mají být zašifrovány. Návrátová hodnota metody `zasifruj` je zašifrované pole bajtů zarovnané paddingem, které se ale zapíše až hlavní části programu.

```
// MOŽNÁ IMPLEMENTACE ČÁSTI 2
public class ZasifrujAESPADDING
{
    byte[] vstup;
    Key klic;
    byte[] zasifrovane;

    String hlaska;

    // KONSTRUKTOR
    public ZasifrujAESPADDING(byte[] vstupByte, Key klicKey)
    {
        vstup = vstupByte;
        klic = klicKey;
        zasifrovane = null;

        hlaska = "";
    }
}
```

```

public byte[] zasifruj()
{
    try
    {
        Cipher cipher =
            Cipher.getInstance("AES/CBC/PKCS5Padding");

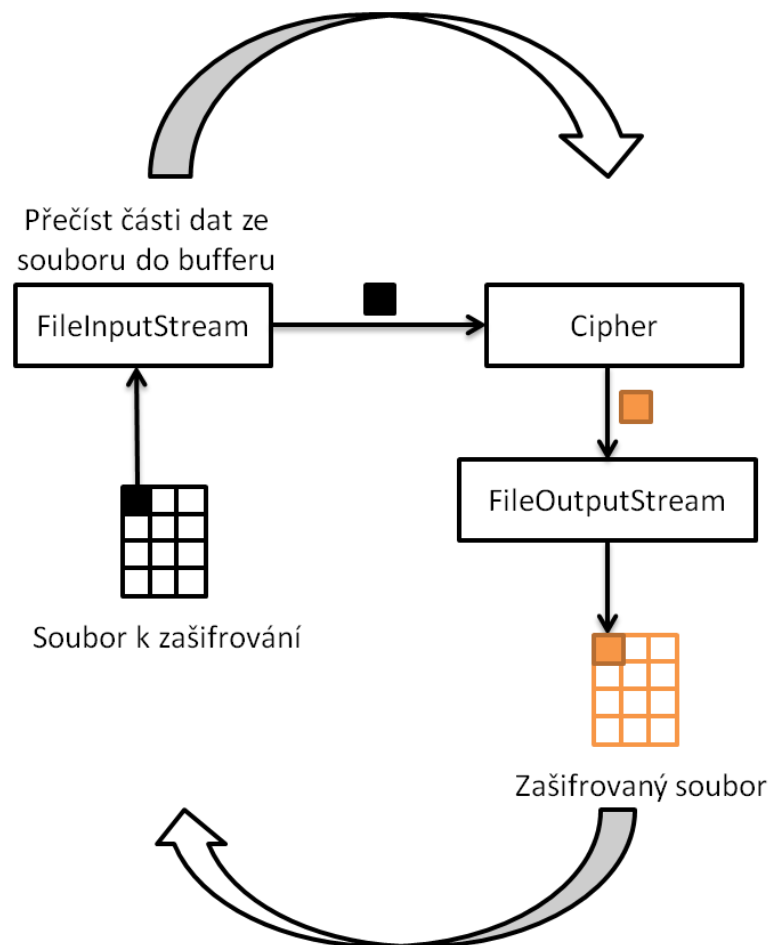
        cipher.init(Cipher.ENCRYPT_MODE, klic);

        zasifrovane = cipher.doFinal(vstup);

        hlaska += "Šifrování - Padding: OK";
    }
    catch (Exception e)
    {
        hlaska += "Chyba paddingu při šifrování: " +
            e.toString();
    }
    return zasifrovane;
}

public String getHlaska()
{
    return hlaska;
}
}

```



Obr. 57 Šifrování pomocí bufferu [zdroj vlastní]

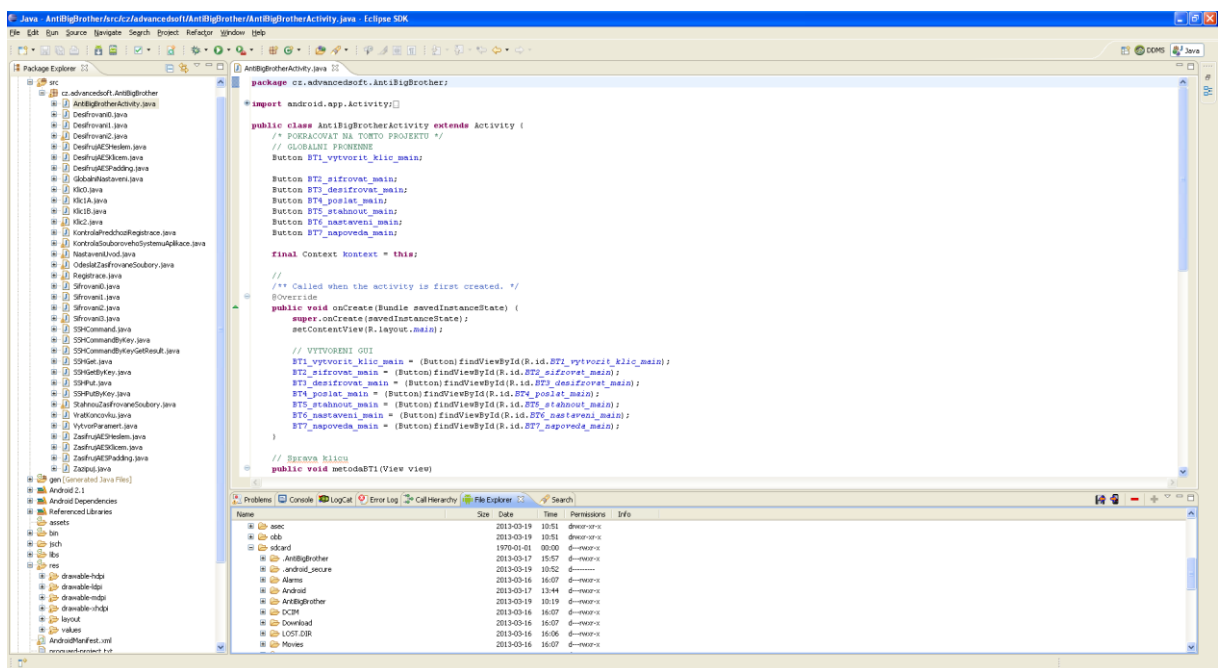
## 4 NÁVRH KRYPTOGRAFICKÉ APLIKACE PRO MOBILNÍ PLATFORMU ANDROID

### 4.1 Nástroje pro vývoj a ukázka vytváření programů

Vývoj profesionálních aplikací se provádí pomocí Android SDK (Software Development Kit) a ADT (Android Development Tools) Plugin [37].

Android SDK poskytuje API knihovny a vývojářské nástroje potřebné k vytvoření, testování a ladění aplikací pro Android [38].

Android ADT je plugin pro vývojové prostředí Eclipse, které integruje nástroje potřebné pro vytváření aplikací běžících pod operačním systémem Android [39]. Na obrázku 58 je znázorněno vývojové prostředí Eclipse s nainstalovaným ADT pluginem.



Obr. 58 Vývojové prostředí Eclipse s nainstalovaným ADT pluginem [zdroj vlastní]

Vytváření mobilních aplikací probíhá na počítačích s operačním systémem Windows, Linux nebo Mac OS X [40].

Vývoj aplikace je rozdělen na dvě části:

1. Aplikační část řeší veškerou funkcionalitu programu, například výpočty, komunikaci se vzdálenými servery atd. Aplikační část je realizovaná pomocí jazyka Java, který je rozšířen o vlastnosti specifické pro operační systém Android. Soubory zdrojových kódů jsou uloženy v adresáři projektu, v podadresáři src.

2. Grafické uživatelské prostředí (GUI) určuje rozmístění a typ ovládacích prvků na obrazovce zařízení. GUI je vytvářeno XML jazykem. Soubory XML kódů jsou uloženy v adresáři projektu, v podadresáři res/layout.

Ukázka XML návrhu grafického uživatelského prostředí. Pro návrh byla použita třída RelativeLayout.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

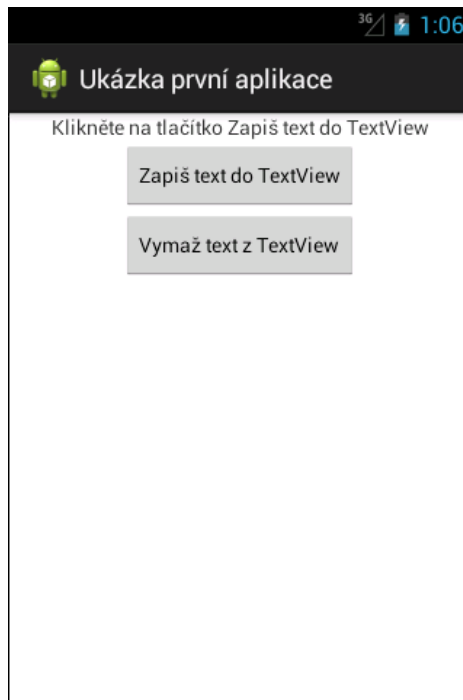
  <TextView
    android:id="@+id/tv1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    tools:context=".MainActivity" />

  <Button
    android:id="@+id/tlacitko1"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_below="@+id/tv1"
    android:onClick="metodaZapis"
    android:text="Zapiš text do TextView" />

  <Button
    android:id="@+id/tlacitko2"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_below="@+id/tlacitko1"
    android:onClick="metodaSmaz"
    android:text="Vymaž text z TextView"
    android:layout_alignLeft="@+id/tlacitko1"
    android:layout_alignRight="@+id/tlacitko1" />

</RelativeLayout>
```

Pomocí XML byl vytvořen návrh obrazovky, který má jedno textové pole a pod ním dvě tlačítka. Všechny ovládací prvky jsou vystředěny na střed obrazovky. Tlačítka zabírají jen tolik místa, kolik je potřeba na zobrazení textu, přičemž tlacitko2 je zarovnáno podle tlacitko1 viz obrázek 59.



*Obr. 59 XML návrh grafického uživatelského prostředí [zdroj vlastní]*

#### Ukázka aplikační části programu

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity
{
    // GUI
    TextView tv1;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // GUI
        tv1 = (TextView) findViewById(R.id.tv1);
        tv1.setText("Klikněte na tlačítko Zapiš text do TextView");
        this.setTitle("Ukázka první aplikace");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

```
public void metodaZapis(View view)
{
    tv1.setText("Ahoj světe!");
}

public void metodaSmaz(View view)
{
    tv1.setText("");
}
}
```

Aby bylo možné ovládacím prvkům definovaných v XML souboru přiřadit nějakou funkcionalitu, je potřeba jako první vytvořit instanci třídy daného ovládacího prvku. V ukázce je to textové pole, do kterého se nastavuje text.

```
TextView tv1;
```

V dalším kroku je propojena instance ovládacího prvku s odpovídajícím prvkem nadefinovaným v XML návrhu.

```
tv1 = (TextView) findViewById(R.id.tv1);
```

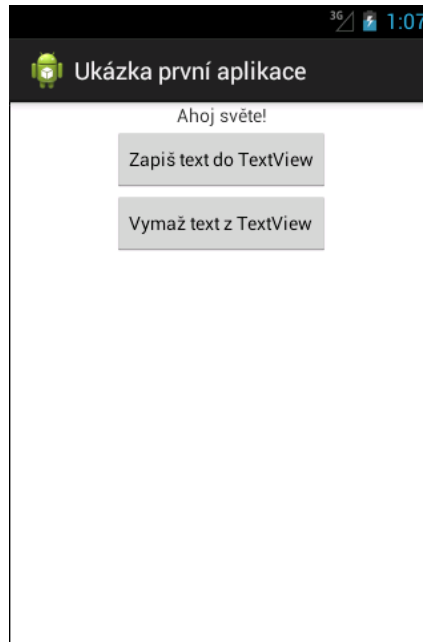
Nyní je možné ovládací prvek používat v programu a přidávat funkcionalitu. Jedním z hlavních zdrojů funkcionality je zpracování události kliknutí na tlačítko. Jako první je potřeba definovat obsluhu události konkrétního prvku v XML návrhu.

```
android:onClick="metodaZapis"
```

Po kliknutí na tlačítko se hledá v Java kódu obsluha události nazvaná *metodaZapis*. Pokud bude nalezena, tak se vykoná, pokud nebude nalezena, aplikace bude násilně ukončena. Obsluha události musí být veřejná a musí obsahovat parametr View.

```
public void metodaZapis(View view)
```

V metodě *metodaZapis* je jediný řádek, který způsobí, že po kliknutí na tlačítko "Zapiš text do TextView" se do textového pole *tv1* zapíše řetězec "Ahoj světe!".

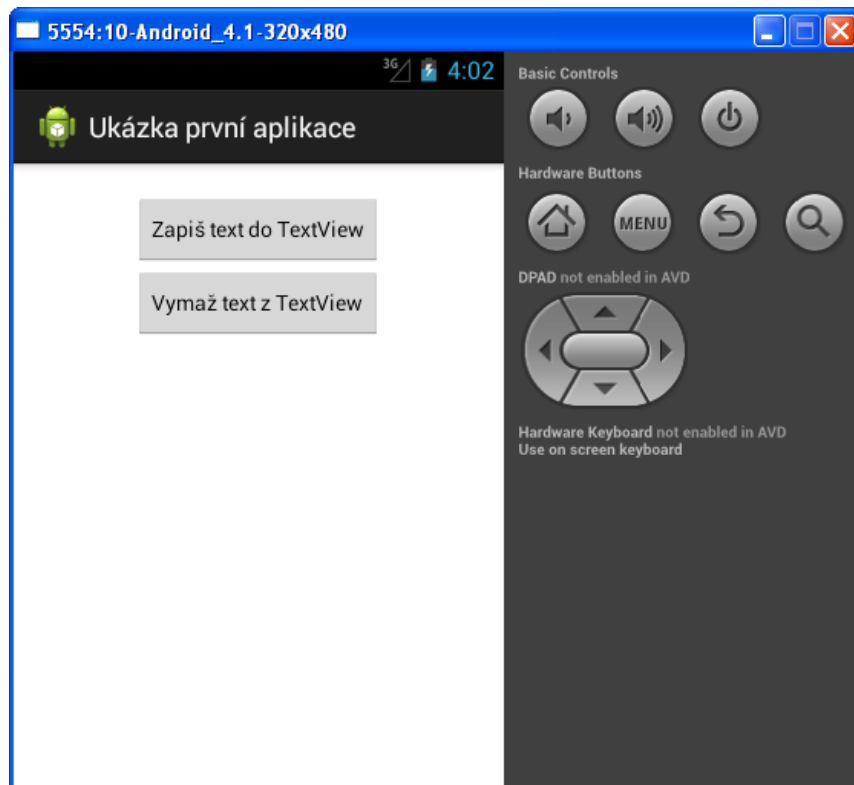


*Obr. 60 Obrazovka aplikace po kliknutí na tlačítko [zdroj vlastní]*

## **4.2 Spuštění a testování vyvíjených aplikací v zařízeních s operačním systémem Android**

Zatím co vývoj mobilních aplikací probíhá na počítačích s operačním systémem Windows, Linux nebo Mac OS X, hotové nebo částečně dokončené aplikace lze spustit pouze v zařízeních s operačním systémem Android. Spouštění mobilních aplikací během vývoje se provádí dvěma způsoby.

V emulátoru telefonu či tabletu s operačním systémem Android, který je vytvořen a spouštěn prostřednictvím Android Virtual Device Manageru. Virtuální telefon s rozlišením 320x480 se spuštěnou ukázkou programu je zachycen na obrázku 61.



*Obr. 61 Zařízení s operačním systémem Android spuštěné prostřednictvím Android Virtual Device Manageru [zdroj vlastní]*

Druhý způsob spuštění je přímo ve fyzickém telefonu. Ladit aplikace ve skutečných zařízeních je vždy nejlepší způsob, protože ne všechny chyby je schopen emulátor zachytit. Ne každá část hardwaru jde emulovat. Například emulace fotoaparátu je velmi problematická. Samostatnou kapitolou jsou nastavby velkých výrobců jako je Samsung nebo HTC nad operačním systémem. Nastavby mohou způsobovat nekorektní chování programů odladěných v emulátoru a to i přes skutečnost, že testované programy s nastavbou nespolupracují. Pro profesionální vývojáře je nezbytné vyzkoušet jejich programy alespoň na jednom fyzickém zařízení firmy Samsung z důvodu jejich světového dominantního postavení na trhu s mobilními telefony a tablety. Například telefon Samsung Galaxy S3 je v době zpracování diplomové práce nejprodávanější telefon s operačním systémem Android na světě [41].



*Obr. 62 Testování aplikace ve skutečném telefonu, který je kabelem připojen k počítači*

Jelikož vyvíjená aplikace běží na zařízení, které je ať už reálně nebo virtuálně mimo vývojářský počítač. Není možné program ladit běžným způsobem, tak jak je běžné při vývoji desktopových aplikací. Proto se pro účely ladění používá tzv. LogCat. LogCat představuje pomyslné okno do telefonu či tabletu jak fyzického tak virtuálního. Pomocí LogCat lze zjistit mnoho důležitých údajů. LogCat se používá dvojím způsobem.

Prvním způsobem je sledování systémových zpráv dané úrovně výpisu. Úrovně výpisu jsou: verbose, debug, info, warn, error, assert. Například je to velmi užitečné pro odladování neošetřených výjimek. V ukázce aplikačního kódu byla zakomentována obsluha události tlačítka "Zapiš text do TextView". Po kliknutí na tlačítko není

obsluha události nalezena a aplikace je ukončena. Zprávy, ze kterých lze vyčíst, co se v telefonu stalo, jsou zachyceny na obrázku 63.

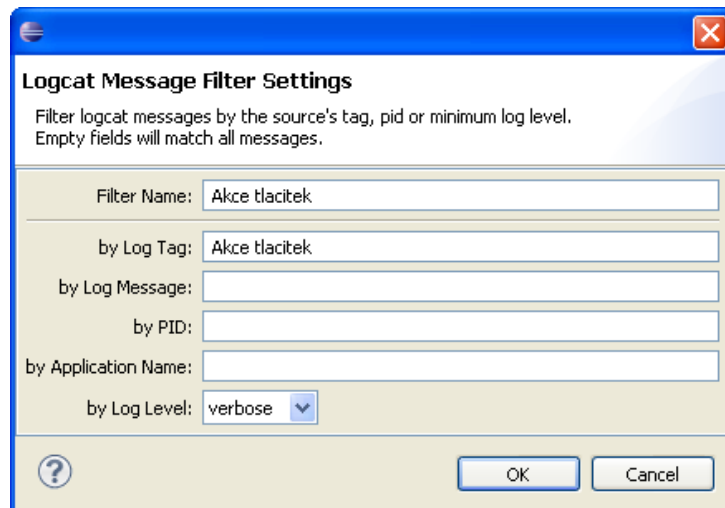
PID	TID	Application	Tag	Text
25374	25374	cs.advancerdsoft....	AndroidRuntime	FATAL EXCEPTION: main
25374	25374	cs.advancerdsoft....	AndroidRuntime	java.lang.IllegalStateException: Could not find a method metodaZapis(View) in the activity class cz.advan D cerdsoft.ukazkaaplikacediplomka.MainActivity for onClick handler on view class android.widget.Button with D id 'tlacitko1'
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.view.View\$1.onClick(View.java:3578)
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.view.View.performClick(View.java:4084)
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.view.View\$PerformClick.run(View.java:16966)
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.os.Handler.handleCallback(Handler.java:615)
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.os.Handler.dispatchMessage(Handler.java:92)
25374	25374	cs.advancerdsoft....	AndroidRuntime	at android.os.Looper.loop(Looper.java:137)

Obr. 63 Výpis LogCat [zdroj vlastní]

Z výpisu `android.view.View$1.onClick(View.java:3578)` se dozvídáme, že chyba vznikla při události `onClick` instance Třídy `View` (Třída `Button` je potomkem třídy `View`). Z dalšího výpisu `03-19 16:58:55.585: E/AndroidRuntime(25374): java.lang.IllegalStateException: Could not find a method metodaZapis(View) in the activity class cz.advancerdsoft.ukazkaaplikacediplomka.MainActivity for onClick handler on view class android.widget.Button with id 'tlacitko1'` se dozvíme, že nebyla nalezena metoda `metodaZapis`, která má obsluhovat událost `onClick` tlačítka `tlacitko1`. Nyní je k dispozici dostatek údajů pro opravu chyby.

Druhým způsobem je sledování zpráv v LogCat, které si programátor sám vytvoří. Tvorba uživatelských zpráv má následující kroky:

1. Do třídy, která představuje aktivitu na obrazovce, je přidána proměnná typu `private static final String`. Například: `private static final String TAG = "Akce tlacitek";`.
2. Na místě v programu, kde je potřeba zjistit, zda byl kód vykonán nebo hodnotu proměnné atd. se vytvoří zpráva pomocí `Log.v(tag, msg);`. Tag je proměnná vytvořená v předchozím kroku. `msg` je zpráva typu `String`, ve které jsou sledované informace programátorem. Například: Do metody `metodaZapis` se přidá `Log.v(TAG, "bylo kliknuto na tlacitko1");` a do metody `metodaSmaz` se přidá `Log.v(TAG, "bylo kliknuto na tlacitko2");`.
3. V LogCat se vytvoří filtr, jak je vidět na obrázku 64.



Obr. 64 Filtr LogCat [zdroj vlastní]

Pokud je program v telefonu spuštěn a bylo kliknuto na obě tlačítka, obě metody byly provedeny. Každá z nich vytvořila zprávu, které jsou vidět na obrázku 65.

Level	Time	PID	TID	Application	Tag	Text
V	03-19 15:54:48.177	23305	23305	cz.advancerdsoft....	Akce tlacitek	bylo kliknuto na tlacitko1
V	03-19 15:54:51.472	23305	23305	cz.advancerdsoft....	Akce tlacitek	bylo kliknuto na tlacitko2

Obr. 65 Zprávy v LogCat vytvořené programátorem [zdroj vlastní]

## 4.3 Grafický návrh aplikace

### 4.3.1 Způsoby vytváření GUI a jejich problémy

Při vytváření GUI může být k velikosti a poloze ovládacích prvků přístupováno několika způsoby:

- pomocí třídy `LinearLayout`;
- pomocí třídy `RelativeLayout`;
- pomocí třídy `TableLayout`.

#### Problémy spojené s návrhem pomocí třídy `LinearLayout`

Třída `LinearLayout` reprezentuje model založený na používání boxů, ve kterém se widgety (ovládací prvky) nebo kontejnery, které jsou potomky kořenového kontejneru, řadí do sloupce nebo řádku jeden po druhém [42]. Výška a šířka ovládacího prvku nebo

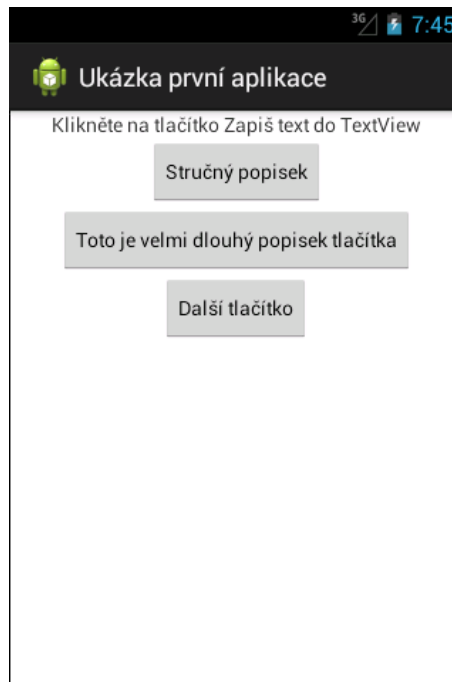
kontejneru sdružujícího množinu ovládacích prvků je určována parametrem `fill_parent` nebo `wrap_content`.

`fill_parent` – ovládací prvek vyplní celou výšku (`android:layout_height="fill_parent"`) nebo šířku (`android:layout_width="fill_parent"`), kterou zabírá nadřazený prvek (například kořenový kontejner), ve kterém nastavovaný prvek leží. Takový přístup je vhodný pro telefony s úhlopříčkou do 3,5 palce. Představme si tlačítka zabírající celou šířku pětipalcového displeje telefonu nebo dokonce šířku deseti palcové obrazovky tabletu. Nepůsobilo by to hezkým dojmem. V případě otočení zařízení uživatelem naležato by uživatelské rozhraní nebylo hezké ani ergonomické. Výsledek je vidět na obrázku 66.



Obr. 66 Aplikace otočená naležato [zdroj vlastní]

`wrap_content` - ovládací prvek vyplní jen takovou výšku (`android:layout_height="wrap_content"`) nebo takovou šířku (`android:layout_width="wrap_content"`), kterou skutečně zabírá. Výsledek působí neuspořádaným dojmem, jak je vidět na obrázku 67.

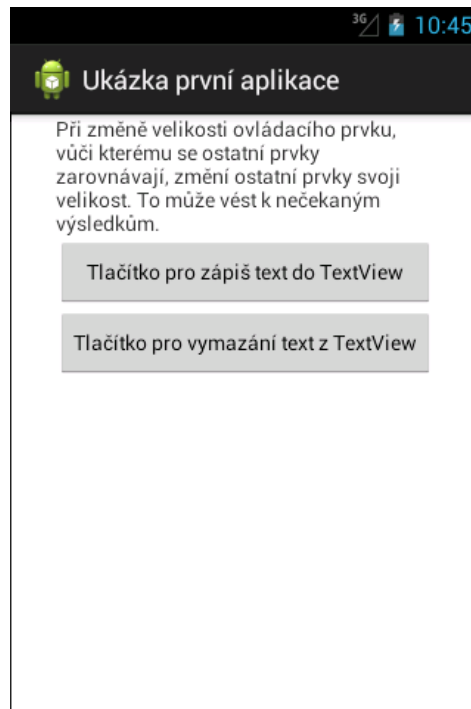


*Obr. 67 Ovládací prvky vyplní jen takovou výšku, kterou zabírají [zdroj vlastní]*

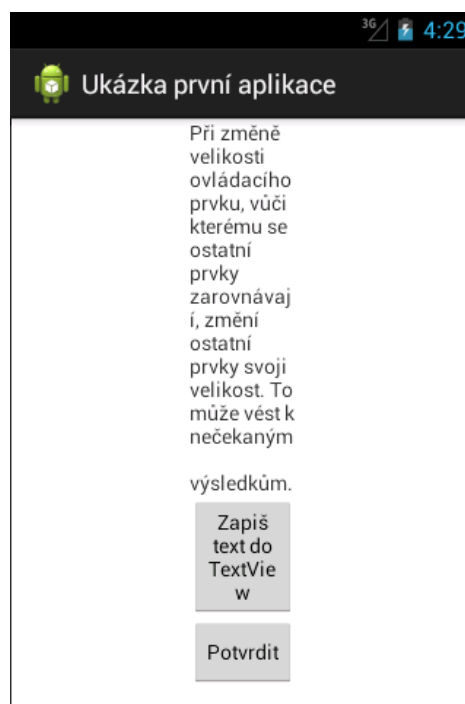
### **Problémy spojené s návrhem pomocí třídy RelativeLayout**

Třída RelativeLayout reprezentuje model založený na vzájemných vztazích mezi jednotlivými widgety v kontejneru a na jejich vztahu k rodičovskému kontejneru. Widget X můžeme umístit nalevo pod widget Y, zarovnat spodní okraj widgetu Z na spodní okraj kontejneru atd [42].

Úskalí uvedeného řešení – například při změně velikosti ovládacího prvku, vůči kterému se ostatní prvky zarovnávají, změni ostatní prvky svoji velikost. To může vést k nečekaným výsledkům. Na obrázku 68 je vidět zarovnání všech ovládacích prvků podle druhého tlačítka. Zdá se, že je grafický návrh pořádku. Ovšem jen do doby než se změni popiska tlačítka dvě. Výsledek je zachycen na obrázku 69.



*Obr. 68 Návrh vytvořený pomocí třídy RelativeLayout [zdroj vlastní]*



*Obr. 69 Návrh vytvořený pomocí třídy RelativeLayout při změně klíčového ovládacího prvku [zdroj vlastní]*

### Problémy spojené s návrhem pomocí třídy TableLayout

Navrhování pomocí třídy TableLayout je podobné vytváření tabulek v jazyce HTML. Mají i podobné problémy. Z toho vyplývá, že návrhy založené na třídě TableLayout jsou vhodné, pokud jsou ovládací prvky návrhu podobně velké, viz obrázek 70.

Ovládací prvek 1	Ovládací prvek 2
Ovládací prvek 3	Ovládací prvek 4
Ovládací prvek 5	Ovládací prvek 6

*Obr. 70 Návrh, ve kterém jsou ovládací prvky návrhu podobně velké [zdroj vlastní]*

Jestliže jsou ovládací prvky návrhu rozdílných velikostí, dojde k problémům známých i z HTML tabulek. Velikost řádku tabulky je dána obsahem její největší buňky a to samé platí i pro sloupce. Výsledný návrh působí chaotickým dojmem. Na obrázku 71 je vidět, jak jediný větší prvek než ostatní může zničit jinak povedený návrh.

Ovládací prvek 1		Ovládací prvek 2
Ovládací prvek 3		
		Ovládací prvek 4
Ovládací prvek 5		Ovládací prvek 6

*Obr. 71 Návrh, ve kterém je jeden prvek větší než ostatní [zdroj vlastní]*

Další možností je zadávání velikostí v programovém kódu aplikace. Nejedná se o standardní postup. Měl by být využíván jen v krajním případě.

Návrh GUI aplikace lze vytvářet i pomocí grafického návrháře. I přes intenzivní snahu společnosti Google v oblasti vylepšení grafického návrháře je stále tento způsob návrhu velmi nedokonalý a zatím nemá takové možnosti jako ručně vytvořený návrh pomocí jazyka XML.

### 4.3.2 Návrh GUI kryptografické aplikace

Z výše uvedených důvodů je jasné, že není jednoduché vytvořit grafický návrh aplikace, který by byl pěkný a zároveň ergonomický na všech typech zařízení s operačním systémem Android. Co je hezké na telefonu s úhlopříčkou obrazovky 3,5 palce, nemusí být pěkné na tabletu s úhlopříčkou 10,1 palce a naopak. V diplomové práci je vytvořen návrh snažící se o jednotný vzhled GUI aplikací použitelný pro všechny velikosti obrazovek. Autor práce jej nazval streak style. Jedná se o autorův originální přístup k řešené problematice a přínos pro teorii a praxi v této oblasti.

Ukázka XML návrhu:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/SV_main"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:fillViewport="true"
    android:background="#3d3b3b">

    <LinearLayout android:id="@+id/LiLa_main"
        android:orientation="vertical"
        android:layout_gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#3d3b3b">

        <RelativeLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_gravity="center_horizontal"
            android:background="#000000">

            <ImageView
                android:layout_centerHorizontal="true"
                android:id="@+id/hlavni_logo"
                android:layout_height="wrap_content"
                android:layout_width="250dp"
                android:adjustViewBounds="true"
                android:src="@drawable/hlavni_logo"
                android:padding="3dp"
                android:contentDescription="@string/IV_main" />

            <Button
                android:id="@+id/BT1_vytvorit_klic_main"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_alignLeft="@+id/hlavni_logo"
                android:layout_alignRight="@+id/hlavni_logo"
                android:layout_below="@+id/hlavni_logo"
                android:onClick="metodaBT1"
                android:text="@string/BT1_sprava_klicu_main"
                android:drawableTop="@drawable/sprava_klicu"
                android:contentDescription=
```

```

                                "@string/BT1_sprava_klicu_main"
                                android:background=
                                "@drawable/ramecek_cerny_a_sedy_podklad"
                                android:padding="9dp"
                                android:textColor="#000000"/>

                                <Button
                                android:id="@+id/BT2_sifrovat_main"
                                android:layout_width="fill_parent"
                                android:layout_height="wrap_content"
                                android:layout_alignLeft="@+id/hlavni_logo"
                                android:layout_alignRight="@+id/hlavni_logo"
                                android:layout_below="@id/BT1_vytvorit_klic_main"
                                android:onClick="metodaBT2"
                                android:text="@string/BT2_sifrovat_main"
                                android:drawableTop="@drawable/zasifrovat"
                                android:contentDescription=
                                "@string/BT2_sifrovat_main"
                                android:background=
                                "@drawable/ramecek_cerny_a_sedy_podklad"
                                android:padding="9dp"
                                android:textColor="#000000" />
                                ...
                                ...
                                ...

                                </RelativeLayout>

                                </LinearLayout>

                                </ScrollView>

```

Streak style si nevystačí s jedinou layout třídou. Je kombinací tříd LinearLayout a RelativeLayout a techniky pevného grafického prvku. Najít uvedenou techniku bylo velmi zdlouhavé a vyžadovalo mnoho experimentování. Základem je neměnný grafický prvek. Nejlépe se osvědčil obrázek ve formátu portable network graphics se šířkou 250 pixelů. Nestačí vytvořit obrázek o šířce 250 pixelů, protože různé verze operačního systému Android na různých velikých obrazovkách provádějí přepočty, aby byla grafika co nejlepší. Z uvedeného důvodu je nutné zadat parametr šířky pevného grafického prvku:

```
android:layout_width="250dp"
```

Tím je zajištěno, že bude šířka obrázku správně zobrazena i na nejmenší obrazovce dostupné na trhu. Tou je obrazovka 2,5 palce s orientací na výšku. Ostatní prvky, kterými jsou v tomto případě tlačítka, se budou zarovnávat zleva a zprava podle pevného grafického prvku pomocí:

```
android:layout_alignLeft="@+id/hlavni_logo"
```

```
android:layout_alignRight="@+id/hlavni_logo"
```

Pro správné fungování je potřeba u každého zarovnávaného prvku nastavit:

```
android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content"
```

Po tomto nastavení se budou všechny ovládací prvky správně zarovnávat podle pevného grafického prvku.

Všechny ovládací prvky jsou obaleny třídou RelativeLayout, která má nastavenou gravitaci na střed:

```
android:gravity="center"
```

```
android:layout_gravity="center_horizontal"
```

Tím je zajištěno, že budou všechny ovládací prvky, které třída obaluje krásně vystředěné. Pro správnou funkci je potřeba třídě RelativeLayout sdělit, aby zabírala jen tolik místa, kolik potřebuje:

```
android:layout_width="wrap_content"
```

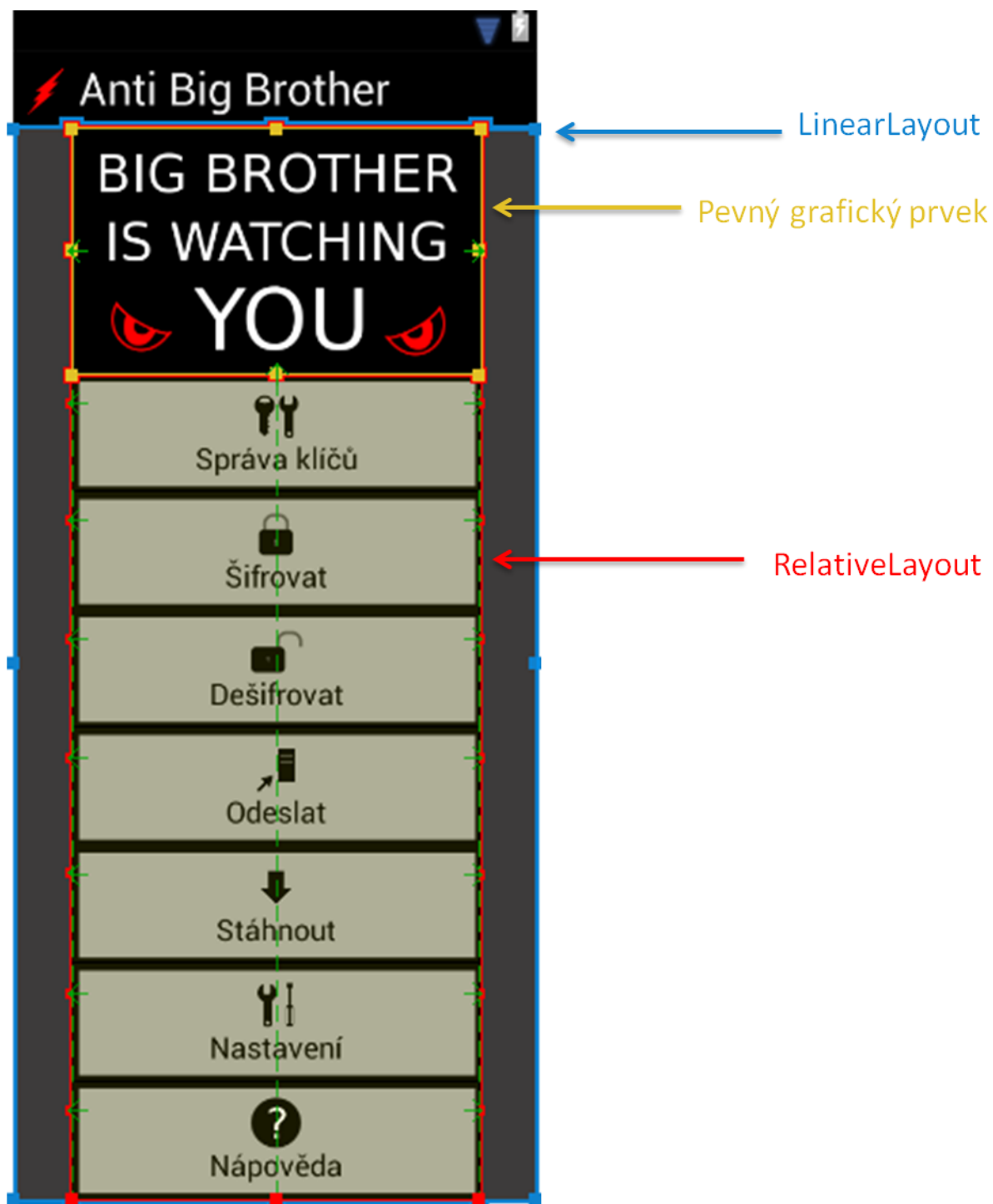
```
android:layout_height="wrap_content"
```

Díky třídě RelativeLayout je možné, aby se prvky umísťovaly pod sebe. Například tímto způsobem je tlačítko sděleno, že se má umístit přímo pod pevný grafický prvek:

```
android:layout_below="@+id/hlavni_logo"
```

Třída RelativeLayout je obalena třídou LinearLayout. Zajišťuje správné zobrazení streak style i na starších Androidech.

Nakonec je celý XML kód obalen třídou ScrollView. Třída zajistí, že pokud se aplikace nevejde celá najednou na obrazovku, bude možné aplikaci po částech po obrazovce posouvat a zobrazovat.



Obr. 72 Praktická ukázka streak style [zdroj vlastní]

## 4.4 Vývoj komponenty pro grafickou práci se soubory

### 4.4.1 Aktuální situace

Filozofie firmy Google vyvíjející a udržující operační systém Android se zakládá na představě, že uživatel by neměl mít svá data uložena lokálně ve svém zařízení. Měl by je mít uloženy v síťových službách Google na serverech společnosti. To je důvod, proč v operačním systému Android neexistují standardizované grafické komponenty pro výběr souborů a adresářů. Což je velký rozdíl oproti jazykům jako je například C#, kde má programátor k dispozici komponenty OpenFileDialog, SaveFileDialog, FolderBrowserDialog a další. Jestliže chce vývojář Android aplikací danou funkcionalitu uživatelům ve svém programu nabídnout, musí ji sám vytvořit.

Pro vývoj komponenty je možné použít třídu File a její metody, zejména:

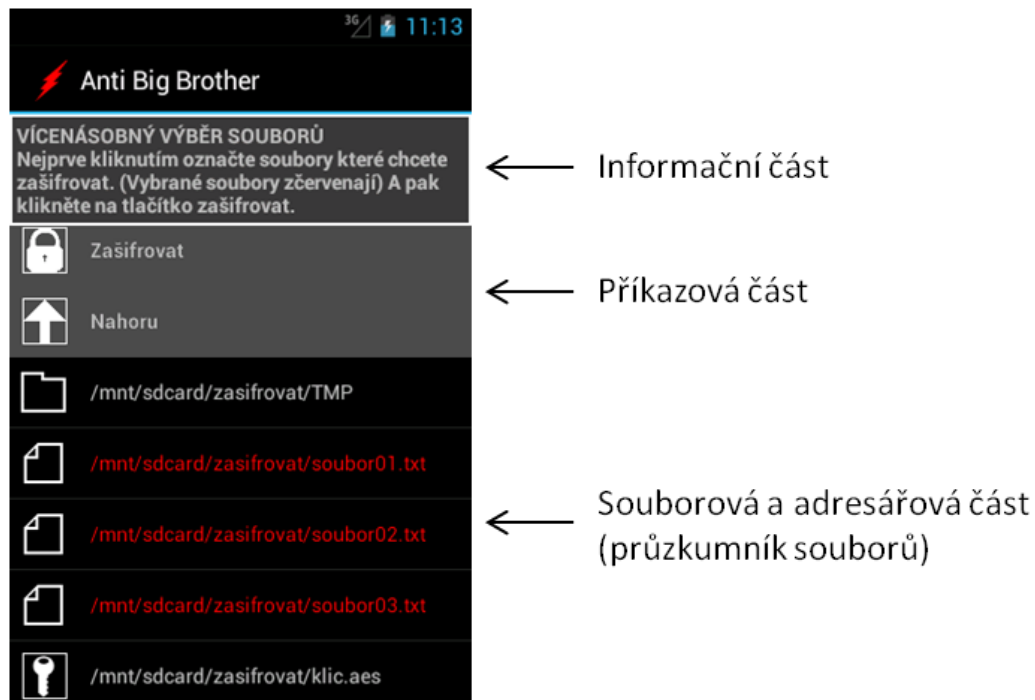
- delete();
- exists();
- getAbsolutePath();
- getName();
- getParent();
- getPath();
- isDirectory();
- isFile();
- listFiles();
- mkdir();

### 4.4.2 Návrh komponenty pro grafickou práci se soubory a adresáři

Univerzální návrh komponenty pro grafickou práci se soubory a adresáři je tvořen třemi částmi:

- Informační část - je realizovaná pomocí TextView.
- Příkazová část - je realizovaná pomocí položek ListView nebo pomocí Button.
- Souborová část (průzkumník souborů) - je realizována pomocí položek ListView.

Komponenta pro grafickou práci se soubory a adresáři je zachycena na obrázku 73.



Obr. 73 Komponenta pro grafickou práci se soubory a adresáři [zdroj vlastní]

Nahrazením implementací události onClick příkazových položek a změnou jejich počtu lze dosáhnout libovolné funkcionality potřebné pro práci se soubory a adresáři.

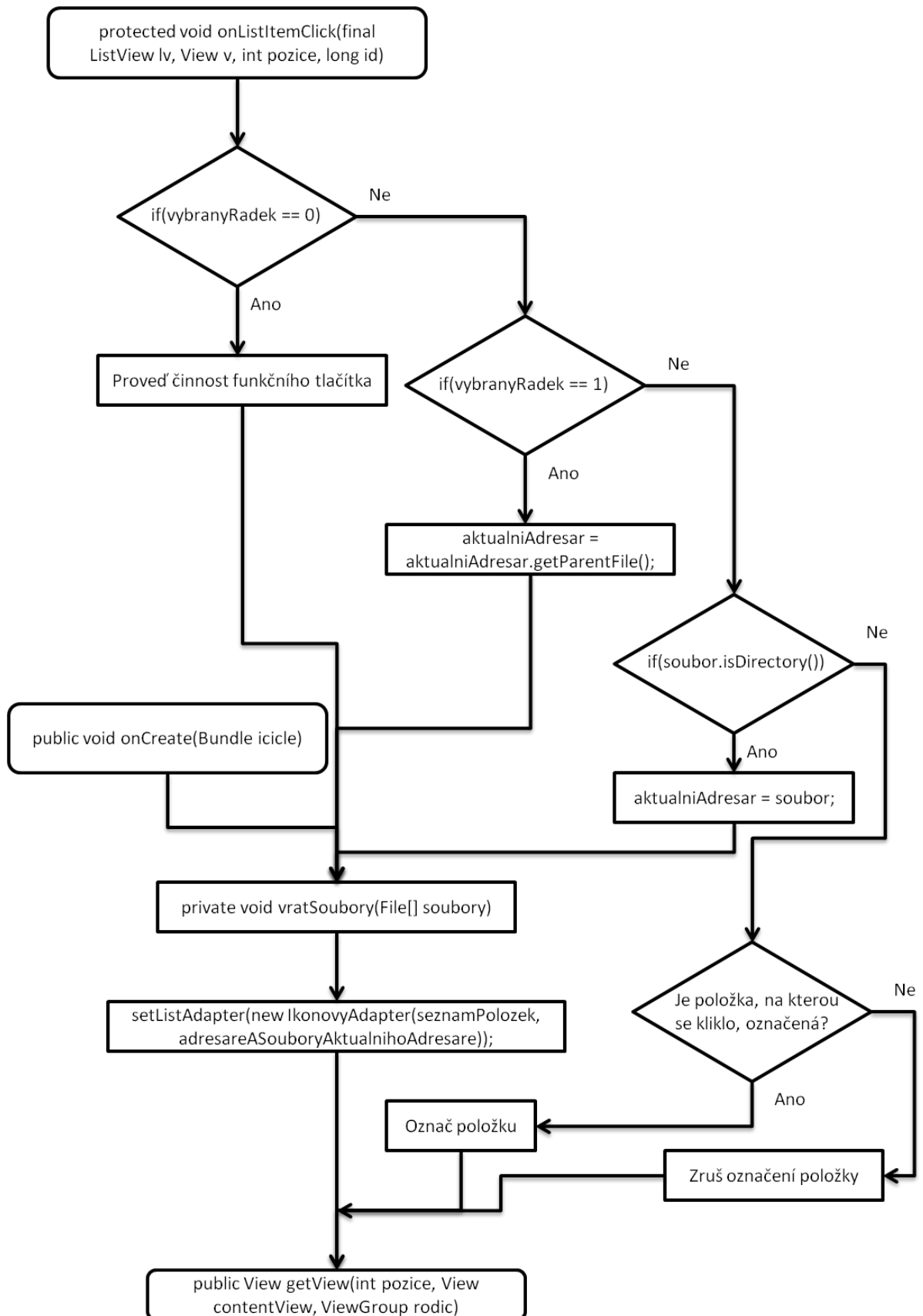
Při prvním spuštění aktivity představující obrazovku pracující s komponentou je vytvořena:

- proměnná `aktualniAdresar` typu `File`. Představuje místo v souborovém systému, kde se právě uživatel nachází. Výchozí hodnotou je kořenový adresář lokálního úložiště.
- seznam `seznamPolozek`, jehož položky budou typu `String`, do něhož je ukládán seznam všech souborů a adresářů aktuálního adresáře včetně absolutní cesty.
- seznam `adresareASouboryAktualnihoAdresare`, jehož položky budou typu `String`, do něhož je ukládán záznam. Záznam říká, zda je odpovídající položka v seznamu `seznamPolozek` adresářem nebo souborem. V případě souboru lze pořídit záznam o typu souboru, na jehož základě se bude souboru přiřazovat obrázek/ikona v ovládacím prvku `ListView`.

Samotné spuštění nebo aktualizace komponenty je realizováno trojím způsobem:

1. Zavoláním metody `public void onCreate(Bundle savedInstanceState)`.
2. Kliknutím uživatele na vybranou příkazovou položku.

3. Jakoukoliv akci vyvolávající změnu zobrazení. Akce způsobí přímé zavolání metody `public View getView(int pozice, View contentView, ViewGroup rodic)`.  
Například uživatel prstem posouvá aplikaci po obrazovce.



Obr. 74 Spuštění nebo aktualizace komponenty [zdroj vlastní]

## Způsob zobrazení souborů a adresářů I

### 1, Metoda public void onCreate(Bundle savedInstanceState)

V metodě public void onCreate(Bundle savedInstanceState) patří třídě aktivity, která komponentu využívá, se nastaví jako aktuální adresář kořenový adresář lokálního úložiště:

```
aktualniAdresar =  
  
    new  
        File(Environment.getExternalStorageDirectory().getPath());
```

V dalším kroku je zavolána metoda vratSoubory. Za její parametr je použito pole typu File, do kterého jsou uloženy adresáře a soubory kořenového adresáře lokálního úložiště:

```
vratSoubory(aktualniAdresar.listFiles());
```

### 2, Metoda private void vratSoubory(File[] soubory)

Na první (index 0) až  $n$ -té místo seznamu seznamPolozek se umístí popisky funkcionalit, které má komponenta se soubory a adresáři vykonat. Funkcionalit může být 1 až  $n$ , například zašifrovat:

```
seznamPolozek.add(getString(R.string.zasifrovat));
```

Na  $n + 1$  místo seznamu seznamPolozek (index  $n$ ) se umístí popiska nahoru zajišťující pohyb o úroveň výš v adresářové struktuře lokálního úložiště:

```
seznamPolozek.add(getString(R.string.nahoru));
```

Na první (index 0) až  $n$ -té místo seznamu adresareASouboryAktualnihoAdresare se umístí popisky funkcionalit, které má komponenta se soubory a adresáři vykonat. Funkcionalit může být 1 až  $n$ , například zašifrovat:

```
adresareASouboryAktualnihoAdresare.add(getString(R.string.zasifrovat));
```

Na  $n + 1$  místo seznamu adresareASouboryAktualnihoAdresare (index  $n$ ) se umístí popiska nahoru zajišťující pohyb o úroveň výš v adresářové struktuře lokálního úložiště:

```
adresareASouboryAktualnihoAdresare.add(getString(R.string.nahoru));
```

Postupně jsou procházeny všechny soubory pole soubory. Pokud je procházený prvek adresář, je do seznamu seznamPolozek adresareASouboryAktualnihoAdresare přidán řetězec "adresar". Pokud je procházený prvek souborem, zjistí se jeho koncovka. Je-li

koncovka rovna typu souboru, který chce programátor zobrazit samostatnou ikonou, pak se vloží do pole adresareASouboryAktualnihoAdresare odpovídající řetězec. Například:

```
if(koncovka.equals(".aes") || koncovka.equals(".pub"))
{
    adresareASouboryAktualnihoAdresare.add("klic");
}
```

Všem ostatním typům souborů, které programátor nechce zobrazovat samostatnou ikonou, se přiřadí obecný řetězec "soubor" na základě, kterého budou zobrazeny jako obecný soubor:

```
adresareASouboryAktualnihoAdresare.add("soubor");
```

Nakonec metoda vratSoubory zavolá metodu setListAdapter, která jako parametr předá nově vytvořenou instanci třídy IkonovyAdapter:

```
setListAdapter(new IkonovyAdapter(seznamPolozek,
adresareASouboryAktualnihoAdresare));
```

### 3, Třída class IkonovyAdapter extends ArrayAdapter<String>

Třída je potomkem třídy ArrayAdapter. Aby bylo možné vytvářet uživatelské podoby ovládacího prvku ListView s obrázky, musí být přepsána metoda getView. To je důvodem, proč byla třída IkonovyAdapter vytvořena.

V metodě public View getView(int pozice, View contentView, ViewGroup rodic) se vytvoří uživatelsky definované řádky podle obsahu seznamů seznamPolozek a adresareASouboryAktualnihoAdresare. Podle seznamPolozek se vytvoří textový popis se jménem a absolutní cestou k souboru či adresáři. Pole soubory je vytvořeno ze seznamu seznamPolozek.

```
TextView TV_radek_sezmanu_souboru =
(TextView) radek.findViewById(R.id.TV_radek_sezmanu_souboru);
TV_radek_sezmanu_souboru.setText(soubory[pozice].toString());
```

Podle adresareASouboryAktualnihoAdresare jsou vytvářeny ikony/obrázky adresářů, souborů známých typů a obecných typů. Například:

```
if(obsahAktualnihoAdresare[pozice] == "adresar")
{
    if(TV_radek_sezmanu_souboru.getCurrentTextColor() == -65536)
    {
        TV_radek_sezmanu_souboru.setTextColor(-4276546);
    }

    TV_radek_sezmanu_souboru.setTypeface(null, Typeface.NORMAL);
}
```

```

        TV_radek_sezmanu_souboru.setCompoundDrawablesWithIntrinsicBounds(
R.drawable.klic_2_adresar, 0, 0, 0 );
    }

    ...

```

Takto vyrobený řádek je vrácen jako výsledek volání metody getView.

## Způsob zobrazení souborů a adresářů II

Pokud uživatel klikne na příkazovou nebo souborovou položku ListView, je zavolána přeepsaná metoda onItemClick. Metoda patří třídě představující aktivitu obrazovky a je zděděná ze třídy ListActivity.

```

@Override
protected void onItemClick(final ListView lv, View v, int pozice,
long id)

```

Proměnná vybranyRadek představuje řádek, na který uživatel klikl. Pokud je vybraný řádek na pozici 0 až  $n - 1$ , kde  $n$  je počet příkazových nabídek. Jedná se o příkazovou položku a vykoná se požadovaná funkcionality. Zde je dobré rozdělit hlavní vlákno na vlákno, které bude vykonávat se soubory danou funkcionality, a na vlákno, které bude zobrazovat animovaný ProgressDialog. To je důležité zejména, v případě že se vykonává časově náročnější operace jako například hromadné šifrování souborů. Bez ProgressDialogu se uživateli jeví aplikace jako zamrzlá. Protože aplikace provádí se soubory zvolenou činností a nereaguje na další podmínky od uživatele, dokud není daná činnost dokončena. Změní-li funkcionality soubory nebo adresáře v aktuálně prohlíženém adresáři, je nutné po spojení obou vláken do hlavního vlákna opět vybudovat zaktualizovaný pohled na aktuální adresář. Provádí se voláním metody vratSoubory.

```

if(vybranyRadek == 0)
{
    final ProgressDialog dialog = ProgressDialog.show(context, "Čekejte
prosím", "Šifruji ...\nTo může v závislosti na výkonu procesoru vašeho
zařízení trvat i několik minut!");

    Thread backgroundThread = new Thread(new Runnable()
    {
        @Override
        public void run()
        {
            for(int i = 0; i < seznamPolozek.size(); i++)
            {
                final File soubor =
                newFile(seznamPolozek.get(i));

                // PROVED ZE SOUBOREM POZADOVANOU FUNKCIONALITU
                . . .
            }
        }
    });
}

```

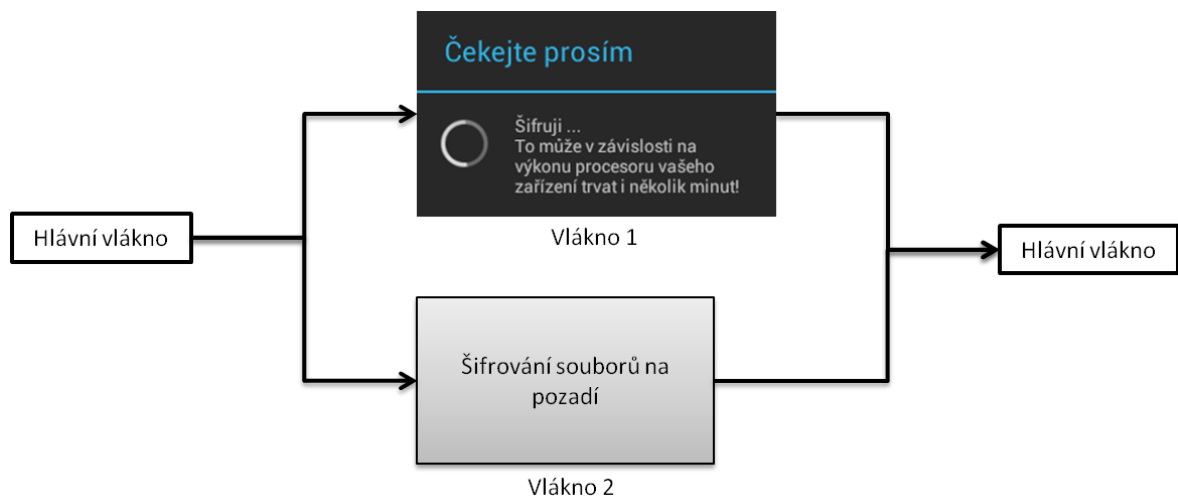
```

        . . .
        . . .
    }

    runOnUiThread(new Runnable()
    {
        @Override
        public void run()
        {
            dialog.dismiss();
            // ZDE MUŽE BYT UPDATE GRAFIKY
            vratSoubory(aktualniAdresar.listFiles());
            . . .
            . . .
            . . .
        }
    });
}
});
backgroundThread.start();
}

. . .
. . .
. . .

```



Obr. 75 Rozdělení vlákna během provádění požadované funkcionality [zdroj vlastní]

Je-li vybraný řádek na pozici  $n$ , kde  $n$  je počet příkazových nabídek. Jedná se o příkazovou položku zajišťující přechod o úroveň výš v adresářové struktuře. Příkaz se provede pouze v případě, že se uživatel pohybuje v lokálním úložišti. Pokud by se chtěl uživatel dostat do kořenového adresáře souborového systému, nebude mu to umožněno. Důvodem je předejití problémům. Například: uživatel si vytvořil klíč do dočasného adresáře, který se vytváří při startu operačního systému a maže při jeho ukončování. Tímto klíčem si zašifroval svá data. Po restartování telefonu se uživatel ke svým datům již nikdy nedostane. Nebo pokud má své zařízení rootnuté (tj. uživatel získal plná oprávnění „super

uživatel systému“). U většiny výrobců s rootem telefonu/tabletu se ztrácí záruka na dané zařízení. Přesto je rootování zařízení velmi populární a dokonce vznikla celá třída rooted aplikací. Aplikace využívají toho, že uživatel má plná práva super uživatele. Uživatel by si mohl zašifrovat nějaké „zbytečné“ soubory třeba v adresáři /etc. Uvedená situace může vést k vážnému poškození operačního systému, proto při pokusu vstoupit do kořenového adresáře by měl být uživatel nejen odmítnut, ale i o odmítnutí informován. Například:

```

if(vybranyRadek == n)
{
    if(aktualniAdresar.getParent().toString().equals("/"))
    {
        new AlertDialog.Builder(this)
        .setTitle("Pozor !")
        .setMessage("Soubory klíčů mohou být ukládány pouze na
lokálním úložišti !!!\nProto je vybírejte jen v adresářích nebo v
podadresářích SD karty.")
        .setNeutralButton("OK", new DialogInterface.OnClickListener()
        {
            public void onClick(DialogInterface di, int button)
            {
                // NEDELAT NIC, pouze uzivatele informovat
            }
        })
        .show();
    }
    else
    {
        vratSoubory(aktualniAdresar.getParentFile().listFiles());
        aktualniAdresar = aktualniAdresar.getParentFile();
        . . .
        . . .
        . . .
    }
}

```

Pokud je vybraný řádek na pozici  $n + 1$  až  $n + j + 1$ , kde  $j$  je počet souborů a adresářů v aktuálním adresáři, jedná se o souborovou nebo adresářovou položku. Je-li soubor na vybraném řádku adresářem, pak bude nastaven jako nový aktuální adresář. Nově vybudovaný pohled pomocí volání metody vratSoubory bude ukazovat soubory a adresáře v tomto vybraném adresáři. To lze provést například:

```

vratSoubory(soubor.listFiles());
aktualniAdresar = soubor;

```

Je-li soubor na vybraném řádku skutečně soubor, pak bude označen nebo bude zrušeno jeho označení.

Upozornění: Je důležité si uvědomit, že pokud je nějaká část položek ListView označena a odscrollována z obrazovky, systém zruší jejich označení, tj. nastaví výchozí

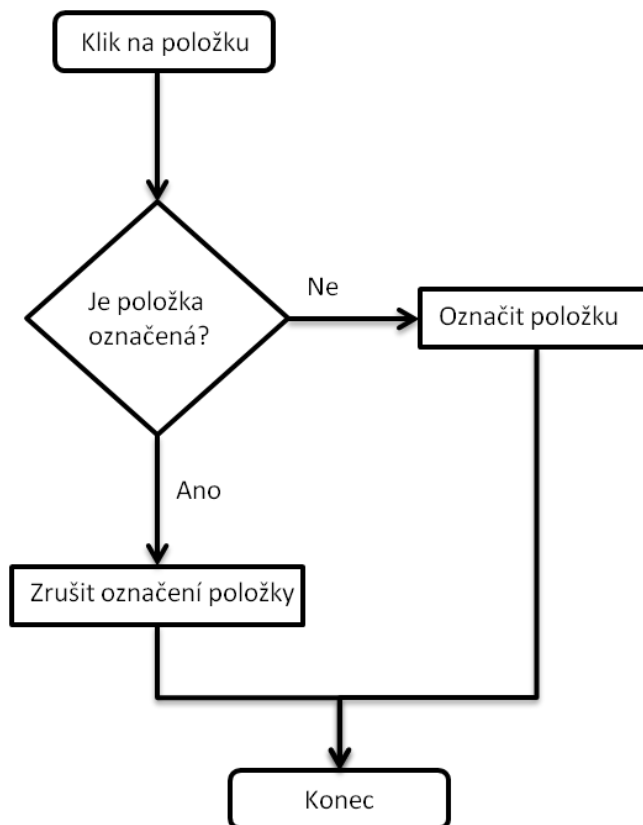
formátování. To jedna z nečekaných vlastností ListView. Programátor musí pro každé kliknutí sám udržovat seznam vybraných položek. Každé volání getView bude položky obarvovat/odbarvovat podle tohoto seznamu. Veškeré souborové operace se musí provádět výhradně podle tohoto seznamu nikoliv podle obarvení. Obarvení slouží výhradně pro orientaci uživatele.



*Obr. 76 Část, která se nevešla na obrazovku, bude po vysunutí zpátky na obrazovku zase odbarvená/bílá [zdroj vlastní]*

### Výběr a zrušení výběru souborů

Chce-li uživatel například zašifrovat soubory v aktuálním adresáři, nebylo by vhodné, aby uživatel označil jeden soubor. Kliknul na příkazovou nabídku zašifrovat, zadal heslo a postup opakoval pro každý soubor, který chce zašifrovat. V takových případech je vhodné umožnit uživateli výběr více souborů najednou.



Obr. 77 Vícenásobný výběr souborů [zdroj vlastní]

Vícenásobný výběr souborů uživatel provede postupným kliknutím na více řádků v souborové části ListView. Pro každý vybraný řádek se provede metoda `onListItemClick`. Metoda pomocí polohy aktuálně vybraného řádku v seznamu `oznaceneSoubory` zjistí, zda byl nebo nebyl řádek vybrán. Pokud nebyl aktuální řádek vybrán, je jeho hodnota nastavena v seznamu `oznaceneSoubory` na "vybran". Barva písma řádku je nastavena na červenou (červená barva odpovídá hodnotě `-65536`). Pokud byl aktuální řádek vybrán, je jeho hodnota nastavena v seznamu `oznaceneSoubory` na "nevybran". Barva písma řádku je nastavena na bílou (bílá barva odpovídá hodnotě `-4276546`). Barvu lze nastavit pohodlně i pomocí: `TV_radek_seznamu_souboru.setTextColor(Color.RED)`; Daný způsob však z nepochopitelných důvodů způsoboval ve starších verzích operačního systému Android problémy. Z tohoto důvodu je vhodnější použít číselné vyjádření.

Ukázka kódu:

```
@Override
protected void onListItemClick(final ListView lv, View v, int pozice,
long id)
{
    int vybranyRadek = (int)id;
```

```

...
...
...
if(vybranyRadek == 0)
{
...
...
...
}
...
...
...
else
{
    TextView TV_radek_sezmanu_souboru =
        (TextView)v.findViewById(R.id.TV_radek_sezmanu_souboru);

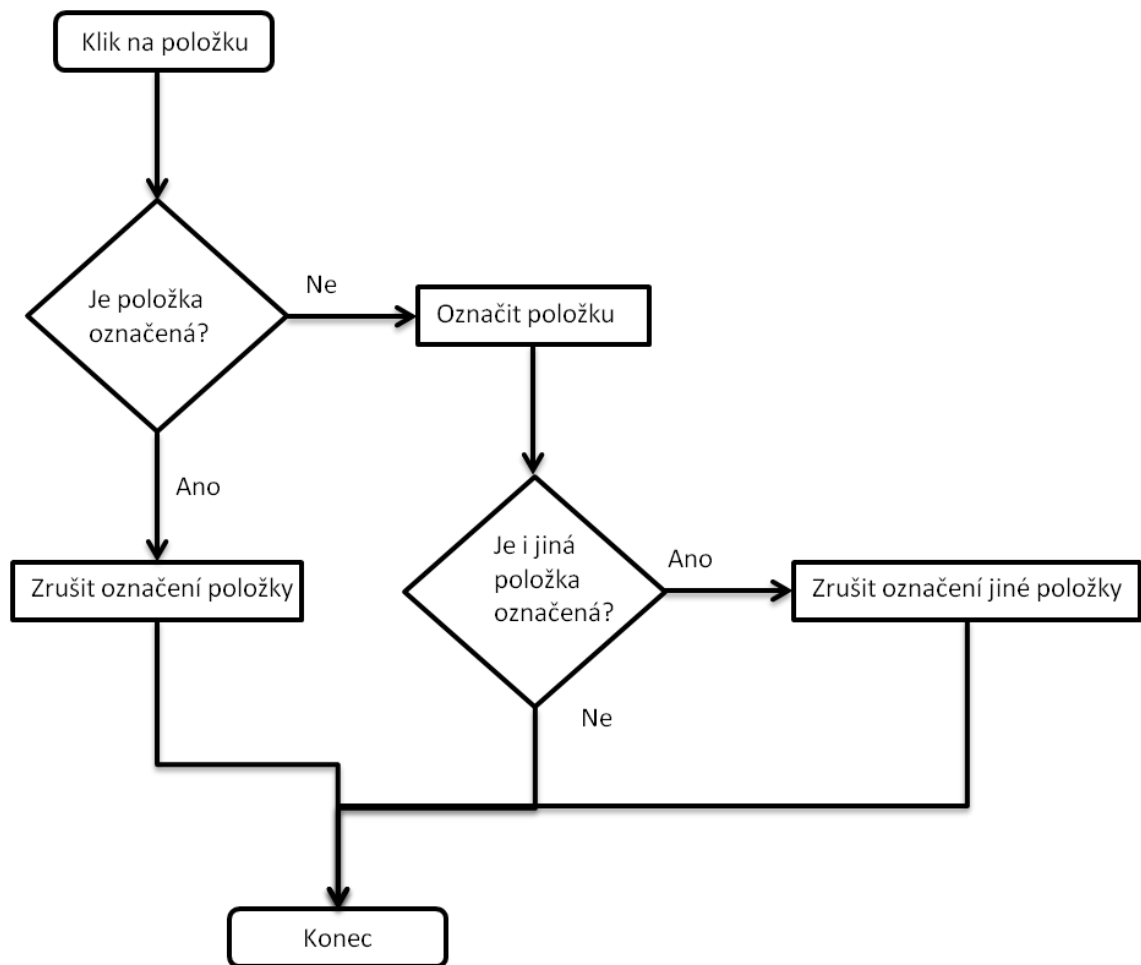
    if(soubor.isDirectory())
    {
        vratSoubory(soubor.listFiles());
        aktualniAdresar = soubor;
    }
    else
    {

        // OBARVENI  && vybranyRadek != 0 aby se neobarvoval
        // prvni radek tj. vybranyRadek s indexem 0
        if(oznaceneSoubory.get(vybranyRadek).equals("nevybran")
        && vybranyRadek != 0 && vybranyRadek != 1 ...)
        {
            TV_radek_sezmanu_souboru.setTextColor(-65536);
            oznaceneSoubory.set(vybranyRadek, "vybran");
        }
        // ODBARVENI
        else if(oznaceneSoubory.get(vybranyRadek).equals
        ("vybran")&& vybranyRadek != 0 && vybranyRadek != 1
...))
        {
            TV_radek_sezmanu_souboru.setTextColor(-4276546);
            oznaceneSoubory.set(vybranyRadek, "nevybran");
        }
    }
}

```

Na druhou stranu jsou úkony, kde je nutné vybrat pouze jediný soubor. Například výběr klíče pro šifrování. V takových případech je vhodné poskytnout uživateli funkcionalitu

výběru jednoho souboru. Postup, jak provádět jedno výběrové označování souborů, je naznačen na vývojovém diagramu, který je na obrázku 78.

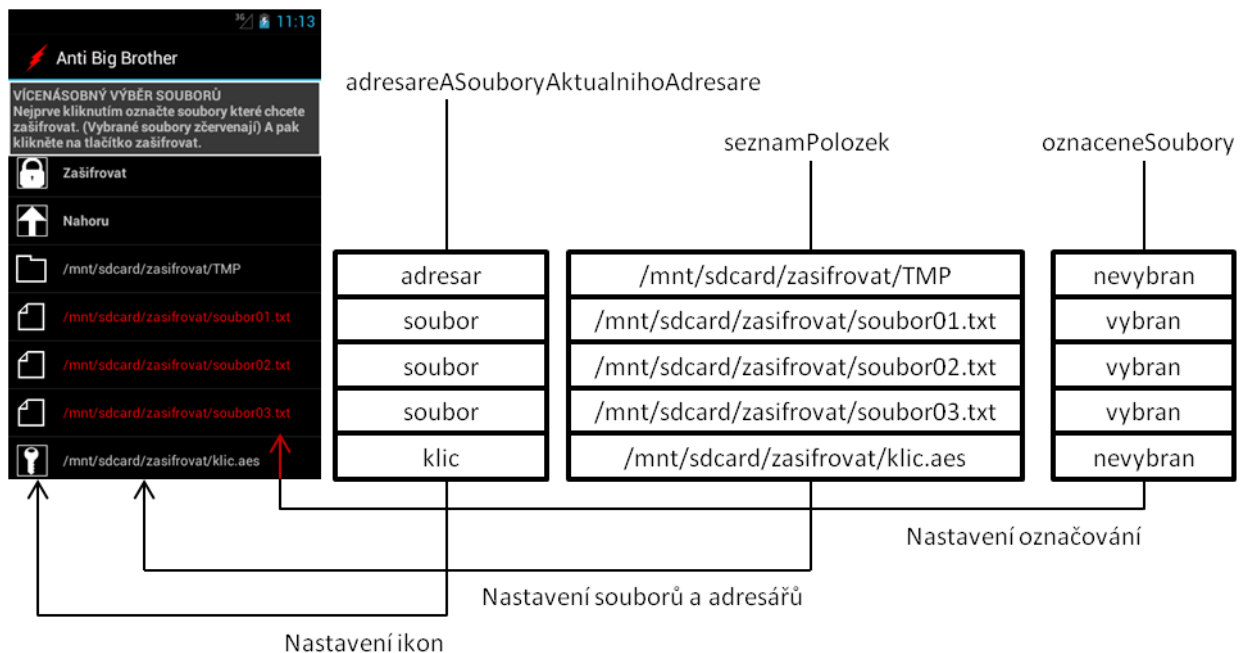


Obr. 78 Jedno výběrové označování souborů [zdroj vlastní]

Rozdíl oproti vícenásobnému výběru naznačenému výše je, že se všechny položky seznamu `oznaceneSoubory` nastaví na "nevybran". Pak se do seznamu na místo, které odpovídá aktuálně zvolenému řádku, nastaví hodnota "vybran". Následně se provede zaktualizovaný pohled.

Na obrázku 79 je vidět celkové řešení komponenty pro grafickou práci se soubory. Komponenta je vytvořena pomocí metod třídy `File`, třídy `IkonovyAdapter`, její přeepsané metody `getView`, která je zděděná z třídy `ArrayAdapter` a tří seznamů (`adresareASouboryAktualnihoAdresare`, `seznamPolozek`, `oznaceneSoubory`). Navržené řešení je výsledkem časově náročných pokusů na skutečném hardwaru a představuje nejlepší možný kompromis mezi rychlostí zobrazování na málo výkonných zařízeních a koncepční čistotou. Nabízí se řešení uložit výsledek volání metody `listFiles()` do seznamu typu `File` a pracovat s ním místo seznamů

adresareASouboryAktualnihoAdresare a seznamPolozek. Je nutné si, ale uvědomit, že metoda getView je volána při sebemenší změně v zobrazení. Pokaždé by se u všech souborů a adresářů musela vytvořit instance třídy File a zavolat minimálně metody getAbsolutePath(), getName(), isDirectory() a isFile(). Což vede ke zpomalení zejména při scrollování obrazovky v adresářích, které obsahují větší množství souborů. Při použití tří seznamů s položkami typu String se uvedené instance použijí jen jednou, a to při vstupu do prohlíženého adresáře. A při dalších změnách vedoucích ke změně zobrazení se již použijí připravené textové řetězce. Uvedený způsob je mnohem rychlejší. Za několik let až budou nejlevnější přístroje s operačním systémem Android mnohem výkonnější, budou obě řešení rovnocenná.



Obr. 79 Celkové řešení komponenty pro grafickou práci se soubory [zdroj vlastní]

## 4.5 Návrh aplikační logiky

### 4.5.1 Úvodní obrazovka

Po spuštění aplikace se zobrazí hlavní obrazovka aplikace. Danou činnost zajišťuje třída AntiBigBrotherActivity ve spolupráci s main.xml. Třídám představující vše, co je vidět na jednotlivých obrazovkách se říká aktivity. Třída AntiBigBrotherActivity představuje rozcestník činností, které může uživatel v aplikaci provádět:

- Správu klíčů
  - Vytvářet a ukládat klíče
  - Mazat klíče
- Hromadné šifrování souborů
  - Hromadné šifrování pomocí hesel
  - Hromadné šifrování pomocí klíčů
- Hromadné dešifrování souborů
  - Hromadné dešifrování pomocí hesel
  - Hromadné dešifrování pomocí klíčů
- Hromadné odesílání zašifrovaných souborů na SSHD server
- Stahování zašifrovaných souborů z SSHD serveru
- Nastavení aplikace
  - Registrace nového uživatele na SSHD serveru
- Zobrazení nápovědy

V příloze PI je vidět celá úvodní obrazovka kryptografické aplikace. Zachytit celou úvodní obrazovku lze jen díky grafickému triku, kdy byla postupně posunována hlavní obrazovka a nasnímány její jednotlivé části. Výsledek byl sestaven v grafickém editoru Gimp.

#### 4.5.2 Správa klíčů

Celá správa klíčů je vidět na obrázku 82. Úvodní obrazovku správy klíčů zajišťuje třída Klic0 a klic\_0.xml. Jedná se o rozcestník umožňující uživateli výběr mezi tvorbou klíče nebo mazáním.

#### 4.5.3 Mazání souborů klíčů

Mazání klíčů zajišťuje třída Klic1 a klic\_1.xml. Třída Klic1 je implementací komponenty pro grafickou práci se soubory, která je přizpůsobená pro mazání souborů klíčů. Po kliknutí do souborové části ListView se u souboru, na který uživatel kliknul, zjistí koncovka. Pokud odpovídá koncovce souboru klíče tj. „.aes“, je zobrazen AlertDialog s dotazem, zda chce uživatel soubor klíče skutečně smazat. Klikne-li uživatel na tlačítko ANO (PositiveButton) je soubor smazán. Neodpovídá-li koncovka souboru řetězci „.aes“, není uživateli umožněno soubor smazat. Uživatel je o této skutečnosti informován. Důvodem je snaha, aby byla tato část aplikace striktně používána uživateli pro správu klíčů a nikoliv jako správce souborů.

Ukázka kódu:

```

...
...
...
else
{
    int delkaJmenaSouboru = soubor.getName().length();
    String koncovka = soubor.getName().substring(delkaJmenaSouboru - 4,
delkaJmenaSouboru);

    if(koncovka.equals(".aes"))
    {
        AlertDialog.Builder smazatSouborDialog =
new
AlertDialog.Builder(this);

        smazatSouborDialog.setTitle("Mazání klíčů");
        smazatSouborDialog.setMessage("Opravdu chcete " +
soubor.getName() + " klíč
smazat?");

        smazatSouborDialog.setPositiveButton("ANO", new
DialogInterface.OnClickListener()
        {
            public void onClick(DialogInterface di, int id)
            {
                soubor.delete();

                vratSoubory(aktualniAdresar.listFiles());
            }
        }
    );

    smazatSouborDialog.setNegativeButton("NE", new
DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface di, int id)
        {
            // ZADNA AKCE
        }
    }
    );

    smazatSouborDialog.show();

}
// je to soubor ale neni to soubor klice
else
{
    new AlertDialog.Builder(this)
        .setTitle("Chyba !")
        .setMessage("Vámi vybraný soubor není klíč !\n\nProto nelze
smazat")
        .setNeutralButton("OK", new DialogInterface.OnClickListener()
        {
            public void onClick(DialogInterface di, int button)
            {

```

```
        // ZADNA AKCE
    }
    }) // konec zavorky setNeutralButton
    .show();
}
}
```

#### 4.5.4 Vytváření a ukládání klíčů

Vytváření a ukládání klíčů zajišťuje třída Klic2 a klic\_2.xml. Třída Klic2 je implementací komponenty pro grafickou práci se soubory, která je přizpůsobena pro vytváření souborů klíčů. Příkazová nabídka umožňuje:

1. Vytvořit a uložit soubor klíče.
2. Přesun o úroveň výš v adresářové struktuře.
3. Vytvořit nový adresář pro uložení klíče.

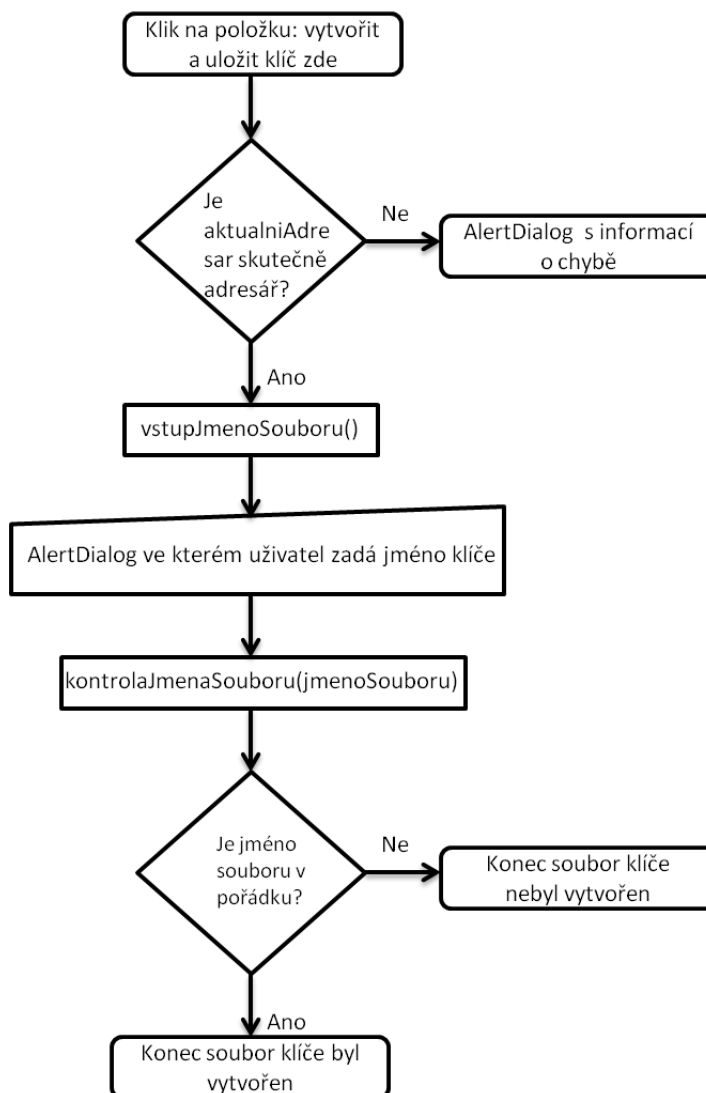
##### Nabídka vytvořit a uložit klíč zde

Kliknutím na položku Vytvořit a uložit klíč zde, je zavolána metoda `public void metodaBT1_vytvorit_klic(View view)`. Metoda zjistí, zda je `aktualniAdresar` standardní adresář. Pokud není, je uživatel informován, že program neumožňuje uložit klíč na zvolené místo a je vyzván k uložení klíče do jiného adresáře. Je-li `aktualniAdresar` standardní adresář, je zavolána metoda `private void vstupJmenoSouboru()`. Ta spustí `AlertDialog`, do kterého uživatel zadá jméno souboru klíče. Následně se provede kontrola jména souboru pomocí zavolání metody `kontrolaJmenaSouboru(jmenoSouboru)`. Zde se provede série následující kontrol:

1. Kontrola délky jména klíče. Jestliže je zadané jméno souboru delší než 16 znaků, skončí kontrola neúspěchem a uživateli je zobrazen `AlertDialog` s informací, že jméno souboru je příliš dlouhé.
2. Kontrola zda jméno souboru není prázdný řetězec. Pokud uživatel v `AlertDialogu` sloužícího pro zadání jména klíče pouze stisknul tlačítko OK bez zadání jména klíče, skončí kontrola neúspěchem. Uživateli je zobrazen `AlertDialog` s informací, že jméno souboru klíče nebylo zadáno.
3. Kontrola nepovolených znaků. Operační systém Android má Linuxové jádro, proto je vhodné zadávat jména důležitých souborů, jako jsou šifrovací klíče bez mezer, diakritiky a speciálních znaků. Kontrola je provedena pomocí regulárních výrazů. (Práce s regulárními výrazy je v Jazyce Java velmi pohodlná.)

4. Kontrola duplicity jména souboru. Ke jménu souboru zadaného uživatelem je přidána koncovka .aes. Poté se zjišťuje, zda jmenoSouboru.aes již adresáři existuje. Pokud ano, není nový soubor klíče vytvořen z důvodu přepsání starého souboru klíče. Všechny soubory zašifrované starým klíčem by uživatel již nemohl rozšifrovat. Uživatel je o této skutečnosti informován prostřednictvím AlertDialogu.

Soubor klíče je velmi důležitý pro správné fungování kryptografické aplikace. Z toho důvodu jsou na jeho jméno kladeny vysoké nároky. Ty mají za cíl nejen se vyhnout pádu aplikace ale i předejít možným budoucím problémům při samotném šifrování a dešifrování. Je-li jméno souboru klíče v pořádku, je hlavní vlákno rozděleno na vlákno, které vytvoří soubor klíče a zapíše ho do zvoleného adresáře v lokálním úložišti, a na vlákno, které uživateli zobrazuje ProgressDialog. Nakonec je provedena aktualizace pohledu na aktuální adresář, aby uživatel viděl nově vytvořený soubor klíče.



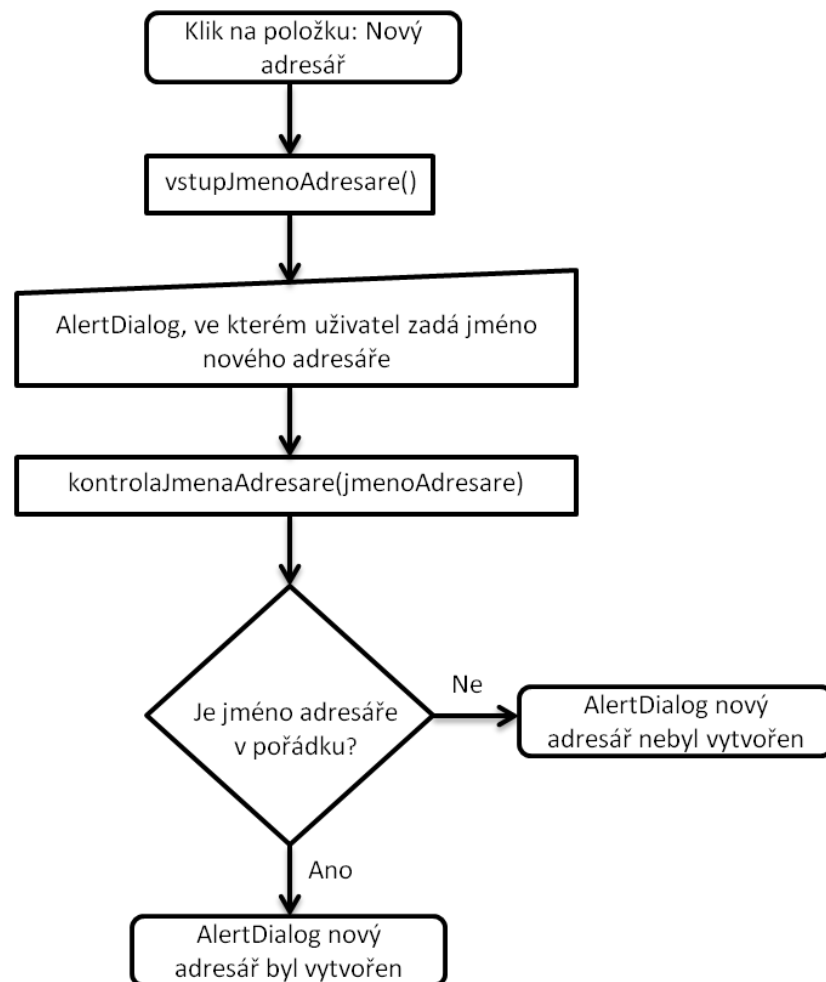
Obr. 80 Vytváření souborů klíčů [zdroj vlastní]

### Nabídka nový adresář

Kliknutím na položku Nový adresář, je zavolána metoda `public void metodaBT3_novy_adresar(View view)`. Ta stutí metodu `private void vstupJmenoAdresare()`. Následně se spustí `AlertDialog`, do kterého uživatel zadá jméno adresáře, který chce vytvořit. Následně se provede kontrola jména souboru pomocí zavolání metody `kontrolaJmenaAdresare(jmenoAdresare)`. Zde se provede série následujících kontrol:

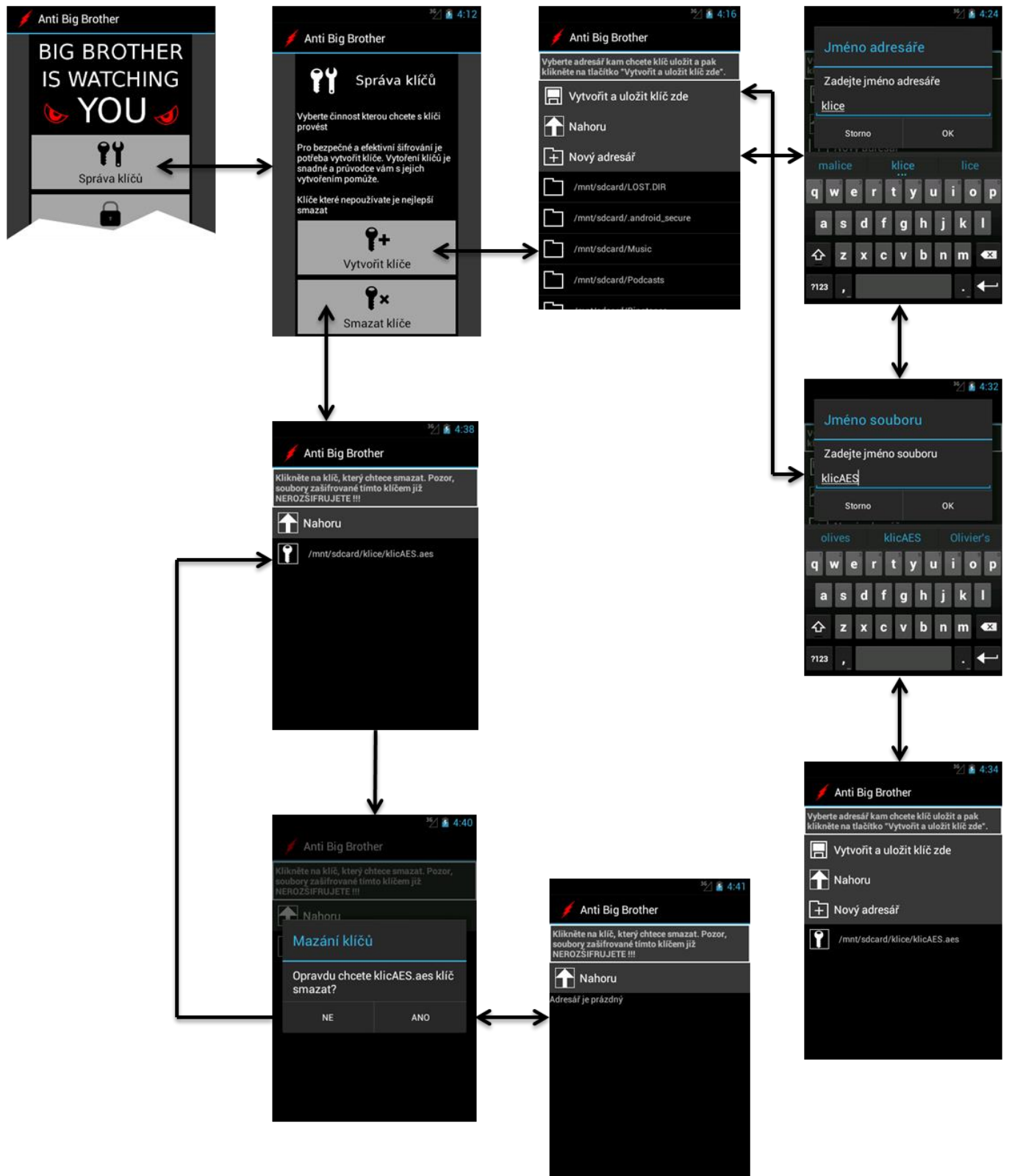
1. Kontrola délky jména adresáře. Pokud je zadané jméno adresáře delší než 16 znaků, skončí kontrola neúspěchem a uživateli je zobrazen `AlertDialog` s informací, že jméno adresáře je příliš dlouhé.

2. Kontrola zda jméno adresáře není prázdný řetězec. Pokud uživatel v AlertDialogu sloužícího pro zadání jména adresáře pouze stisknul tlačítko OK bez zadání jména adresáře, skončí kontrola neúspěchem. Uživateli je zobrazen AlertDialog s informací, že jméno adresáře nebylo zadáno.
3. Kontrola nepovolených znaků. Kontrola je provedena pomocí regulárních výrazů.
4. Kontrola duplicity jména adresáře. Pokud již existuje adresář s daným jménem, není nový adresář vytvořen, protože by duplicita vyvolala výjimku. Uživatel je o duplicitě informován prostřednictvím AlertDialogu.



Obr. 81 Vytvoření nového adresáře [zdroj vlastní]

Část aplikace, která se zabývá správou klíčů, je celá přehledně zobrazena na obrázku 82.



Obr. 82 Celková správa klíčů [zdroj vlastní]

### 4.5.5 Šifrování

Šifrování začíná kliknutím na tlačítko Šifrovat na hlavní obrazovce aplikace. Celý proces šifrování je vidět na obrázku 83. Úvodní obrazovku šifrování zajišťuje třída Sifrovani0 a sifrovani0\_xml. Jedná se o rozcestník, ve kterém si uživatel vybere, zda chce šifrovat pomocí klíče nebo pomocí hesla.

Po kliknutí na tlačítko Šifrovat klíčem se nastaví do proměnné stylSifrovani, hodnota "KLICEM". Poté se spustí aktivita reprezentovaná třídou Sifrovani1.

Po kliknutí na tlačítko Šifrovat heslem se nastaví do proměnné stylSifrovani, hodnota "HESLEM". Následně se spustí aktivita reprezentovaná třídou Sifrovani3.

Proměnná stylSifrovani patří třídě Sifrovani0. Třída Sifrovani3, která organizuje samotné šifrování, řídí styl šifrování podle hodnoty, kterou si přečte z této proměnné pomocí metody getStylSifrovani().

Ukázka možné realizace třídy Sifrovani0:

```
public class Sifrovani0 extends Activity
{
    // PROMENNE TRIDY
    //
    final Context kontext = this;
    //
    static String stylSifrovani;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sifrovani_0);
    }

    // PRIPUSTNE HODNOTY:
    // HESLEM
    // KLICEM
    public void setStylSifrovani(String stylSif)
    {
        stylSifrovani = stylSif;
    }

    public static String getStylSifrovani()
    {
        return stylSifrovani;
    }

    // TLACITKO Sifrovat heslem
    public void metodaBT1Sifrovani0(View view)
    {
        this.setStylSifrovani("HESLEM");
    }
}
```

```
        Intent intent = new Intent(kontext, Sifrovani3.class);
        startActivity(intent);
    }

    // TLACITKO Sifrovat klicem
    public void metodaBT2Sifrovani0(View view)
    {
        this.setStyleSifrovani("KLICEM");
        Intent intent = new Intent(kontext, Sifrovani1.class);
        startActivity(intent);
    }
}
```

### Šifrování pomocí klíče

Po kliknutí uživatele na tlačítko Šifrovat klíčem na úvodní obrazovce šifrování se spustí třída Sifrovani1 a sifrovani1.xml. Zde se může uživatel rozhodnout, zda chce k šifrování použít již existující klíč. V tom případě použije tlačítko Pokračovat. Pokud nemá uživatel ještě vytvořen žádný klíč nebo chce použít pro šifrování nový klíč, tak se nemusí vracet na hlavní obrazovku aplikace a z ní přecházet do správy klíčů. Stačí, když klikne na tlačítko Vytvořit klíč, které uživatele přepne přímo na úvodní obrazovku Správy klíčů. Pokud uživatel nechce využít ani jedné z nabízených možností, může se vrátit pomocí tlačítka Zpět na úvodní obrazovku šifrování.

Klikne-li uživatel na tlačítko Pokračovat, zobrazí se aktivita reprezentovaná třídou Sifrovani2 a sifrovani2.xml. Třída Sifrovani2 je implementací komponenty pro grafickou práci se soubory, která je přizpůsobená pro výběr jediného souboru klíče. Příkazová nabídka umožňuje pouze přesun o úroveň výš v adresářové struktuře a kliknutím vybrat soubor klíče. Po kliknutí na řádek v souborové části ListView metoda protected void onItemClick(ListView lv, View v, int pozice, long id) zjistí, zda je vybraný řádek adresář nebo soubor.

Je-li vybraný řádek adresář, pak je nastaven jako aktuální adresář a je vybudován pohled do tohoto adresáře, tak jak bylo popsáno výše.

Je-li vybraný řádek soubor, pak je provedena kontrola, zda je vybraný soubor klíč. Pokud je vybraný soubor skutečně klíč, tak je soubor klíče pomocí metody setSifrovaciKlic(soubor) nastaven jako šifrovací klíč, který se bude používat pro dané šifrování. Je zobrazena bublina s informací, že vybraný klíč byl přijat pro šifrování. Poznámka: Bubliny jsou jedním ze specifických informačních nástrojů operačního systému Android. Jsou velmi praktické k zobrazování tzv. provozních informací na rozdíl od AlertDialogu, protože uživatele nenutí ke kliku navíc.

Ukázka:

```
CharSequence hlaska = "Byl vybrán klíč:" + soubor.getName() + ".";
int trvani = Toast.LENGTH_SHORT;
Toast t = Toast.makeText(context, hlaska, trvani);
t.show();
```

Pak je spuštěna třída `Sifrovani3` a `sifrovani3_xml`. Třída `Sifrovani3` je implementací komponenty pro grafickou práci se soubory, která je přizpůsobená pro hromadné šifrování souborů. Příkazová nabídka umožňuje:

- Zašifrovat
- Přesun o úroveň výš v adresářové struktuře
- Hromadné označování souborů v souborové části `ListView`

Po označení souborů určených k zašifrování a kliknutí na položku `Zašifrovat` se hlavní vlákno rozdělí na vlákno, které na pozadí šifruje vybrané soubory, a vlákno, které zobrazuje `ProgressDialog` uživateli. Šifrovací vlákno je společné pro šifrování pomocí klíče i pro šifrování pomocí hesla, proto bude popsáno na konci této podkapitoly o šifrování. V posledním kroku se zruší označení souborů, které byly vybrány k zašifrování. Následně je zaktualizován pohled na zašifrované soubory v aktuálním adresáři:

```
upravSeznamOznaceneSoubory(aktualniAdresar.listFiles());
vratSoubory(aktualniAdresar.listFiles());
```

### Šifrování pomocí hesla

Kliknutí uživatele na tlačítko `Šifrovat` heslem na úvodní obrazovce šifrování se spustí přímo třída `Sifrovani3` a `sifrovani3_xml`. Po označení souborů určených k zašifrování, a kliknutí na položku `Zašifrovat` se hlavní vlákno rozdělí na vlákno, které na pozadí šifruje vybrané soubory, a vlákno, které zobrazuje `ProgressDialog` uživateli. V posledním kroku se zruší označení souborů, které byly vybrány k zašifrování. Je zaktualizován pohled na zašifrované soubory v aktuálním adresáři.

Samotné šifrování probíhá následovně:

Nejprve se zjistí pomocí metody `Sifrovani0.getStylSifrovani()` styl šifrování.

Je-li styl šifrování nastaven na `"KLICEM"`, je zkontrolován soubor klíče pomocí:

```
if(Sifrovani2.getSifrovaciKlic().getName() != null ||
(Sifrovani2.getSifrovaciKlic().getName().length() > 0))
{
```

```

    ...
    ...
    ...
}

```

Pokud je soubor klíče v pořádku, je postupně procházen seznam seznamPolozek, a pokud je na stejném místě seznamu v oznaceneSoubory hodnota "vybran", je provedeno zašifrování daného souboru:

```

final ZasifrujAESKlicem za = new ZasifrujAESKlicem(soubor);
za.zasifruj();

```

Je-li styl šifrování nastaven na "HESLEM", je zkontrolováno heslo. Pokud je heslo prázdné, je zavolána metoda třídy Sifrovani3 jménem nastavHeslo:

```

if(Sifrovani0.getStylSifrovani() == "HESLEM" && heslo.equals(""))
{
    nastavHeslo(backgroundThread);
}

```

```

...
...
...

```

Je důležité provést nastavení hesla v samostatné metodě mimo vlákno na pozadí (šifrovací vlákno), protože pokud by se objevil AlertDialog přímo v tomto vlákně, byla by aplikace bez milosti ukončena. Pokud by bylo heslo nastaveno na samém začátku metody:

```

@Override
protected void onItemClick(final ListView lv, View v, int pozice,
long id)
{
    int vybranyRadek = (int)id;

    if(vybranyRadek == 0)
    {
        // NASTAVIT HESLO ZDE

        ...
        ...
        ...
    }
}

```

Nejprve by se vykonalo šifrovací vlákno s prázdným heslem a heslo by bylo nastaveno až po ukončení vlákna.

Proto se šifrovací vlákno vezme sebou do metody nastavHeslo. Pomocí AlertDialogu se nastaví heslo a teprve pak se šifrovací vlákno spustí. V šifrovacím vlákně na pozadí je

postupně procházen seznam seznamPolozek, a pokud je na stejném místě v seznamu oznaceneSoubory hodnota "vybran", je provedeno zašifrování daného souboru:

```
final ZasifrujAESHeslem zah = new ZasifrujAESHeslem(soubor, heslo);
zah.zasifruj();
```

Třídy použité v ukázce ZasifrujAESKlicem a ZasifrujAESHeslem jsou implementací postupů popsaných v kapitole 3.

Po ukončení šifrovacího vlákna na pozadí se provede nastavení hesla na prázdný řetězec a aktualizace pohledu do aktuálního adresáře na zašifrované soubory. Následuje ukázka možného řešení metody nastavHeslo:

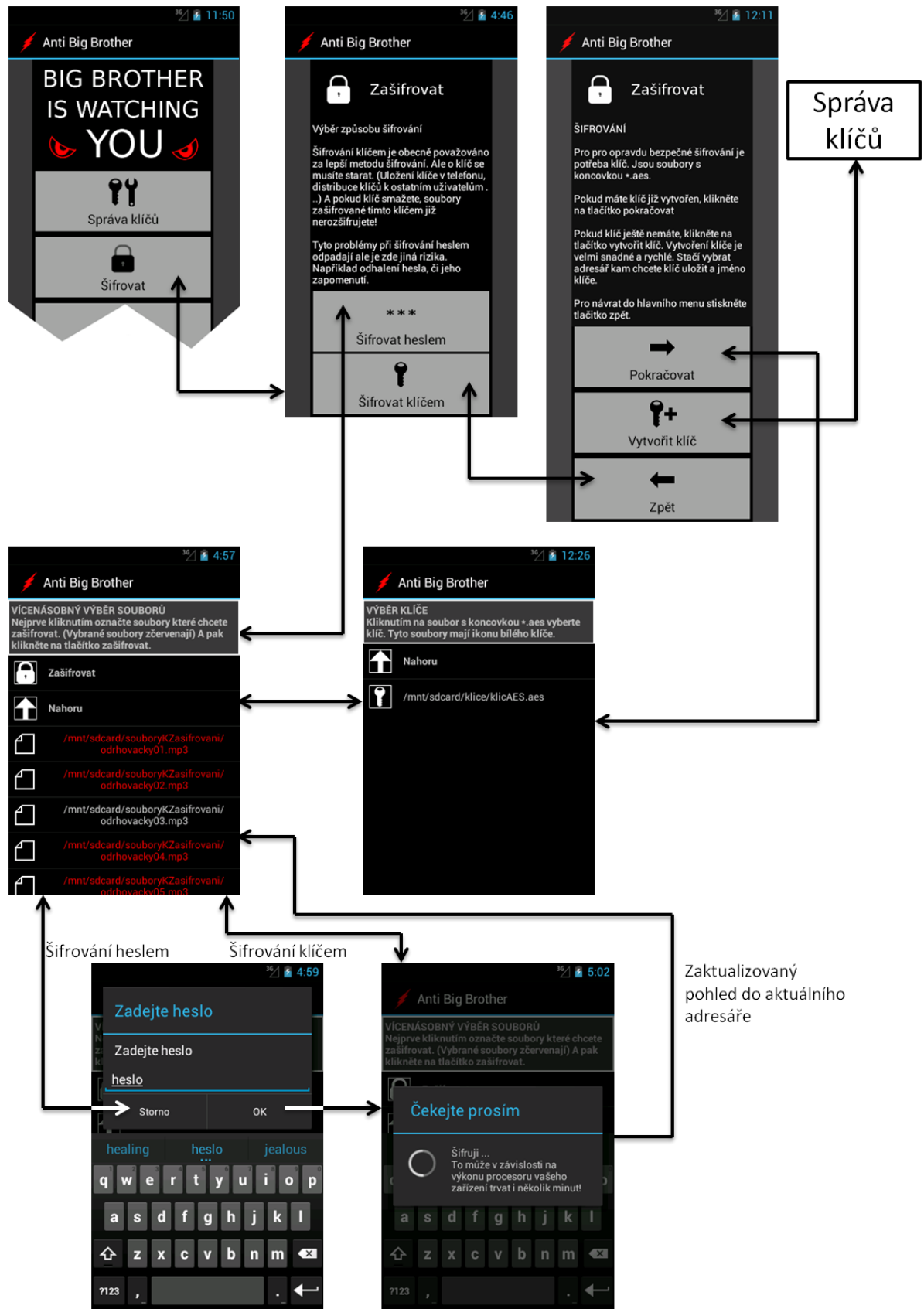
```
private void nastavHeslo(final Thread vlaknoNaPozadi)
{
    AlertDialog.Builder vstupHeslo = new AlertDialog.Builder(this);
    vstupHeslo.setTitle("Zadejte heslo");
    vstupHeslo.setMessage("Zadejte heslo");

    final EditText etVstup = new EditText(this);
    vstupHeslo.setView(etVstup);

    vstupHeslo.setPositiveButton("OK", new
    DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface di, int id)
        {
            heslo = etVstup.getText().toString().trim();
            vlaknoNaPozadi.start();
        }
    }
    );

    vstupHeslo.setNegativeButton("Storno",
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface di, int id)
        {
            // Storno - zadna akce
        }
    }
    );

    vstupHeslo.show();
}
```



Obr. 83 Celý proces šifrování [zdroj vlastní]

#### 4.5.6 Dešifrování

Dešifrování začíná kliknutím na tlačítko Dešifrovat na hlavní obrazovce aplikace. Celý proces dešifrování je vidět na obrázku 84. Úvodní obrazovku dešifrování zajišťuje třída Desifrovani0 a desifrovani0\_xml. Jedná o rozcestník, ve kterém si uživatel vybere, zda chce dešifrovat pomocí klíče nebo pomocí hesla. Desifrovani0 a desifrovani0\_xml jsou velmi podobné Sifrovani0 a sifrovani0\_xml.

Po kliknutí na tlačítko Dešifrovat klíčem se nastaví do proměnné stylDesifrovani, hodnota "KLICEM". Poté se spustí aktivita reprezentovaná třídou Desifrovani1.

Po kliknutí na tlačítko Dešifrovat heslem se nastaví do proměnné stylDesifrovani, hodnota "HESLEM". Pak se spustí aktivita reprezentovaná třídou Desifrovani2.

Proměnná stylDesifrovani patří třídě Desifrovani0. Třída Desifrovani2, která organizuje samotné dešifrování, řídí styl dešifrování podle hodnoty, kterou si přečte z této proměnné pomocí metody getStylDesifrovani().

##### Dešifrování pomocí klíče

Po kliknutí uživatele na tlačítko Dešifrovat klíčem na úvodní obrazovce dešifrování se spustí třída Desifrovani1 a desifrovani1\_xml. Třída Desifrovani1 je implementací komponenty pro grafickou práci se soubory, která je přizpůsobená pro výběr jediného souboru klíče. Příkazová nabídka umožňuje pouze přesun o úroveň výš v adresářové struktuře a kliknutím vybrat soubor klíče. Po kliknutí na řádek v souborové části ListView metoda protected void onItemClick(ListView lv, View v, int pozice, long id) zjistí, zda je vybraný řádek adresář nebo soubor.

Je-li vybraný řádek adresář, pak je nastaven jako nový aktuální adresář, a je vybudován pohled do tohoto adresáře.

Je-li vybraný řádek soubor, pak je provedena kontrola, zda je vybraný soubor klíč. Pokud je vybraný soubor skutečně klíč, tak je soubor klíče pomocí metody setDesifrovaciKlic(soubor) nastaven jako dešifrovací klíč, který se bude používat pro aktuální dešifrování. Následně je zobrazena bublina s informací, že vybraný klíč byl přijat pro dešifrování.

Následně je spuštěna třída Desifrovani2 a desifrovani2\_xml. Třída Desifrovani2 je implementací komponenty pro grafickou práci se soubory, která je přizpůsobena pro hromadné dešifrování souborů. Příkazová nabídka umožňuje:

- Dešifrovat
- Přesun o úroveň výš v adresářové struktuře
- Hromadné označování souborů v souborové části ListView

Po označení souborů určených k dešifrování a kliknutí na položku Dešifrovat se hlavní vlákno rozdělí na vlákno, které na pozadí dešifruje vybrané soubory, a vlákno, které zobrazuje ProgressDialog uživateli. Dešifrovací vlákno je společné pro dešifrování pomocí klíče i pro dešifrování pomocí hesla, proto bude popsáno na konci této podkapitoly o dešifrování. V posledním kroku se zruší označení souborů, které byly vybrány k dešifrování. Je zaktualizován pohled na dešifrované soubory v aktuálním adresáři.

### Dešifrování pomocí hesla

Kliknutí uživatele na tlačítko Dešifrovat heslem na úvodní obrazovce dešifrování se spustí přímo třída Desifrovani2 a sifrovani2\_xml. Po označení souborů určených k dešifrování a kliknutí na položku Dešifrovat se hlavní vlákno rozdělí na vlákno, které dešifruje na pozadí vybrané soubory, a vlákno, které zobrazuje ProgressDialog uživateli. V posledním kroku se zruší označení souborů, které byly vybrány k dešifrování. Je zaktualizován pohled na dešifrované soubory v aktuálním adresáři.

Samotné dešifrování probíhá následovně:

Nejprve se pomocí metody `Desifrovani0.getStylDesifrovani()` zjistí styl dešifrování.

Je-li styl dešifrování nastaven na "KLICEM", je zkontrolován soubor klíče pomocí:

```
if(Desifrovani1.getDesifrovaciKlic().getName() != null ||
(Desifrovani1.getDesifrovaciKlic().getName().length() > 0))
{
    ...
    ...
    ...
}
```

Pokud je soubor klíče v pořádku, je postupně procházen seznam seznamPolozek, a pokud je na stejném místě seznamu v oznaceneSoubory hodnota "vybran", je provedeno dešifrování daného souboru:

```
final DesifrujAESKlicem dak = new DesifrujAESKlicem(soubor);
dak.desifruj();
```

Je-li styl dešifrování nastaven na "HESLEM", je zkontrolováno heslo. Pokud je heslo prázdné, je zavolána metoda třídy Desifrovani2 jménem nastavHeslo:

```

if(Desifrovani0.getStylDesifrovani() == "HESLEM" && heslo.equals(""))
{
    nastavHeslo(backgroundThread);
}
...
...
...

```

Stejně jako u šifrování je důležité provést nastavení hesla v samostatné metodě mimo vlákno na pozadí (dešifrovací vlákno), protože pokud by se objevil AlertDialog přímo v tomto vlákně byla by aplikace bez milosti ukončena. Pokud by bylo heslo nastaveno na samém začátku metody:

```

@Override
protected void onItemClick(final ListView lv, View v, int pozice,
long id)
{
    int vybranyRadek = (int)id;

    if(vybranyRadek == 0)
    {
        // NASTAVIT HESLO ZDE

        ...
        ...
        ...
    }
}

```

Nejprve by se vykonalo dešifrovací vlákno s prázdným heslem a heslo by bylo nastaveno až po ukončení vlákna.

Proto se dešifrovací vlákno vezme sebou do metody nastavHeslo, pomocí AlertDialogu se nastaví heslo a teprve pak se vlákno spustí. V dešifrovacím vlákně na pozadí je postupně procházen seznam seznamPolozek, a pokud je na stejném místě v seznamu oznaceneSoubory hodnota "vybran", je provedeno dešifrování daného souboru. Metoda nastavHeslo třídy Desifrovani2 je velmi podobná metodě nastavHeslo patřící třídě Sifrovani3.

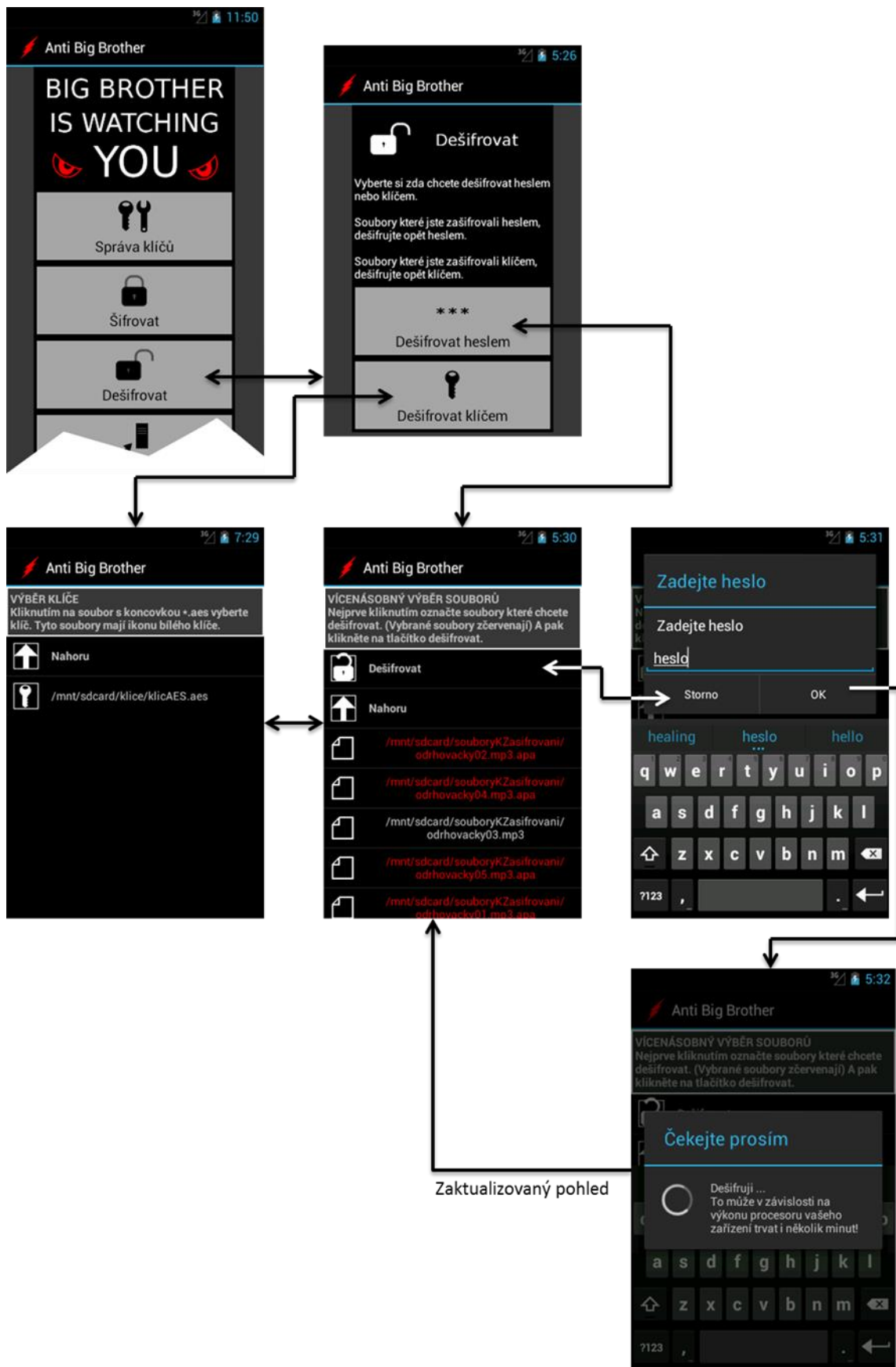
```

final DesifrujAESHeslem dah = new DesifrujAESHeslem(soubor, heslo);
dah.desifruj();

```

Po ukončení dešifrovacího vlákna na pozadí se provede nastavení hesla na prázdný řetězec a aktualizace pohledu do aktuálního adresáře na zašifrované soubory.

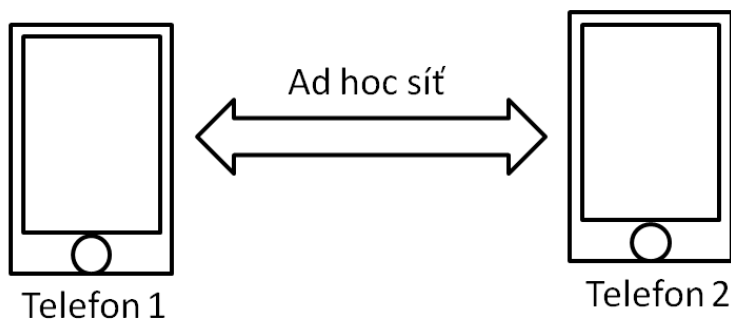
Třídy DesifrujAESKlicem a DesifrujAESHeslem jsou implementací postupů popsanych v kapitole 4.



Obr. 84 Celý proces dešifrování [zdroj vlastní]

## 5 PŘENOS ŠIFROVANÝCH SOUBORŮ MEZI MOBILNÍMI ZAŘÍZENÍMI S OPERAČNÍM SYSTÉMEM ANDROID

Pokud má být přenos dat mezi mobilními zařízeními s operačním systémem Android skutečně použitelný jako jeden z efektivních nástrojů firemní komunikace, pak není možné si vystačit s Ad hoc sítí (obrázek 85) vytvořenou mezi dvěma telefony nebo více telefony s dosahem pouhých pár metrů.



Obr. 85 Ad hoc síť [zdroj vlastní]

Mnoho velkých firem má pobočky po celém světě a jejich zaměstnanci potřebují sdílet svá data odkudkoliv bez ohledu na vzdálenosti či časové posuny. Z uvedených potřeb vyplývá, že navržené řešení musí zahrnovat server, na který se budou ukládat zašifrovaná uživatelská data, a který poběží 24 hodin denně. Server bude přístupný prostřednictvím sítě internet, tedy prostřednictvím nedůvěryhodné sítě. Takové řešení je vidět na obrázku 86. Uživatelé mohou ke svým datům přistupovat prostřednictvím GPRS, EDGE, HSPA i Wifi.

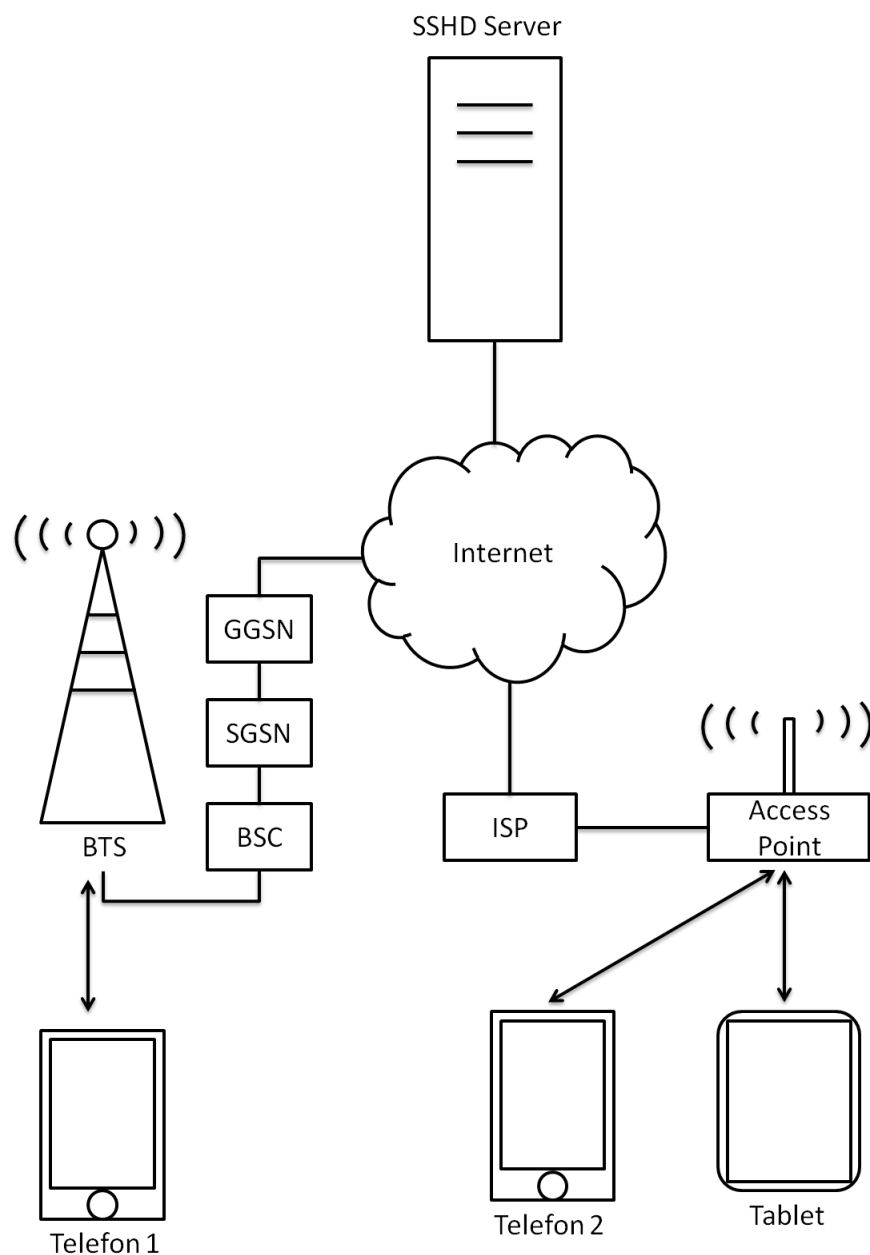
### 5.1 Java Secure Channel

Pro skutečně bezpečný přenos dat přes nedůvěryhodnou síť je potřeba použít odpovídající prostředky, které mají vysoký standard bezpečnosti. Technologie splňující všechny bezpečnostní požadavky se jmenuje SSH2. Ve světě Javy existuje čistá implementace SSH2, která se nazývá Java Secure Channel (JSCH). JSCH umožňuje připojení k SSHD serveru, použít port forwarding, X11 forwarding, přenos souborů, atd. JSCH umožňuje programátorům integrovat funkcionalitu SSH2 do svých vlastních programů napsaných v jazyce Java. JSCH lze volně používat.

### 5.2 Použití JSCH ve svém programu

Aby bylo možné používat funkcionalitu poskytovanou JSCH ve svých programech, je potřeba provést následující kroky:

1. Z webových stránek <http://www.jcraft.com/jsch> stáhnout knihovnu jsch-0.1.49.jar.
2. Nakopírovat soubor jsch-0.1.49.jar do adresáře ...\AdresarProjektu\libs.
3. Ve vývojovém prostředí Eclipse v okně Package Explorer kliknout pravým tlačítkem na projekt a vybrat nabídku Build Path. Poté vybrat Configure Build Path... Na kartě Libraries kliknout na tlačítko Add External JARs... a vybrat soubor jsch-0.1.49.jar, který je uložen v ...\AdresarProjektu\libs. Pro dokončení stisknout tlačítko OK.



Obr. 86 Přenos zašifrovaných souborů mezi mobilními zařízeními a serverem [zdroj vlastní]

Nyní je možné začít funkcionalitu JSCH používat ve svých kódech. Ukázka použití kódu, který pomocí uživatelského jména a hesla stáhne z SSHD serveru požadovaný soubor a uloží ho v lokálním úložišti telefonu/tabletu:

1. Vytvoří se instance třídy JSCH:

```
JSch jsch = new JSch();
```

2. Vytvoří se sezení:

```
Session sezeni = null;
```

3. Nyní je možné realizovat samotné spojení s SSHD serverem a přenos souboru. Je důležité, aby byla tato část v bloku try:

- a. Naplnění sezení: `sezeni =`

```
jsch.getSession(jmenoUzivatele, ipAdresa, 22);
```

Bude se používat standardní TCP port číslo 22.

- b. Na serveru neproběhne HostKeyChecking, protože se v tomto případě nepoužívá pro ssh spojení interaktivní konzole. Předejde se daným nastavením mnoha problémům:

```
sezeni.setConfig("StrictHostKeyChecking", "no");
```

- c. Nastavení hesla pro spojení: `sezeni.setPassword(password);`

- d. Vytvoření spojení s SSHD serverem: `sezeni.connect();` Ssh spojení lze na serveru sledovat pomocí: `cat /var/log/messages | grep sshd.`

- e. Vytvoření kanálu a jeho použití:

```
sftp: Channel kanal = sezeni.openChannel("sftp");
```

```
kanal.connect();
```

```
ChannelSftp chSftp = (ChannelSftp) kanal;
```

- f. Stažení souboru se provede:

- i. tak, že se vytvoří outputstream, pomocí kterého se bude zapisovat stažený soubor ze serveru do lokálního úložiště telefonu/tabletu:

```
OutputStream os = new
```

```
FileOutputStream(cilovySoubor);
```

- ii. samotné stažení a zápis se provede pomocí `OutputStreamu`:

```
chSftp.get(zdrojovySoubor, os);
```

- g. Stažení souboru bylo dokončeno, proto je možné provést odpojení:

```
chSftp.exit();
```

```
kanal.disconnect();  
sezeni.disconnect();
```

4. Na závěr je vhodné v catch části ošetřit výjimky:

```
JSchException , SftpException. Všechny ostatní výjimky se zachytí  
v obecné Exception.
```

### 5.3 Zapouzdření funkcionality JSCH do tříd

Pro přehlednost výsledného kódu a jeho neopakování se, kdykoliv bude potřeba použít funkce JSCH, autor práce zapouzdřil funkcionalitu JSCH do specializovaných tříd. JSCH specializované třídy lze použít i v jiných projektech.

SSHGet – pomocí uživatelského jména a hesla stáhne z SSHD serveru požadovaný soubor a uloží ho v lokálním úložišti telefonu/tabletu. Konstruktor:

```
public SSHGet(String zdrojovySoubor, String cilovySoubor,  
String ipAdresa, String jmenoUzivatele, String passwd)
```

SSHGetByKey – pomocí uživatelského jména a klíče stáhne z SSHD serveru požadovaný soubor a uloží ho v lokálním úložišti telefonu/tabletu. Konstruktor:

```
public SSHGetByKey(String klic, String zdrojovySoubor, String  
cilovySoubor, String ipAdresa, String jmenoUzivatele)
```

SSHPut – pomocí uživatelského jména a hesla se na SSHD server nahraje požadovaný soubor. Konstruktor:

```
public SSHPut(String souborKPreosu, String  
cilovyAdresarNaServeru, String ipAdresa, String  
jmenoUzivatele, String passwd)
```

SSHPutByKey – pomocí uživatelského jména a klíče se na SSHD server nahraje požadovaný soubor. Konstruktor:

```
public SSHPutByKey(String klic, String souborKPreosu, String  
cilovyAdresarNaServeru, String ipAdresa, String  
jmenoUzivatele)
```

SSHCommand – vytvoří ssh spojení se serverem pomocí uživatelského jména a hesla. Na SSHD serveru provede příkaz specifikovaný v konstruktoru. Je důležité, aby měl uživatel, pod jehož serverovým účtem jsou prováděny příkazy, odpovídající úroveň oprávnění. Konstruktor:

```
public SSHCommand(String prikaz, String ipAdresa, String
jmenoUzivatele, String passwd)
```

SSHCommandByKey – vytvoří ssh spojení se serverem pomocí uživatelského jména a klíče. Na SSHD serveru provede příkaz specifikovaný v konstruktoru. Je důležité, aby měl uživatel, pod jehož serverovým účtem jsou prováděny příkazy, odpovídající úroveň oprávnění. Konstruktor:

```
public SSHCommandByKey(String klic, String prikaz, String
ipAdresa, String jmenoUzivatele)
```

SSHCommandByKeygetResult – vytvoří ssh spojení se serverem pomocí uživatelského jména a klíče. Na SSHD serveru provede příkaz specifikovaný v konstruktoru a jeho výsledek uloží v proměnné stavPrikazu. Výsledek si mohou ostatní třídy přečíst pomocí veřejné metody getStavPrikazu(). Třída SSHCommandByKeygetResult je vhodná například, pokud potřebuje program získat výpis adresáře vzdáleného serveru. Je důležité, aby měl uživatel, pod jehož serverovým účtem jsou prováděny příkazy, odpovídající úroveň oprávnění. Konstruktor:

```
public SSHCommandByKeygetResult(String klic, String prikaz,
String ipAdresa, String jmenoUzivatele)
```

U všech tříd jsou ošetřeny výjimky a je zjišťován exit status. Je-li exit status -1, proběhlo celé spojení v pořádku.

```
if(kanal.getExitStatus() == -1)
{
    vysledek = "OK";
}
```

Je dobré zjišťovat, zda proběhla všechna odpojení. Například pomocí:

- `chSftp.isClosed();`
- `sezeni.isConnected();`
- `kanal.isConnected();`

## 5.4 Registrace

Aby mohl vlastník telefonu začít využívat služeb SSHD serveru, musí provést registraci na SSHD serveru. V návrhu šifrovací aplikace může registrace vypadat například takto:

Registrace začíná kliknutím uživatele na tlačítko Nastavení na hlavní obrazovce aplikace. Celý proces registrace je vidět na obrázku 87. Úvodní obrazovku nastavení zajišťuje třída

NastaveniUvod a nastaveni\_uvod.xml. Zde uživatel klikne na tlačítko Registrovat. Po kliknutí na tlačítko Registrovat je spuštěna metoda public void metodaBT1NastaveniUvod(View view). Metoda provede kontrolu předchozí registrace pomocí instance třídy KontrolaPredchoziRegistrace a její metody provedKontrolu(). Aplikace používá pro uživatelskou komunikaci s SSHD serverem tři soubory:

- zašifrovaný soukromý klíč uživatele: m16o01.abb
- zašifrovaný veřejný klíč uživatele: m16o02.pub.abb
- zašifrované jméno účtu uživatele na SSHD serveru: m16o03.abb

Všechny tři soubory jsou uloženy v lokálním úložišti zařízení v adresáři .m16o. Metoda provedKontrolu() vrací nezápornou celočíselnou hodnotu v rozsahu od 0 do 3:

- 0 – Byla pouze vytvořena instance třídy KontrolaPredchoziRegistrace, ale nebyla dosud spuštěna její metoda provedKontrolu(). Cílem je zabránit špatné implementaci třídy KontrolaPredchoziRegistrace, kdy by se mohlo stát, že aplikační programátor vytvoří pouze instanci třídy KontrolaPredchoziRegistrace, ale zapomene zavolat její metodu metoda provedKontrolu().
- 1 – Všechny tři soubory m16o01.abb, m16o02.pub.abb a m16o03.abb v adresáři .m16o neexistují. Registrace nebyla dosud provedena.
- 2 – Existuje některý ze tří souborů m16o01.abb, m16o02.pub.abb a m16o03.abb v adresáři .m16o. Registrace byla již provedena, ale soubory nejsou kompletní.
- 3 – Všechny tři soubory m16o01.abb, m16o02.pub.abb a m16o03.abb v adresáři .m16o existují. Registrace již byla provedena a soubory jsou kompletní.

Je-li návratová hodnota rovna jedné, je možné provést novou registraci pomocí metody provedRegistraci(), která patří třídě NastaveniUvod.

Je-li návratová hodnota rovna dvěma, znamená to, že registrace je poškozena. Je zobrazeno dialogové okno uživateli, ve kterém se rozhodne, zda chce vytvořit novou registraci. Rozhodne-li se uživatel pro novou registraci, je zavolána metoda provedRegistraci().

Ve všech ostatních případech není nová registrace uživateli umožněna. Je žádoucí, aby každý uživatel měl právě jednu registraci.

Metoda provedRegistraci by měla z důvodu možné časové náročnosti svou činnost rozdělit do dvou vláken. Jedno vlákno pomocí instance třídy Registrace provádí registraci a druhé vlákno uživateli zobrazuje ProgressDialog. Možný příklad metody provedRegistraci:

```
private void provedRegistraci()
{
    final ProgressDialog dialog = ProgressDialog.show(kontext, "Čekejte
    prosím", "Provádím Vaši registraci na SSHD serveru zabezpečeným kanálem.
    To může v závislosti na výpočetní výkonu Vašeho zařízení a kvalitě Vašeho
    připojení trvat i několik minut. Tuto akci nepřerušujte!");

    Thread vlaknoNaPozadi = new Thread(new Runnable()
    {
        @Override
        public void run()
        {
            Registrace registrace = new Registrace();
            registrace.provedRegistraci();

            runOnUiThread(new Runnable()
            {
                @Override
                public void run()
                {
                    dialog.dismiss();
                }
            });
        }
    });

    vlaknoNaPozadi.start();
}
```

Metoda `provedRegistraci()` třídy `Registrace` vytvoří instanci třídy `KontrolaSouborovehoSystemuAplikace` a pomocí její metody `provedKontrolu()` provede kontrolu souborového systému aplikace.

Kontrola adresářů:

- Neexistuje-li adresář `.AntiBigBrother` v lokálním úložišti, je vytvořen.
- Neexistuje-li adresář pro ukládání klíčů jménem `.m16om`, je vytvořen v adresáři `.AntiBigBrother`.
- Neexistuje-li servisní adresář jménem `.m25o`, je vytvořen v adresáři `.AntiBigBrother`.
- Neexistuje-li dočasný adresář jménem `.m34o`, je vytvořen v adresáři `.AntiBigBrother`.

Kontrola souborů:

Kontrola zjistí, zda existuje soukromý zašifrovaný klíč účtu registrace `m25o01.abb` v adresáři `.m25o`. Serverový účet registrace provádí proces registrace na SSHD serveru. Pokud neexistuje, je pomocí třídy `SSHGet` stažen do mobilního zařízení. Třída `SSHGet` použije uživatelské jméno `klicregistrace` a hlavní heslo aplikace pro stažení zašifrovaného

klíče m25o01.abb ze serveru do adresáře .m25o. Účet na SSHD serveru klicregistrace má jediný účelem, tím je poskytovat zašifrovaný RSA klíč /home/klicregistrace/m25o01.abb účtu registrace. Klíč m25o01.abb je zašifrován hlavním heslem aplikace a má unixové oprávněními 400.

Je důležité, aby byla kontrola souborového systému na tomto místě provedena. Pokud by souborový systém aplikace nebyl v pořádku, mohlo by to vážně poškodit stabilitu procesu registrace. Kontrolu souborového systému aplikace je vhodné provádět na více místech aplikace, proto je navržena univerzálním způsobem a je zapouzdřena do samostatné třídy.

Pokud je souborový systém v pořádku, může metoda provedRegistraci() v registraci pokračovat: Instance třídy DesifrujAESHeslem dočasně dešifruje soukromý klíč účtu registrace m25o01.abb. Instance třídy VytvorParametr vytvoří parametr ve tvaru RRRR.MM.DD.HH.SS.NNN, kde:

- RRRR je rok.
- MM je měsíc.
- DD je den.
- SS je sekunda.
- NNN je náhodně vygenerované nezáporné celé číslo od 0 do 999. To zajistí, že na serveru může teoreticky probíhat až 1000 registrací za sekundu. Přičemž pravděpodobnost kolize každé registrace  $r_i$  je  $p(r_i) = \frac{1}{1000}$ , kde  $i \in \{x \mid x \in \langle 0, 999 \rangle \wedge x \in \mathbb{Z}\}$ .

Pomocí instance třídy SSHCommandByKey je na SSHD serveru vykonán příkaz `sudo /usr/bin/pridejUzivatele.sh RRRR.MM.DD.HH.SS.NNN`. Účet registrace je obyčejný unixový účet s jedinou výjimkou. Pomocí `visudo` byl zeditován soubor `/etc/sudoers`, který umožní účtu registrace na serveru vykonat příkaz `sudo /usr/bin/pridejUzivatele.sh RRRR.MM.DD.HH.SS.NNN` s oprávněními super uživatele. Nic víc kromě přidání uživatele s oprávněními super uživatele provést nemůže.

```
final SSHCommandByKey sshcommandbykey = new
SSHCommandByKey(soukromyRegKlicDesif.getAbsolutePath(), "sudo
/usr/bin/pridejUzivatele.sh " + vp.getParametr(),
gn.getIPAdresaServeru(), gn.getUcetRegistrace());
```

Server vytvoří nového uživatele, nastaví vše potřebné a do souboru RRRR.MM.DD.HH.SS.NNN uloží uživatelské jméno a heslo.

Metoda provedRegistraci() stáhne v dalším kroku soubor RRRR.MM.DD.HH.SS.NNN ze serveru do dočasného souboru, který uloží do servisního adresáře aplikace jménem .m25o.

Učet registrace není v tuto chvíli již potřeba, proto je okamžitě zašifrován jeho soukromý klíč.

Pomocí SSHCommand vykoná příkaz /usr/bin/vygenerujKliceRSAPoprve.sh. Použije se přitom uživatelské jméno a heslo ze souboru RRRR.MM.DD.HH.SS.NNN. Skript vygenerujKliceRSAPoprve.sh na serveru vygeneruje pár RSA klíčů uživatele. Následně přidá veřejný klíč uživatele do souboru authorized\_keys. Tím je zajištěno, že se uživatel může hlásit na SSHD server pomocí svého veřejného klíče. Požadovaná délka klíčů je 2048 bitů. Skript může vypadat například takto:

```
#!/bin/bash
UZIVATEL=$(whoami)
mkdir /home/$UZIVATEL/.ssh
ssh-keygen -t rsa -b 2048 -N "" -f /home/$UZIVATEL/.ssh/klicRSA > /dev/null
cat /home/$UZIVATEL/.ssh/klicRSA.pub > /home/$UZIVATEL/.ssh/authorized_keys
```

Je smazán dočasný soubor RRRR.MM.DD.HH.SS.NNN, protože již není potřeba.

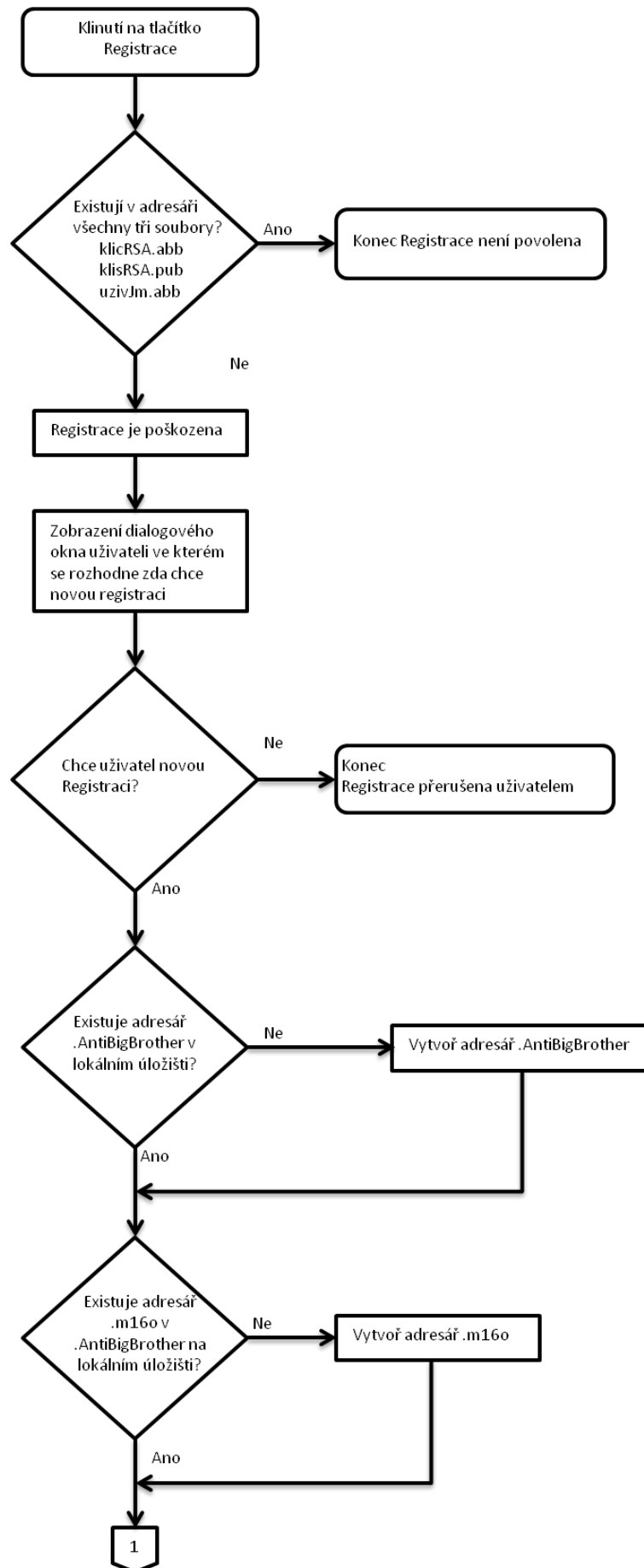
Metoda provedRegistraci() spustí SSHGet sshgetKlicRSA a stáhne soukromý klíč uživatele ze serveru, pak spustí SSHGet sshgetKlicRSAPub a stáhne veřejný klíč uživatele ze serveru do adresáře .AntiBigBrother/.m160.

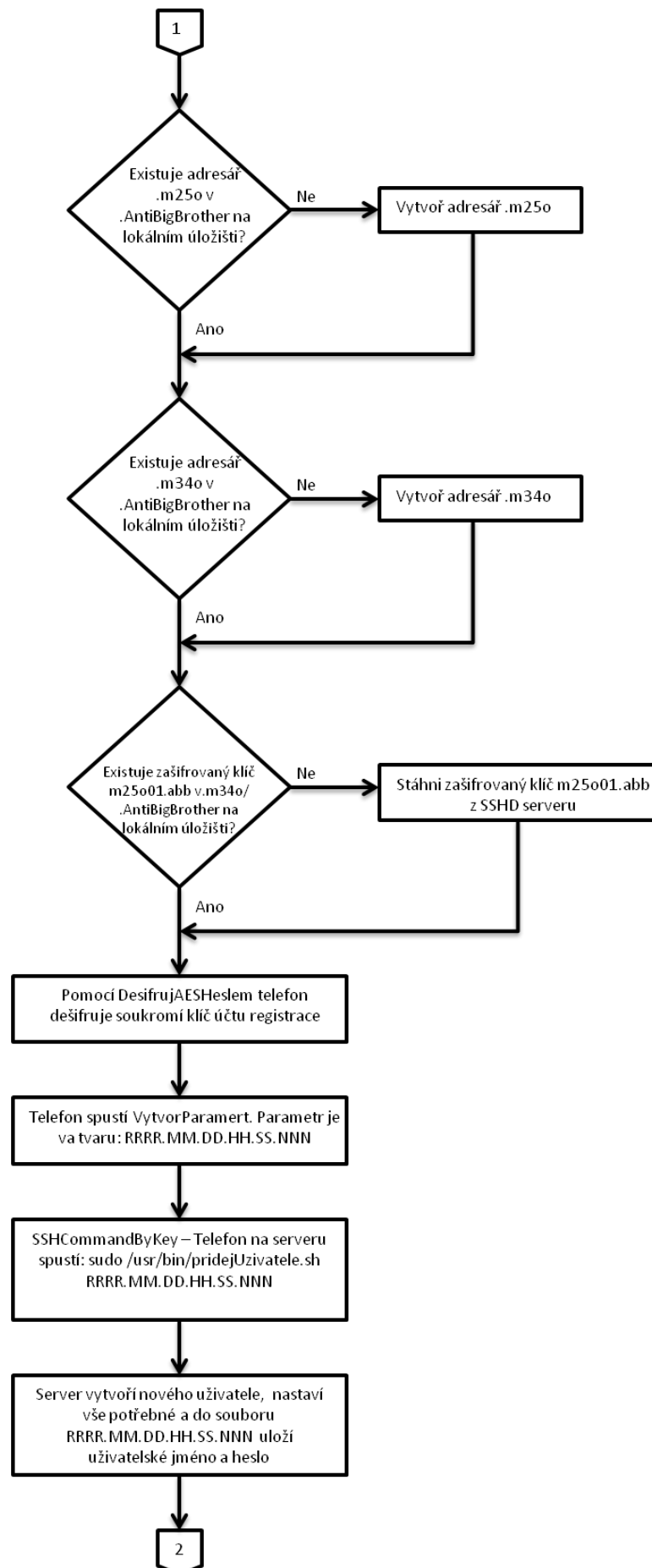
Následně je zašifrován soukromý i veřejný klíč uživatele.

Dále je vytvořen soubor m16o03, do kterého je uloženo uživatelské jméno, a je zašifrován.

Pokud dospěl proces registrace až sem, je uživateli zobrazena zpráva, že registrace proběhla úspěšně.

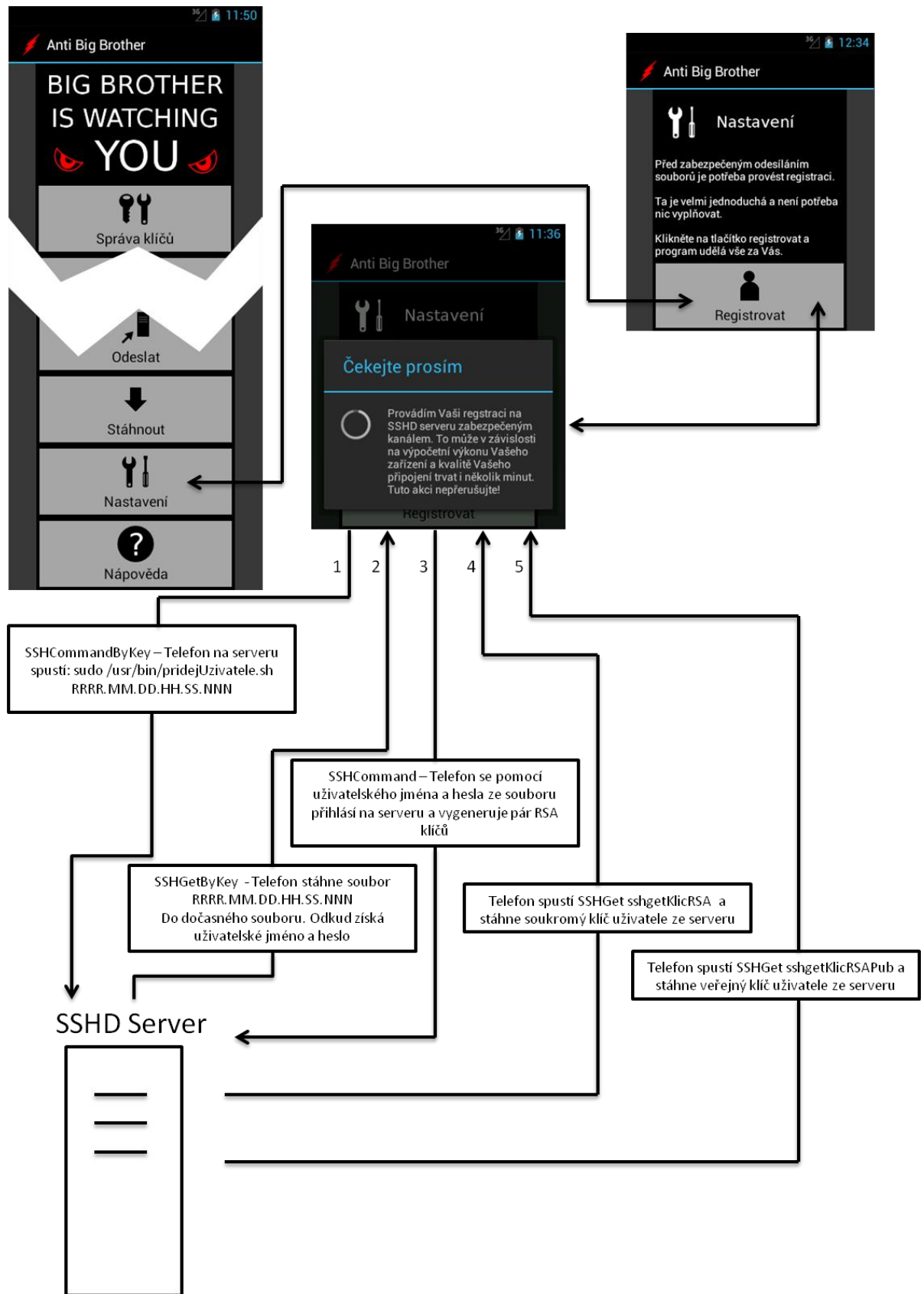
Pro lepší přehlednost je na obrázku 88 zachycena jen samotná komunikace mezi serverem a mobilním zařízením během procesu registrace.





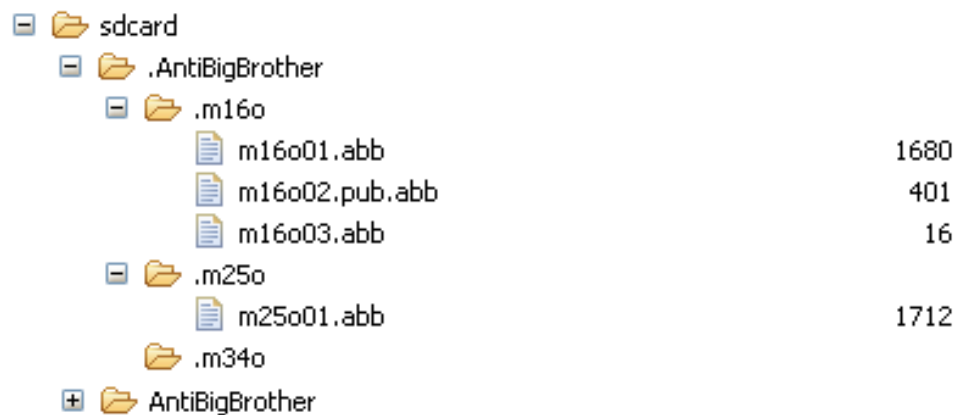


Obr. 87 Celý proces registrace  
[zdroj vlastní]



Obr. 88 Komunikace mezi serverem a mobilním zařízením během procesu registrace [zdroj vlastní]

Pokud je registrace úspěšná, je v mobilním zařízení struktura souborů a adresářů, tak jak je zachycena na obrázku 89.



Obr. 89 Struktura souborů a adresářů po registraci [zdroj vlastní]

## 5.5 Odesílání souborů na SSHD server

Odesílání souborů na SSHD server začíná kliknutím uživatele na tlačítko Odeslat na hlavní obrazovce aplikace. Tím je spuštěna aktivita reprezentovaná třídou OdeslatZasifrovaneSoubory a odeslatzasifrovanesoubory.xml. Třída OdeslatZasifrovaneSoubory je implementací komponenty pro grafickou práci se soubory, která je přizpůsobena pro vícenásobný výběr zašifrovaných souborů, určených k odeslání na SSHD server. Příkazová nabídka umožňuje:

- Odeslat vybrané soubory na server.
- Přesun o úroveň výš v adresářové struktuře.
- Vícenásobný výběr souborů.

Po označení souborů určených k odeslání a kliknutí na položku Odeslat soubory na server se hlavní vlákno rozdělí na vlákno, které zajistí odeslání vybraných souborů na SSHD server, a vlákno, které zobrazuje ProgressDialog uživateli. V posledním kroku se zruší označení souborů, které byly vybrány k odeslání. Je zaktualizován pohled na aktuální adresář.

Samotné odesílací vlákno může být řešeno například takto:

Vytvoří se seznam souboru k zazipování jménem seznamSouboryKZazipovani, do kterého se přidají soubory k zazipování. Postupně je v cyklu procházen seznam seznamPolozek, a pokud je na stejném místě v seznamu oznaceneSoubory hodnota "vybran", je vybraný soubor přidán do seznamu seznamSouboryKZazipovani:

```

final List<String> seznamSouboryKZazipovani = new ArrayList<String>();

for(int i = 0; i < seznamPolozek.size(); i++)
{
    final File soubor = new File(seznamPolozek.get(i));

    if(oznaceneSoubory.get(i).equals("vybran"))
    {
        seznamSouboryKZazipovani.add(soubor.getAbsolutePath());
    }
}

```

V dalším kroku je provedena kontrola souborového systému, jak bylo popsáno výše. Je důležité, aby byla kontrola souborového systému na tomto místě provedena. Pokud by souborový systém aplikace nebyl v pořádku, mohlo by to vážně poškodit proces odesílání vybraných souborů na server. Je důležité, aby byla tato část v bloku try: Instance třídy VytvorParametr vytvoří parametr ve tvaru RRRR.MM.DD.HH.SS.NNN. Zip archiv bude mít jméno archiv\_ RRRR.MM.DD.HH.SS.NNN.zip a bude uložen v dočasném adresáři .m34o. Instance třídy Zazipuj provede zazipování vybraných souborů. Konstruktor může mít například tento tvar:

```

public          Zazipuj(String[]          souboryKZazipovani,          String
cestaKamZipArchivUlozit)

```

Samotné zazipování může vypadat například takto:

```

try
{
    byte[] buffer = new byte[1024];

    FileOutputStream fos = new FileOutputStream(cestaKamZipAchrUloz);
    ZipOutputStream zos = new ZipOutputStream(fos);

    for(int j=0; j < soubKZazip.length; j++)
    {
        FileInputStream fin = new FileInputStream(soubKZazip[j]);

        File novyZipSoubor = new File(soubKZazip[j]);

        zos.putNextEntry(new ZipEntry(novyZipSoubor.getName()));

        int delka;

        while((delka = fin.read(buffer)) > 0)
        {
            zos.write(buffer, 0, delka);
        }

        zos.closeEntry();

        fin.close();
    }

    zos.close();
}

```

```
        fos.flush();
        fos.close();
    }
    catch (Exception e)
    {
        // Ošetření vyjímky
    }
}
```

V dalším kroku je rozšifrován soukromý serverový klíč uživatele, který je uložen na lokálním úložišti v `.../.AntiBigBrother/.m160/m16o01.abb`.

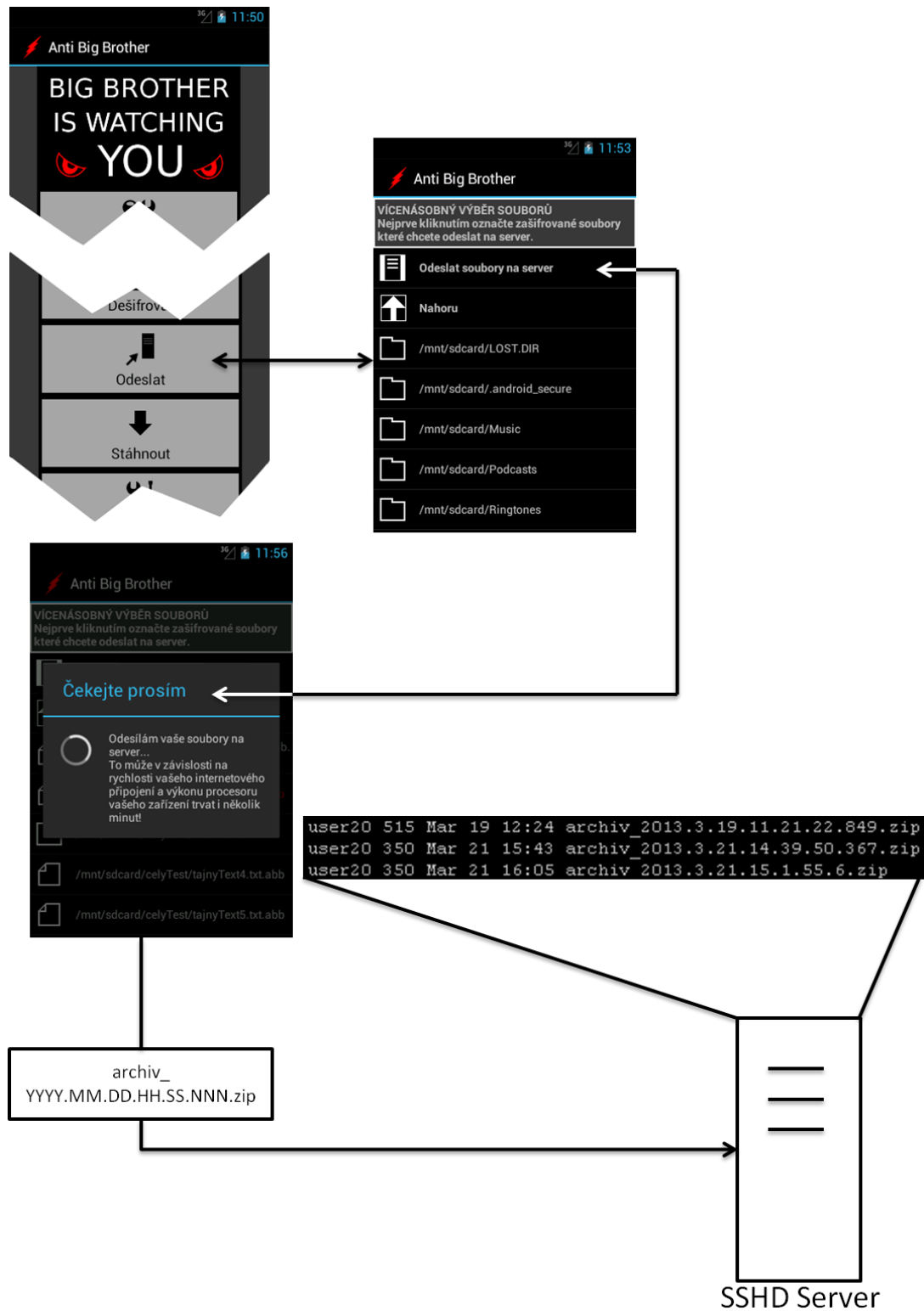
Pak je rozšifrováno serverové jméno uživatele, které je uloženo na lokálním úložišti v `.../.AntiBigBrother/.m160/m16o03.abb`.

Z rozšifrovaného souboru `m16o03.abb` je přečteno uživatelské jméno.

Pomocí `SSHPutByKey` jsou odeslány zazipované soubory na `SSHD` server.

V následujícím kroku je proveden úklid:

- Je zašifrován soukromý serverový klíč uživatele `m16o01.abb`.
- Je zašifrováno jméno uživatele uložené v souboru `m16o03.abb`.
- Je smazán zip archiv `archiv_RRRR.MM.DD.HH.SS.NNN.zip` z dočasného adresáře `.m34o`.



Obr. 90 Vytvoření a odeslání archiv\_RRRR.MM.DD.HH.SS.NNN.zip na server [zdroj vlastní]

## 5.6 Stahování souborů z SSHD serveru

Stahování souborů z SSHD serveru začíná kliknutím uživatele na tlačítko Stáhnout na hlavní obrazovce aplikace. Tím je spuštěna aktivita reprezentovaná třídou `StahnouZasifrovaneSoubory` a `stahnouzasifrovanesoubory.xml`. Třída `StahnouZasifrovaneSoubory` je implementací komponenty pro grafickou práci se soubory, která je přizpůsobená pro zobrazování, mazání a stahování souborů ze vzdáleného SSHD serveru. Příkazová nabídka umožňuje:

- Stáhnout soubory ze vzdáleného serveru.
- Smazat soubory ze vzdáleného serveru.
- Vícenásobný výběr souborů na vzdáleném serveru.

### 5.6.1 Zobrazování souborů na vzdáleném serveru

Při vytváření třídy `StahnouZasifrovaneSoubory` se zavolá metoda `onCreate`. Zde je hlavní vlákno aplikace rozděleno na vlákno, které provede výpis domovského adresáře uživatele ze vzdáleného SSHD serveru, a na vlákno, které uživateli zobrazuje `ProgressDialog`.

Vlákno výpisu zavolá metodu `vypisZeServeru()`. Metoda provede výpis domovského adresáře uživatele na vzdáleném SSHD serveru a výsledek zobrazí v grafické komponentě, která je upravena pro zobrazení souborů a adresářů ze vzdáleného serveru. Výpis může vypadat například takto.

V bloku `try` se provede:

```
try
{
    GlobalniNastaveni gn = new GlobalniNastaveni();

    String soukKlicUzivateleNaSrv = gn.getCestaDoAdresareAplikace() +
"/" + gn.adresarProKlice + "/" + gn.getSoukromyKlicUzivatele();

    DesifrujAESHeslem dah = new DesifrujAESHeslem(new
File(soukKlicUzivateleNaSrv), gn.getAppHeslo());

    dah.desifruj();

    String souborUzivatelskeJmenoNaSRV =
gn.getCestaDoAdresareAplikace() + "/" + gn.adresarProKlice + "/" +
gn.getUzivatelskeJmenoNaSRV();

    DesifrujAESHeslem dahUzivJm = new DesifrujAESHeslem(new
File(souborUzivatelskeJmenoNaSRV), gn.getAppHeslo());

    dahUzivJm.desifruj();
}
```

```

    Log.d(TAG, "dahUzivJm.getHlaska() : " + dahUzivJm.getHlaska());

    BufferedReader br = new BufferedReader(new
FileReader(souborUzivatelskeJmenoNaSRV.substring(0,
souborUzivatelskeJmenoNaSRV.length() - 4)));

    String uzivJmNaSRV = br.readLine();

    uzivJmNaSRV = uzivJmNaSRV.trim();

    br.close();

    SSHCommandByKeyGetResult scbkgr = new
SSHCommandByKeyGetResult(soukKlicUzivateleNaSrv.substring(0,
soukKlicUzivateleNaSrv.length() - 4), "ls", gn.getIPAdresaServeru(),
uzivJmNaSRV);

    vysledekVzdalenehoPrikazu = scbkgr.getStavPrikazu();

    vysledekVzdalenehoPrikazu =
vysledekVzdalenehoPrikazu.replace(".zip", ".zip ");

    ZasifrujAESHeslem zah = new ZasifrujAESHeslem(new
File(soukKlicUzivateleNaSrv.substring(0, soukKlicUzivateleNaSrv.length()
- 4)), gn.getAppHeslo());

    zah.zasifruj();

    ZasifrujAESHeslem zahUzivJm = new ZasifrujAESHeslem(new
File(souborUzivatelskeJmenoNaSRV.substring(0,
soukKlicUzivateleNaSrv.length() - 4)), gn.getAppHeslo());

    zahUzivJm.zasifruj();
}
catch (Exception e)
{
    Log.d(TAG, "Chyba : " + e.toString());
}

```

V dalším kroku je rozšifrován soukromý serverový klíč uživatele, který je uložen na lokálním úložišti v `.../.AntiBigBrother/.m160/m16o01.abb`.

Pak je rozšifrováno serverové jméno uživatele, které je uloženo na lokálním úložišti v `.../.AntiBigBrother/.m160/m16o03.abb`.

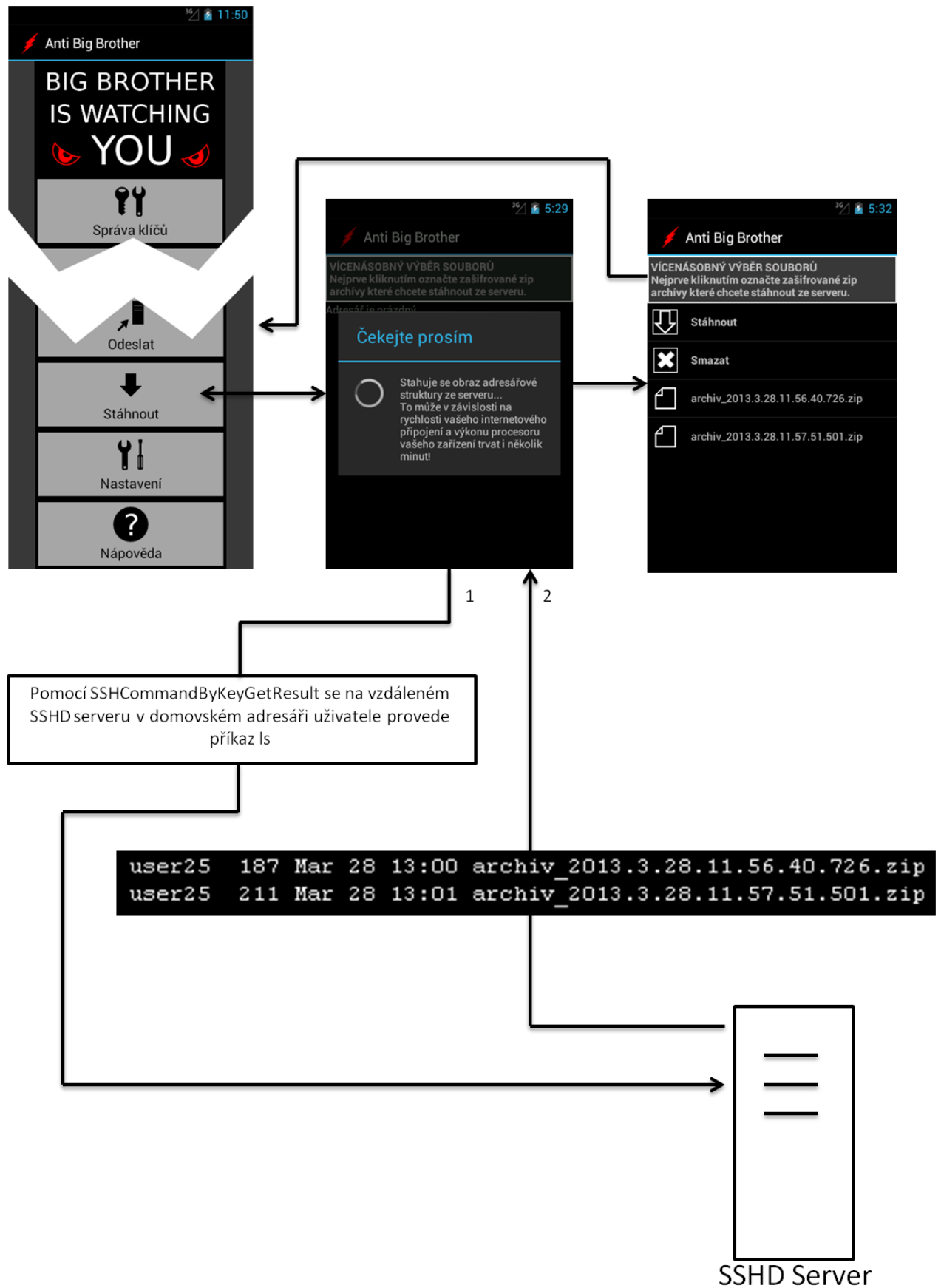
Z rozšifrovaného souboru `m16o03.abb` je přečteno uživatelské jméno.

Pomocí třídy `SSHCommandByKeyGetResult` se na vzdáleném SSHD serveru v domovském adresáři uživatele provede příkaz `ls`. Výsledek příkazu je získán pomocí metody `getStavPrikazu()` a je použit v komponentě pro grafickou práci se soubory. Zde jsou z něho vytvořeny seznamy `seznamPolozek` a `adresareASouboryAktualnihoAdresare` sloužící k zobrazování souborů a adresářů, jak bylo popsáno v kapitole 5.

Je zašifrován soukromý serverový klíč uživatele m16o01.abb.

Je zašifrováno jméno uživatele uložené v souboru m16o03.abb.

Nakonec je zavolána metoda vratSoubory(). Metody zobrazí uživateli pohled na strukturu souborů a adresářů v domovském adresáři uživatele na vzdáleném SSHD serveru.



Obr. 91 Výpis souborů a adresářů ze vzdáleného serveru [zdroj vlastní]

### 5.6.2 Stahování souborů zip ze vzdáleného serveru

Po označení zip souborů určených ke stažení a kliknutí na položku Stáhnout se hlavní vlákno rozdělí na vlákno, které zajistí stahování vybraných souborů z SSHD serveru, a vlákno, které zobrazuje ProgressDialog uživateli. V posledním kroku se zruší označení souborů, které byly vybrány ke stažení. Je zaktualizován pohled na domovský adresář uživatele na SSHD SERVERU.

Samotné stahovací vlákno může být řešeno například takto:

Vytvoří se seznam zip souborů určených ke stažení ze serveru jménem seznamZipArchivuKeStazeni. Postupně je v cyklu procházen seznam seznamPolozek, a pokud je na stejném místě v seznamu v oznaceneSoubory hodnota "vybran", je vybraný soubor zip přidán do seznamu seznamZipArchivuKeStazeni. Po dokončení seznamu seznamZipArchivuKeStazeni je zavolána metoda stahniZipArchivy. Ta může být realizována například takto:

V bloku try se provede kontrola, zda seznam seznamZipArchivuKeStazeni není prázdný. Pokud není prázdný, pokračuje se dál.

Je rozšifrován soukromý serverový klíč uživatele, který je uložen na lokálním úložišti v .../.AntiBigBrother/.m160/m16o01.abb.

Pak je rozšifrováno serverové jméno uživatele, které je uloženo na lokálním úložišti v .../.AntiBigBrother/.m160/m16o03.abb.

Z rozšifrovaného souboru m16o03.abb je přečteno uživatelské jméno.

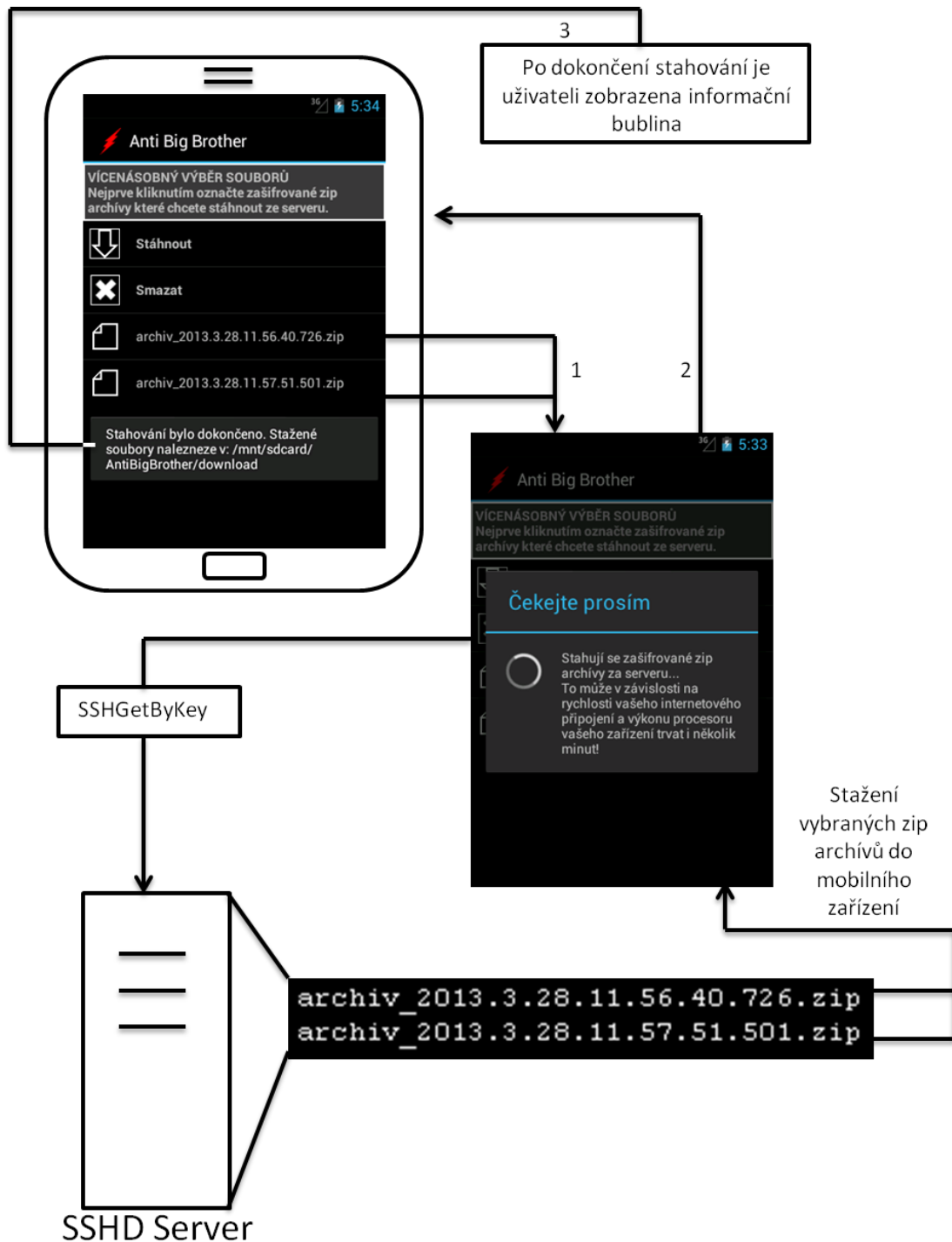
V cyklu se pomocí SSHGetByKey stahují zip soubory vybrané uživatelem ke stažení. Lze to provést například:

```
for(int i = 0; i < seznamZipArch.size(); i++)
{
    SSHGetByKey                sshgetbykey                =                new
SSHGetByKey(soukKlicUzivateleNaSrv.substring(0,
soukKlicUzivateleNaSrv.length() - 4), "/home/" + uzivJmNaSRV + "/" +
seznamZipArch.get(i), gn.getCestaDoUzivatelskehoAdresareAplikace() + "/"
+ gn.getStazenoVUzivatelskemAdresari() + "/" + seznamZipArch.get(i),
gn.getIPAdresaServeru(), uzivJmNaSRV);
}
```

Zip soubory jsou ukládány na lokální úložiště do adresáře AntiBigBrother/download. Uživatel je informován o dokončení stahování a o tom, že stažené zip soubory nalezne v adresáři AntiBigBrother/download na lokálním úložišti. Jak je vidět na obrázku 92.

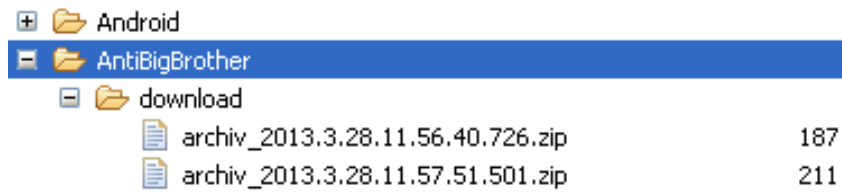
Je zašifrován soukromý serverový klíč uživatele m16o01.abb.

Je zašifrováno uživatele uložené v souboru m16o03.abb.



Obr. 92 Stahování zip archívů ze vzdáleného serveru [zdroj vlastní]

Proběhlo-li vše v pořádku, je možné vidět pomocí komponenty File Explorer ve vývojovém prostředí Eclipse, že zip soubory byly staženy do mobilního zařízení. Situace je zachycena na obrázku 93.



Obr. 93 Stažené zip soubory ze serveru do mobilního zařízení  
[zdroj vlastní]

### 5.6.3 Mazání souborů zip ze vzdáleného serveru

Po označení zip souborů určených ke smazání a po kliknutí na položku Smazat se hlavní vlákno rozdělí na vlákno, které zajistí mazání vybraných zip souborů z SSHD serveru, a na vlákno které zobrazuje ProgressDialog uživateli. V posledním kroku se zruší označení souborů, které byly vybrány ke smazání. Je zaktualizován pohled na domovský adresář uživatele na SSHD serveru. Tím jsou z pohledu na domovský adresář uživatele odstraněny smazané zip soubory.

Samotné mazací vlákno může být řešeno například takto:

Vytvoří se seznam zip souborů určených ke smazání ze serveru jménem seznamZipArchivuKeSmazani. Postupně je v cyklu procházen seznam seznamPolozek, a pokud je na stejném místě v seznamu v oznaceneSoubory hodnota "vybran", je vybraný zip soubor přidán do seznamu seznamZipArchivuKeSmazani. Po dokončení seznamu seznamZipArchivuKeSmazani je zavolána metoda smazZipArchivy. Ta může být realizována například takto:

V bloku try se provede kontrola, zda seznam seznamZipArchivuKeSmazani není prázdný. Pokud není prázdný, pokračuje se dál.

Je rozšifrován soukromý serverový klíč uživatele, který je uložen na lokálním úložišti v .../.AntiBigBrother/.m160/m16o01.abb.

Pak je rozšifrováno serverové jméno uživatele, které je uloženo na lokálním úložišti v .../.AntiBigBrother/.m160/m16o03.abb.

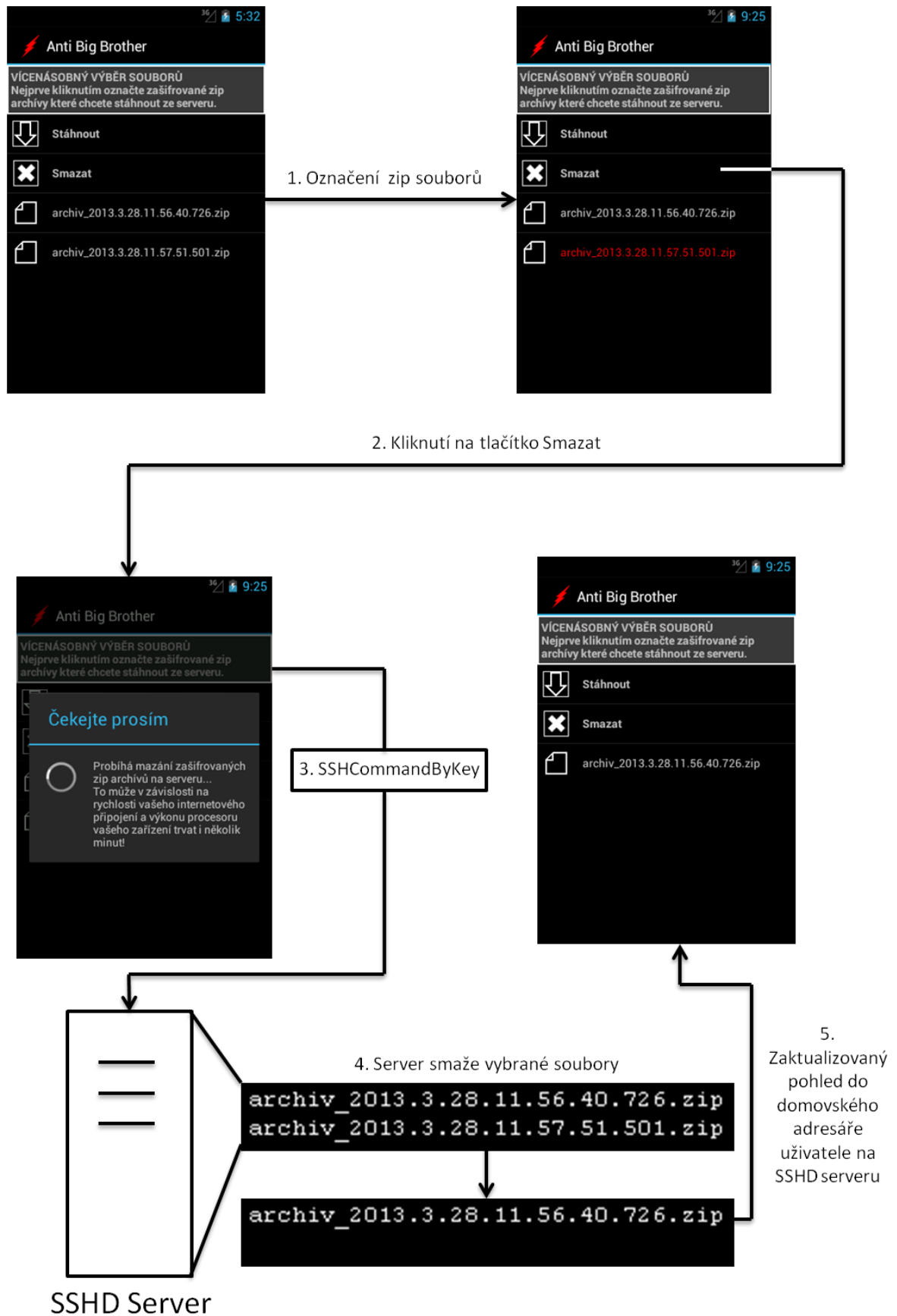
Z rozšifrovaného souboru m16o03.abb je přečteno uživatelské jméno.

V cyklu se pomocí SSHCommandByKey mažou zip soubory vybrané uživatelem ke smazání. Například:

```
for(int i = 0; i < seznamZipArch.size(); i++)
{
    SSHCommandByKey          sshcommandbykey          =          new
SSHCommandByKey(soukKlicUzivateleNaSrv.substring(0,
soukKlicUzivateleNaSrv.length() - 4), "rm -f " + seznamZipArch.get(i),
gn.getIPAdresaServeru(), uzivJmNaSRV);
}
```

Je zašifrován soukromý serverový klíč uživatele m16o01.abb.

Je zašifrováno jméno uživatele uložené v souboru m16o03.abb.



Obr. 94 Mazání zip archivů ze vzdáleného serveru [zdroj vlastní]

## ZÁVĚR

V teoretickém základu diplomové práce byly na základě zahraniční a domácí literatury vypočítány příklady symetrického a asymetrického šifrování a dešifrování. Výpočty příkladů jsou kompletně provedeny od začátku do konce. To znamená, že na začátku je otevřená zpráva, následně je vypočítána její zašifrovaná podoba a nakonec je vypočítána dešifrovaná podoba. V práci jsou vysvětleny matematické základy útoku na RSA, které mohou být úspěšné vlivem slabého klíče nebo nedokonalostí implementace RSA.

Vývoj mobilních aplikací je novou dynamickou vyvíjející se oblastí, což na sebe váže velké množství nevyřešených problémů. Neexistuje žádná literatura, která by se specializovala na specifika kryptografie na mobilní platformě Android. Při řešení kryptografických problémů se autor diplomové práce výhradně opíral o zahraniční literaturu věnující se kryptografii na tzv. velké (desktopové) Javě. Ta byla upravována podle profesionální programátorské „vědomostní báze“ [14] a na vlastních příkladech byly převedeny na funkční podobu symetrické i asymetrické kryptografie na mobilní platformě Android.

Během šifrování větších souborů jsem narazil na problém malé operační paměti některých mobilních zařízení s operačním systémem Android. Konkrétně šlo o chybu `OutOfMemoryError`. Operační systém Android přiděluje aplikacím přesně daný prostor. Velikost přiděleného prostoru závisí na fyzické velikosti operační paměti daného zařízení. Je tedy u každého zařízení jiná. Pokud aplikace požaduje více prostoru než má přiděleno a pokusí se obsadit další paměť. Je aplikace operačním systémem okamžitě ukončena. Zatímco u desktopu se odstránkovávají nepoužívané části operační paměti do stránkovacího souboru, a tak podobné riziko nehrozí. Problém nebyl řešen v žádné odborné literatuře a upravená řešení ze `stackoverflow` nebyla uspokojivá, protože se v nich vyskytovaly problémy typu `NullPointerException`, jindy problémy s paddingem při dešifrování. Nebo se dokonce při procesu šifrování a následném dešifrování ztratilo pár bajtů. Proto jsem provedl svůj vlastní výzkum, který byl velmi zdlouhavý. Na jeho konci je robustní řešení, pomocí něhož mohou správně šifrovat i dešifrovat všechny typy mobilních zařízení bez ohledu na velikost šifrovaných/dešifrovaných souborů či velikosti operační paměti.

V práci byla objasněna problematika vývoje aplikací pro telefony/tablety s operačním systémem Android v prostředí Eclipse s nainstalovaným ADT pluginem. Včetně testování

vyvíjených aplikací, jak prostřednictvím Android Virtual Device Manageru, tak pomocí fyzického telefonu. Práce odpověděla na otázku, jak při testování aplikací hledat chyby pomocí LogCat. Pomocí sledování systémových zpráv dané úrovně výpisu (verbose, debug, info, warn, error a assert), i pomocí sledování zpráv, které si programátor sám vytvoří.

Diplomová práce objasnila problematiku GUI návrhu pomocí jazyka XML včetně problémů, které jsou s návrhem GUI pomocí LinearLayout, RelativeLayout a TableLayout. V diplomové práci jsem vyřešil nejčastější problém GUI návrhu. Návrh vypadá dobře na 2,5 palcovém telefonu shodně jako na 10,1 palcovém tabletu. Vytvořil jsem styl GUI návrhu, který jsem nazval Streak style. Streak style si nevystačí s jedinou layout třídou, ale je kombinací tříd LinearLayout a RelativeLayout a techniky pevného grafického prvku.

Firma Google vyvíjející a udržující operační systém Android má filozofii založenou na představě, že uživatel by neměl mít svá data uložená lokálně ve svém zařízení. Uživatel by je měl mít uloženy v síťových službách Google na serverech společnosti. Uvedenou skutečností lze vysvětlit, proč v operačním systému Android neexistují standardizované grafické komponenty pro výběr souborů a adresářů.

Jedná se o velký rozdíl oproti jazykům, jako je například C#, kde má programátor k dispozici komponenty OpenFileDialog, SaveFileDialog, FolderBrowserDialog a další. Práce přichází s návrhem univerzální komponenty pro grafickou práci se soubory, kterou lze snadno upravovat pro nejrůznější účely.

Pro skutečně bezpečný přenos dat přes nedůvěryhodnou síť je potřeba použít odpovídající prostředky, které mají vysoký standard bezpečnosti. Technologie splňující všechny bezpečnostní požadavky se jmenuje SSH2. SSH2 je v dnešní době synonymem bezpečí. Ve světě Javy existuje čistá implementace SSH2, která se nazývá Java Secure Channel (JSCH). JSCH umožňuje připojení k SSHD server, použít port forwarding, X11 forwarding, přenos souborů, atd. JSCH dovoluje programátorům integrovat funkcionalitu SSH2 do svých vlastních programů napsaných v jazyce Java. JSCH lze volně používat. Pro přehlednost výsledného kódu a jeho neopakování se, kdykoliv bude potřeba použít funkce JSCH, autor práce zapouzdřil funkcionalitu JSCH do specializovaných tříd použitelných i pro jiné projekty.

SSHGet – pomocí uživatelského jména a hesla stáhne z SSHD serveru požadovaný soubor a uloží ho v lokálním úložišti telefonu/tabletu.

SSHGetByKey – pomocí uživatelského jména a klíče stáhne s SSHD serveru požadovaný soubor a uloží ho v lokálním úložišti telefonu/tabletu.

SSHPut – pomocí uživatelského jména a hesla se na SSHD server nahraje požadovaný soubor.

SSHPutByKey - pomocí uživatelského jména a klíče se na SSHD server nahraje požadovaný soubor.

SSHCommand – vytvoří ssh spojení se serverem pomocí uživatelského jména a hesla. Na SSHD serveru provede příkaz specifikovaný v konstruktoru. Je důležité, aby měl uživatel, pod jehož serverovým účtem jsou prováděny příkazy odpovídající úroveň oprávnění.

SSHCommandByKey – vytvoří ssh spojení se serverem pomocí uživatelského jména a klíče. Na SSHD serveru provede příkaz specifikovaný v konstruktoru. Je důležité, aby měl uživatel, pod jehož serverovým účtem jsou prováděny příkazy odpovídající úroveň oprávnění.

V diplomové práci jsem vytvořil vlastní návrh přenosu šifrovaných souborů mezi mobilními zařízeními s operačním systémem Android. Návrh využívá interaktivní spolupráce SSHD serveru s mobilními zařízeními.

Výsledky ročního zpracování diplomové práce budou prezentovány na 4. ročníku mezinárodní konference Bezpečnostní management a společnost pořádanou Univerzitou obrany v rámci CATE 2013 (Community-Army-Technology\_Environment) ve dnech 22. – 23. května 2013 a publikovány ve sborníku z konference.

## ZÁVĚR V ANGLIČTINĚ

The theoretical basis for this thesis was based on foreign and domestic literature calculated examples of symmetric and asymmetric encryption and decryption. Examples of calculations are made completely from beginning to end. This means that at the beginning of the open message is subsequently calculated the encrypted form and finally calculated decrypted form. The paper explains the mathematical foundations of the attack on RSA, which can be successful due to weak keys or imperfections implementation of RSA.

Development of mobile applications is a new dynamic developing area, which binds to a large number of unsolved problems. There is no literature that would specialize in specific cryptography for mobile platform Android. When solving cryptographic problems, the author of the thesis was based solely on foreign literature on cryptography for desktop Java. It was edited by a professional programmer "knowledge base" [14] and on its own examples were translated into the functional form of symmetric and asymmetric cryptography for mobile platform Android.

While encrypting large files I came across a small memory problem of some mobile devices running Android. Specifically, it was a mistake `OutOfMemoryError`. The Android operating system allocates the exact space applications. Size of the allocated space depends on the physical memory size of the device. It is therefore different for each device. If application requires more space than is allocated and attempts to occupy memory more. The application operating system is immediately terminated. Meanwhile at the desktop `ForceNewPage` unused portion of memory to the paging file, and so no such risk. The problem was not addressed in any of the literature and adapted solutions from `stackoverflow` was not satisfactory, because in these type of problems `NullPointerException`, sometimes problems with padding when decrypting absent. Or even in the process of encryption and subsequent decryption lost a few bytes. Therefore, I conducted my own research, which was very tedious. At the end is a robust solution with which they can properly encrypt and decrypt all types of mobile devices, regardless of the size of encrypted / decrypted files or memory.

The work was to clarify the issue of application development for phones / tablets running Android in Eclipse with ADT plugin installed. Including the testing of applications developed, both through Android Virtual Device Manager and using the physical phone. Work answered the question of how to test applications find fault with `logC`. By tracking

system reports the level dump (verbose, debug, info, warn, error and assert) and by monitoring the messages that the programmer creates himself.

The thesis clarifies the issue of GUI design using XML, including problems associated with design GUI using LinearLayout, RelativeLayout and TableLayout. In my work I solved the most common problem GUI design. The proposal looks good on the 2.5-inch phone, same as the 10.1-inch tablet. I created a style GUI design, which I called Streak style. Streak style can not enough with a single class layout, but a combination of classes LinearLayout and RelativeLayout and technology firm graphic element.

Google developing and maintaining operating system Android has a philosophy based on the idea that the user should not have their data stored locally on your device. User should be stored in the network services on Google's servers. This fact may explain why the Android operating system there are no standardized graphical components for the selection of files and directories.

This is a big difference from languages such as C #, where the programmer available components OpenFileDialog, SaveFileDialog, FolderBrowserDialog and more. The thesis concludes with a proposal for a universal component for graphical work with files that can be easily modified for a variety of purposes.

For really secure transmission of data over an untrusted network is necessary to use appropriate means to have a high standard of safety. Technology meets all safety requirements called SSH2. SSH2 is nowadays synonymous with safety. In the world there is a pure Java implementation of SSH2, called the Java Secure Channel (JSCH). Jsch allows you to connect to the SSHD server, use port forwarding, X11 forwarding, file transfer, etc. JSCH allows programmers to integrate SSH2 functionality into their own programs written in Java. JSCH can be used freely. For clarity, the resulting code and the non-recurrence whenever the need to use the JSCH, author of works JSCH encapsulate functionality into specialized classes applicable to other projects.

SSHGet - using the username and password from the SSHD server downloads the file and saves it in a local store, phone/tablet.

SSHGetByKey - using the username and key download SSHD server with the file and stores it in a local store, phone/tablet.

SSHPut - using the username and password in the SSHD server uploads the file.

SSHPutByKey - using the username and key to the SSHD server uploads the file.

SSHCommand - create a ssh connection to the server using the username and password. On the SSHD server executes the command specified in the constructor. It is important that the user account under which the server commands are executed appropriate permission level.

SSHCommandByKey - create a ssh connection to the server using the username and key. On the SSHD server executes the command specified in the constructor. It is important that the user account under which the server commands are executed appropriate permission level.

In my work I created a own design transfer encrypted files between mobile devices running Android. The proposal uses interactive collaboration SSHD server with mobile devices.

The results of the annual processing of the thesis will be presented at the 4<sup>th</sup> the International Conference of Security Management and Society organized by the University of Defence in the CATE 2013 (Community-Army-Technology-Environment) from 22 - 23<sup>rd</sup> May 2013 and published in the conference proceedings.

## SEZNAM POUŽITÉ LITERATURY

- [1] SINGH, Simon. *The Coode Book: The Secret History of Codes and Code-breaking Cryptography*. London: Fourt Estate, 2000. ISBN 9781857028898.
- [2] PIPER, Fred and Sean Murphy. *Cryptography: Very Short Introduction*. London: Oxford Univerzity Press, 2002, ISBN 9780192803153.
- [3] BEKER, Henry and Fred Piper. *Cipher Systems: The Protection of Communications*. London: Northwood Books, 1982.
- [4] CRYPTOGRAMS. F.A.Q. *Cryptograms.org* [online]. [cit. 2013-01-15]. Dostupné z: <http://www.cryptograms.org/letter-frequencies.php>
- [5] CENTRUM ZPRACOVÁNÍ PŘIROZENÉHO JAZYKA. Frekvence písmen, bigramů, trigramů, délka slov. *Nlp.fi.muni.cz* [online]. © 2012. [cit. 2013-01-14]. Dostupné z: [http://nlp.fi.muni.cz/cs/Frekvence\\_pismen\\_bigramu\\_trigramu\\_delka\\_slov](http://nlp.fi.muni.cz/cs/Frekvence_pismen_bigramu_trigramu_delka_slov)
- [6] CAMERON, David. David Cameron: Conservatives will battle for Britain's future. *Telegraph.co.uk*. [online]. © 2012 [cit. 2013-03-02]. Dostupné z: <http://www.telegraph.co.uk/news/politics/david-cameron/9904735/David-Cameron-Conservatives-will-battle-for-Britains-future.html>
- [7] MARTINEK, Pavel. *Základy teoretické informatiky*. Olomouc: Univerzita Palackého, 2006. Dostupné také z: <http://phoenix.inf.upol.cz/esf/ucebni/zti.pdf>
- [8] MOORE, Gordon. Gramming more components onto integrated circuits. *Electronics*. [online]. 1965, Volume 38, Number 8, [cit. 2013-03-02]. Dostupné z: [http://download.intel.com/museum/Moores\\_Law/ArticlesPress\\_releases/Gordon\\_Moore\\_1965\\_Article.pdf](http://download.intel.com/museum/Moores_Law/ArticlesPress_releases/Gordon_Moore_1965_Article.pdf)
- [9] RIVEST, R. L., A. SHAMIR AND L. ADLEMAN. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In: *Communications of the ACM*, 1978, **21** (2). pp. 120–126.
- [10] GROŠEK, Otokar a Štefan PORUBSKÝ. *Sifrovanie algoritmy metódy prax*. Praha: Grada, 1992. ISBN 9788085424621.
- [11] VELEBIL, Jiří. *Diskrétní matematika*. Praha: ČVUT. 2007. Dostupné také z: <http://math.feld.cvut.cz/pub/velebil/y01dma/dma-notes.pdf>.

- [12] DEVELOPERS. Developer Tools. *Developer.android.com* [online]. [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/tools/index.html>
- [13] HOOK, David. *Beginning Cryptography with Java*. USA (Indiana): Wrox, 2005. ISBN 978-0764596339.
- [14] STACKOVERFLOW. Welcome to Stack Overflow. *Stackoverflow.com* [online]. © 2012 [cit. 2013-01-16]. Dostupné z: <http://stackoverflow.com/about>
- [15] Java™ Platform Standard Ed. 6. Class SecretKeySpec. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/javax/crypto/spec/SecretKeySpec.html#SecretKeySpec\(byte \[\], int, int, java.lang.String\)](http://docs.oracle.com/javase/6/docs/api/javax/crypto/spec/SecretKeySpec.html#SecretKeySpec(byte[],int,int,java.lang.String))
- [16] Java™ Platform Standard Ed. 6. Java™ Cryptography Architecture Standard Algorithm Name Documentation Java™ Platform Standard Edition 6. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html#AlgorithmParameters>
- [17] Java™ Platform Standard Ed. 6. SecretKeySpec. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/javax/crypto/spec/SecretKeySpec.html#SecretKeySpec\(byte \[\], java.lang.String\)](http://docs.oracle.com/javase/6/docs/api/javax/crypto/spec/SecretKeySpec.html#SecretKeySpec(byte[],java.lang.String))
- [18] Java™ Platform Standard Ed. 6. getInstance. *Docs.oracle.com* [online]. © 2004-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/1.5.0/docs/api/javax/crypto/Cipher.html#getInstance\(java.lang.String,java.lang.String\)](http://docs.oracle.com/javase/1.5.0/docs/api/javax/crypto/Cipher.html#getInstance(java.lang.String,java.lang.String))
- [19] Java™ Platform Standard Ed. 6. Java™ Cryptography Architecture (JCA) Reference Guide. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#trans>
- [20] Java™ Platform Standard Ed. 6. Class Cipher. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html>

- [21] Java™ Platform Standard Ed. 6. Javax.crypto Class Cipher. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html>
- [22] Java™ Platform Standard Ed. 6. `init`. *Docs.oracle.com* [online]. © 2004-2010 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/1.5.0/docs/api/javax/crypto/Cipher.html#init\(int, java.security.Key\)](http://docs.oracle.com/javase/1.5.0/docs/api/javax/crypto/Cipher.html#init(int, java.security.Key))
- [23] Java™ Platform Standard Ed. 6. `public final void init`. *Docs.oracle.com* [online]. © 2009-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#init\(int, java.security.Key\)](http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#init(int, java.security.Key))
- [24] Java™ Platform Standard Ed. 6. `update`. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#update\(byte\[\], int, int, byte\[\]\)](http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#update(byte[], int, int, byte[]))
- [25] Java™ Platform Standard Ed. 6. `doFinal`. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#doFinal\(byte\[\], int\)](http://docs.oracle.com/javase/6/docs/api/javax/crypto/Cipher.html#doFinal(byte[], int))
- [26] Java™ 2 Platform St. Ed. V1.4.2. Class `MessageDigest`. *Docs.oracle.com* [online]. © 2003-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html#getInstance\(java.lang.String\)](http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html#getInstance(java.lang.String))
- [27] Java™ 2 Platform St. Ed. V1.4.2. `clone`. *Docs.oracle.com* [online]. © 2003-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html#getInstance\(java.lang.String, java.security.Provider\)](http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html#getInstance(java.lang.String, java.security.Provider))
- [28] Java™ Platform Standard Ed. 6. Class `KeyFactory`. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html>
- [29] Java™ Platform Standard Ed. 6. `generatePrivate`. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#generatePrivate\(java.security.spec.KeySpec\)](http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#generatePrivate(java.security.spec.KeySpec))

- [30] Java™ Platform Standard Ed. 6. generatePublic. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#generatePublic\(java.security.spec.KeySpec\)](http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#generatePublic(java.security.spec.KeySpec))
- [31] Java™ Platform Standard Ed. 6. translateKey. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#getKeySpec\(java.security.Key,java.lang.Class\)](http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#getKeySpec(java.security.Key,java.lang.Class))
- [32] Java™ Platform Standard Ed. 6. getKeySpec. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#getKeySpec\(java.security.Key,java.lang.Class\)](http://docs.oracle.com/javase/6/docs/api/java/security/KeyFactory.html#getKeySpec(java.security.Key,java.lang.Class))
- [33] Java™ Platform Standard Ed. 6. Class KeyPairGenerator. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: <http://docs.oracle.com/javase/6/docs/api/java/security/KeyPairGenerator.html>
- [34] Java™ Platform Standard Ed. 6. genKeyPair. *Docs.oracle.com* [online]. © 1993-2011 [cit. 2013-01-16]. Dostupné z: [http://docs.oracle.com/javase/6/docs/api/java/security/KeyPairGenerator.html#genKeyPair\(\)](http://docs.oracle.com/javase/6/docs/api/java/security/KeyPairGenerator.html#genKeyPair())
- [35] NVIDIA. Introducing NVIDIA® TEGRA® 4, the world's fastest mobile processor. *Nvidia.com* [online]. © 2013 [cit. 2013-01-16]. Dostupné z: <http://www.nvidia.com/object/tegra-4-processor.html>
- [36] DEVELOPERS. Honeycomb. *Developer.android.com* [online]. © 2013 [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/about/versions/android-3.0-highlights.html>
- [37] DEVELOPERS. Building Your First App. *Developer.android.com* [online]. © 2013 [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/training/basics/firstapp/index.html>
- [38] DEVELOPERS. Get the Android SDK. *Developer.android.com* [online]. © 2013 [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/sdk/index.html>

- [39] DEVELOPERS. ADT Plugin. *Developer.android.com* [online]. © 2013 [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/tools/sdk/eclipse-adt.html>
- [40] DEVELOPERS. Get the Android SDK – Download for other platforms. *Developer.android.com* [online]. © 2013 [cit. 2013-02-01]. Dostupné z: <http://developer.android.com/sdk/index.html>
- [41] STRATEGY ANALYTICS. Apple iPhone 5 Overtakes Samsung Galaxy S3 to Become World's Best-Selling Smartphone Model in Q4 2012. *Strategyanalysis.com* [online]. © 2013 [cit. 2013-03-01]. Dostupné z: <http://blogs.strategyanalytics.com/HCST/post/2013/02/20/Strategy-Analytics-Apple-iPhone-5-Becomes-Worlds-Best-Selling-Smartphone-Model-in-Q4-2012.aspx>
- [42] MURPHY, M. L. *Android 2: Průvodce programováním mobilních aplikací*. Brno: Computer Press, 2011. ISBN 978-80-251-3194-7.

## SEZNAM OBRÁZKŮ

Obr. 1 Historický vývoj utajené komunikace [1] .....	11
Obr. 2 Příklad šifrované abecedy [zdroj vlastní] .....	12
Obr. 3 Příklad šifrování posunutím písmene A [zdroj vlastní].....	13
Obr. 4 Příklad prvního způsobu pokládání písmen na koleje [zdroj vlastní] .....	14
Obr. 5 Příklad druhého způsobu pokládání písmen na koleje [zdroj vlastní].....	14
Obr. 6 Příklad kódování [zdroj vlastní] .....	14
Obr. 7 Část textu novinového článku [6].....	17
Obr. 8 Program pro frekvenční analýzu [zdroj vlastní] .....	18
Obr. 9 Histogram výsledků frekvenční analýzy [zdroj vlastní].....	21
Obr. 10 Schéma otevřené a šifrovací abecedy [zdroj vlastní] .....	22
Obr. 11 Ohraničení funkce gn shora [zdroj vlastní] .....	24
Obr. 12 Ohraničení funkce gn zdola [zdroj vlastní] .....	25
Obr. 13 Doba výpočtu kódu 1 [zdroj vlastní] .....	26
Obr. 14 Doba výpočtu kódu 2 [zdroj vlastní] .....	27
Obr. 15 Moderní kryptografie - typy šifer [zdroj vlastní].....	29
Obr. 16 Šifrování a dešifrování pomocí symetrického klíče [zdroj vlastní].....	30
Obr. 17 Šifrování pomocí Feistelovi šifry [zdroj vlastní].....	32
Obr. 18 Dešifrování pomocí Feistelovi šifry [zdroj vlastní].....	33
Obr. 19 Šifrování a dešifrování pomocí veřejného a soukromého klíče [zdroj vlastní].....	35
Obr. 20 Výpočet inverze čísla C [zdroj vlastní] .....	36
Obr. 21 Architektura JCE/JCA [autorem přeložen zdroj 13] .....	41
Obr. 22 Výjimka vyvolaná dlouhým klíčem [zdroj vlastní].....	43
Obr. 23 Vyjimka ShortBufferException [zdroj vlastní] .....	46
Obr. 24 Výsledek šifrování [zdroj vlastní] .....	47
Obr. 25 Dorovnávání bajtů [zdroj vlastní].....	47
Obr. 26 Doplnění chybějících bajtů [zdroj vlastní] .....	48
Obr. 27 Padding souboru, který má velikost rovnu n násobku velikosti bloku šifry [zdroj vlastní] .....	49
Obr. 28 Automatický padding [zdroj vlastní].....	51
Obr. 29 32 bitové pole, které obsahuje dva identické 16 bitové bloky [zdroj vlastní].....	52
Obr. 30 Opakování bajtů po osmy bajtech [zdroj vlastní].....	53
Obr. 31 Pole zašifrované pomocí AES a CBC bez opakování [zdroj vlastní].....	55

Obr. 32 Pole zašifrované pomocí DES a CBC bez opakování [zdroj vlastní].....	55
Obr. 33 Ukázka šifrování s heslem [zdroj vlastní] .....	57
Obr. 34 Zašifrovaný text, kryptogram a dešifrovaný text [zdroj vlastní] .....	59
Obr. 35 Ukázka RSA šifrování [zdroj vlastní] .....	62
Obr. 36 Ukázka RSA dešifrování [zdroj vlastní].....	63
Obr. 37 Soubor určený k posolení a sůl [zdroj vlastní] .....	64
Obr. 38 Osolení souboru [zdroj vlastní] .....	65
Obr. 39 Obsah odsoleného souboru [zdroj vlastní] .....	68
Obr. 40 Obsah osoleného souboru [zdroj vlastní] .....	68
Obr. 41 Soubor k posolení a sůl jsou různě velké [zdroj vlastní].....	69
Obr. 42 Posolení 50 B solí o velikosti 10 B po částech.....	70
Obr. 43 Lichý způsob solení [zdroj vlastní] .....	70
Obr. 44 Sudý způsob solení .....	71
Obr. 45 Program Android Virtual Device Manager [zdroj vlastní].....	74
Obr. 46 Vytváření virtuálního zařízení [zdroj vlastní] .....	75
Obr. 47 Přidělování operační paměti aplikacím [zdroj vlastní].....	76
Obr. 48 Pokus aplikace o použití další paměti [zdroj vlastní] .....	76
Obr. 49 Ukázka zašifrování souboru o velikosti 4410036 B [zdroj vlastní] .....	77
Obr. 50 Velikost souboru před a po zašifrování [zdroj vlastní].....	78
Obr. 51 Hlavička nezašifrovaného souboru 1.mp3 je čitelná [zdroj vlastní] .....	78
Obr. 52 Hlavička zašifrovaného souboru není čitelná [zdroj vlastní] .....	79
Obr. 53 Ukázka zašifrování souboru o velikosti 15932229 B [zdroj vlastní] .....	80
Obr. 54 Pád aplikace při šifrování souboru o velikosti 15932229 B [zdroj vlastní] .....	81
Obr. 55 Výpis z LogCat [zdroj vlastní] .....	81
Obr. 56 Soubor šifrovaný po částech [zdroj vlastní] .....	82
Obr. 57 Šifrování pomocí bufferu [zdroj vlastní].....	85
Obr. 58 Vývojové prostředí Eclipse s nainstalovaným ADT pluginem [zdroj vlastní].....	86
Obr. 59 XML návrh grafického uživatelského prostředí [zdroj vlastní] .....	88
Obr. 60 Obrazovka aplikace po kliknutí na tlačítko [zdroj vlastní].....	90
Obr. 61 Zařízení s operačním systémem Android spuštěné prostřednictvím Android Virtual Device Manageru [zdroj vlastní].....	91
Obr. 62 Testování aplikace ve skutečném telefonu, který je kabelem připojen k počítači .....	92

Obr. 63 Výpis LogCat [zdroj vlastní] .....	93
Obr. 64 Filtr LogCat [zdroj vlastní].....	94
Obr. 65 Zprávy v LogCat vytvořené programátorem [zdroj vlastní] .....	94
Obr. 66 Aplikace otočená naležato [zdroj vlastní] .....	95
Obr. 67 Ovládací prvky vyplní jen takovou výšku, kterou zabírají [zdroj vlastní] .....	96
Obr. 68 Návrh vytvořený pomocí třídy RelativeLayout [zdroj vlastní] .....	97
Obr. 69 Návrh vytvořený pomocí třídy RelativeLayout při změně klíčového ovládacího prvku [zdroj vlastní].....	97
Obr. 70 Návrh, ve kterém jsou ovládací prvky návrhu podobně velké [zdroj vlastní].....	98
Obr. 71 Návrh, ve kterém je jeden prvek větší než ostatní [zdroj vlastní] .....	98
Obr. 72 Praktická ukázka streak style [zdroj vlastní] .....	102
Obr. 73 Komponenta pro grafickou práci se soubory a adresáři [zdroj vlastní].....	104
Obr. 74 Spuštění nebo aktualizace komponenty [zdroj vlastní] .....	106
Obr. 75 Rozdělení vlákna během provádění požadované funkcionality [zdroj vlastní]....	110
Obr. 76 Část, která se nevešla na obrazovku, bude po vysunutí zpátky na obrazovku zase odbarvená/bílá [zdroj vlastní].....	112
Obr. 77 Vícenásobný výběr souborů [zdroj vlastní].....	113
Obr. 78 Jedno výběrové označování souborů [zdroj vlastní] .....	115
Obr. 79 Celkové řešení komponenty pro grafickou práci se soubory [zdroj vlastní].....	116
Obr. 80 Vytváření souborů klíčů [zdroj vlastní].....	121
Obr. 81 Vytvoření nového adresáře [zdroj vlastní] .....	122
Obr. 82 Celková správa klíčů [zdroj vlastní].....	123
Obr. 83 Celý proces šifrování [zdroj vlastní] .....	129
Obr. 84 Celý proces dešifrování [zdroj vlastní].....	133
Obr. 85 Ad hoc síť [zdroj vlastní].....	134
Obr. 86 Přenos zašifrovaných souborů mezi mobilními zařízeními a serverem [zdroj vlastní].....	135
Obr. 87 Celý proces registrace [zdroj vlastní] .....	145
Obr. 88 Komunikace mezi serverem a mobilním zařízením během procesu registrace [zdroj vlastní] .....	146
Obr. 89 Struktura souborů a adresářů po registraci [zdroj vlastní].....	147
Obr. 90 Vytvoření a odeslání archiv_RRRR.MM.DD.HH.SS.NNN.zip na server [zdroj vlastní] .....	150

---

Obr. 91 Výpis souborů a adresářů ze vzdáleného serveru [zdroj vlastní] .....	154
Obr. 92 Stahování zip archívů ze vzdáleného serveru [zdroj vlastní] .....	156
Obr. 93 Stažené zip soubory ze serveru do mobilního zařízení [zdroj vlastní] .....	157
Obr. 94 Mazání zip archívů ze vzdáleného serveru [zdroj vlastní] .....	159

**SEZNAM TABULEK**

Tab. 1 Tabulka relativních četností písmen v anglickém jazyce [3] .....	16
Tab. 2 Počet bigramů a trigramů v anglickém a českém jazyce [4][5] .....	17
Tab. 3 Vypočtená četnost programem pro frekvenční analýzu [3][zdroj vlastní] .....	20
Tab. 4 Porovnání délek výpočtů u algoritmů s různou časovou náročností [7].....	26
Tab. 5 Zvětšení rozsahu zpracovatelných dat při 100krát a 1000krát rychlejších počítačích [7] .....	28
Tab. 6 Nárůst počtu tranzistorů podle Moorova zákona [zdroj vlastní] .....	28

## SEZNAM PŘÍLOH

PI Úvodní obrazovka kryptografické aplikace

# PŘÍLOHA P I: ÚVODNÍ OBRAZOVKA KRYPTOGRAFICKÉ APLIKACE

